Universidad del Valle de Guatemala

Facultad de Ingeniería

Sistema de medición y asistencia para control de tránsito vehicular

Trabajo de graduación en modalidad de Megaproyecto presentado por
Luis Fernando Valdevallano Wurmser,
Jorge Rolando Lainfiesta Rosales,
Benito Mariano Maza Rodas y
Eddy Omar Castro Jáuregui
para optar al grado académico de Licenciados en Ingeniería en Ciencia de la Computación y Tecnologías de la Información; y
María Fernanda Martínez Rivera y
Vinicio Emanuel Gómez Pellecer
para optar al grado académico de Licenciados en Ingeniería en Electrónica.

Guatemala
2015

Sistema de medición y asistencia para control de tránsito vehicular

Universidad del Valle de Guatemala

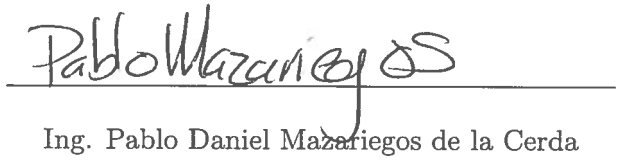
Facultad de Ingeniería

Sistema de medición y asistencia para control de tránsito vehicular

Trabajo de graduación en modalidad de Megaproyecto presentado por
Luis Fernando Valdevallano Wurmser,
Jorge Rolando Lainfiesta Rosales,
Benito Mariano Maza Rodas y
Eddy Omar Castro Jáuregui
para optar al grado académico de Licenciados en Ingeniería en Ciencia de la Computación y Tecnologías de la Información; y
María Fernanda Martínez Rivera y
Vinicio Emanuel Gómez Pellecer
para optar al grado académico de Licenciados en Ingeniería en Electrónica.

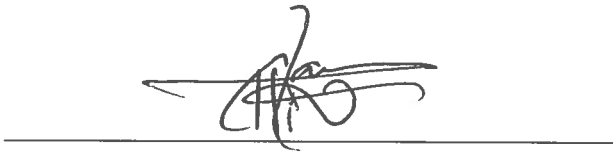
Guatemala
2015

Vo.Bo.:

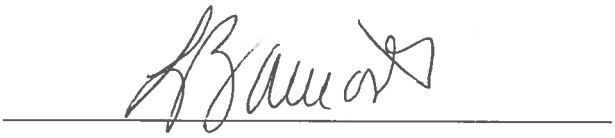


Ing. Pablo Daniel Mazariegos de la Cerda

Directores:



Ing. Carlos Alberto Esquit Hernández



Ing. Douglas Leonel Barrios Gonzales

Fecha de aprobación:

Guatemala, 25 de noviembre de 2015

Prefacio

El motivo del presente trabajo de Megaproyecto surgió de la pregunta: «¿Los agentes de la Policía Municipal de Tránsito ayudan o entorpecen el flujo vehicular?». Esta cuestión dirige inmediatamente al problema de cómo medir el estado del tráfico, y a la pregunta final: «¿Cómo mejorarlo?».

Los integrantes de este Megaproyecto incluyen estudiantes de dos departamentos de ingeniería, quienes buscan proponer las bases sobre las cuales un trabajo futuro pueda llegar a hacer realidad una solución que logre mejorar el flujo vehicular en una ciudad tan congestionada como la Ciudad de Guatemala

Índice general

Índice de figuras

Índice de cuadros

Resumen

El módulo de *Simulación de tránsito vehicular a través de un modelo de Cellular Automaton* contempla el diseño e implementación de un simulador de tránsito basado en Traffic Cellular Automata. Luego de una revisión de la literatura sobre diferentes extensiones del modelo básico de NaSch TCA para la consideración de sistemas complejos de tránsito se propone un diseño que permita escalabilidad en la creación de mapas y extensión de reglas.

El módulo de *Diseño e implementación de un algoritmo genético para la optimización de ciclos de semáforo en intersecciones de una cuadrícula* contempla el diseño e implementación de un algoritmo genético para optimizar el tránsito según dos parámetros: velocidad promedio de vehículos y tiempo de espera de vehículos. Luego de una revisión de la literatura sobre algoritmos genéticos para la optimización de ciclos de semáforos se propone una codificación de soluciones candidatas, una la estrategia híbrida de selección, y los operadores de cruce y mutación. Estos componentes se han refinado tras obtener resultados empíricos del algoritmo. La aptitud de las soluciones candidatas se evalúa ejecutando simulaciones en un simulador basado en *Traffic Cellular Automata*.

El módulo de diseño e implementación de un sistema multiagente para la optimización de tránsito se basa en un modelo de una ciudad con intersecciones de una cuadrícula y busca encontrar los mejores patrones de tiempo de semáforo para estas intersecciones utilizando un algoritmo de aprendizaje por refuerzo. Este módulo presentará el diseño de un sistema multiagente, lo cual incluye los diferentes tipos de agentes involucrados y la comunicación entre ellos, y la implementación de un algoritmo por refuerzo, donde se detalla los parámetros a optimizar, la discretización del espacio de estados del entorno y las recompensas y penalizaciones para las diferentes acciones que el agente realice.

El módulo de integración de algoritmos con Traffic Cellular Automaton, evaluación y análisis de resultados de inteligencia artificial contempla establecer el marco de trabajo para la optimización del tránsito de tal forma que se puedan obtener resultados comparables. Esto incluye la integración, por medio de interfaces, entre el módulo de Traffic Cellular Automaton con los algoritmos de inteligencia artificial implementados en otros módulos. Adicionalmente se

realizarán mediciones en cada uno de los algoritmos obteniendo datos que posteriormente serán analizados. Estos resultados también se contrastarán con algoritmos adicionales que serán implementados basándose en enfoques aleatorios. El análisis a realizar contempla determinarla eficiencia de cada algoritmo en diferentes situaciones.

I. INTRODUCCIÓN

El tránsito es un problema global, particularmente caótico en ciudades como Guatemala, en la que la infraestructura es insuficiente para el parque vehicular. Como un paliativo, la Municipalidad de Guatemala ha optado por conformar y fortalecer la Policía Municipal de Tránsito (PMT), incluyendo entre sus tareas el dirigir el tránsito en horas de particular congestionamiento. Mientras cumplen esta tarea, se apagan los semáforos y el flujo de vehículos queda en manos de los agentes, quienes deben hacer proezas para determinar cuándo dar vía a una calle o a otra, además de intentar coordinarse con agentes en otras intersecciones cercanas por radio. De ahí surgió la motivación de este megaproyecto: brindar herramientas para empoderar a los agentes y ayudarles a tomar decisiones sobre el tránsito.

La idea es ambiciosa al considerarla en todos sus componentes, variables e implicaciones. Por lo que en este megaproyecto se busca proponer bases en dos frentes: hardware y software. En el primero, se presenta un prototipo de la infraestructura que tendría que adoptar la ciudad para sensores de tránsito, agentes de PMT y comunicación de estos con un sistema centralizado. En el segundo, se explora el diseño, implementación y comparación de dos algoritmos de inteligencia artificial para optimizar el tránsito; además, se propone un simulador basado en *Traffic Cellular Automata*, que será la herramienta utilizada para brindar información a los algoritmos de inteligencia artificial.

El sistema de medición y asistencia de control de tránsito se encuentra en su primera fase. Se pretende que en la fase final se mida la velocidad de los vehículos en los carriles de calles principales de la ciudad de Guatemala. En esta primera fase, se encapsulan y transmiten los datos que se supone provienen de los sensores, y se almacenan en una base de datos local. Además, se desarrolla el sistema que se encarga de transmitir la información resultante del sistema de análisis hacia el dispositivo portátil, permitiendo a este último que notifique sucesos que perjudican el tránsito vehicular.

Este megaproyecto consiste en el desarrollo de un prototipo de una herramienta de apoyo para los policías municipales de tránsito, no del control automático de dispositivos electrónicos relacionados al flujo vehicular. Esto es, el proyecto se limita a la implementación del sistema de sensado, dispositivo portátil, y comunicación entre los diversos módulos. Se seleccionó un medio de comunicación y sistemas de control de bajo costo y que permitan el funcionamiento adecuado del sistema total.

II. OBJETIVOS

A. OBJETIVO GENERAL DEL MEGAPROYECTO

Crear un prototipo de un sistema de detección de tránsito vehicular cuyas mediciones sean transmitidas y almacenadas en un módulo de control, el cual se encarga de comunicar instrucciones a un dispositivo portátil que despliega la información al policía de tránsito para mejorar la gestión del flujo vehicular.

III. JUSTIFICACIÓN

Se estima que entre el año 2005 y 2015 el parque vehicular ha crecido un 154 % en Guatemala, alcanzando una concentración de 1 millón 247 mil 657 vehículos en el Departamento de Guatemala, lo cual representa el 45 % del total del país. (Fernández , 2015) Esto ocasiona graves embotellamientos en la Ciudad de Guatemala, cuya infraestructura solo tiene capacidad para 350 mil vehículos circulando diariamente. Debido a que se estima que 1 millón 100 mil vehículos transitan la ciudad al día , como indica Rodríguez (2015), la Municipalidad de Guatemala, entre otras medidas, ha enfocado sus esfuerzos en el crecimiento de la Policía Municipal de Tránsito (PMT). Esta entidad se encarga, entre otras funciones, de dirigir el tránsito a horas especialmente críticas de embotellamiento en áreas carentes de semáforos y reemplazando a los mismos en intersecciones donde están presentes.Utilizan métodos heurísticos basados en experiencias de agentes particulares y carecen de datos suficientes para construir estrategias para la regulación del tránsito. Por otro lado, el comportamiento de los conductores en la ciudad de Guatemala es particular y no se apega a las normas o convenciones establecidas, por lo que requiere un sistema adaptado a las prácticas locales.

La municipalidad de Guatemala provee de policías de tránsito que intentan dar solución al problema. Actualmente se posee cámaras, vehículos, radios, entre otros recursos. Pero el trabajo cada vez es más complicado debido a la sobrepoblación de vehículos y la ausencia de parámetros objetivos en la toma de decisiones para la gestión del flujo vehicular. Con este mega proyecto se pretende desarrollar un prototipo de lo que podría llegar a ser una ayuda para los policías de tránsito, facilitando la toma de decisiones para que haya mayor flujo de vehículos. La información recopilada por los sensores se encuentra disponible para cualquier sistema de análisis cuyo resultado son las recomendaciones a brindar al agente de tránsito.

IV. ANTECEDENTES

A SIMULADORES DE TRÁNSITO

Rosenblueth, *et al.* (2011) identifica que en observaciones empíricas de tránsito en carreteras, es posible distinguir dos diferentes regímenes. Para bajas densidades de vehículos, el flujo muestra tener un comportamiento aproximadamente lineal. Sin embargo, para densidades mayores el flujo presenta fluctuaciones abruptas que resultan en un comportamiento complejo que no ha sido comprendido en su totalidad (Chowdhury, *et al.* , 2000). Debido a estas fluctuaciones no es posible utilizar un modelo funcional y adicionalmente, estados metaestables han sido observados.

Como resultado de esta complejidad, una gran cantidad de modelos de tránsito en carreteras puede encontrarse en la literatura. Chowdhury, *et al.* (2000) hacen un recorrido de estos modelos a través de las diferentes aproximaciones *macroscópicas* y *microscópicas*, sin embargo concluye que en ocasiones las ecuaciones fenomenológicas del flujo vehicular en los modelos macroscópicos pueden ser obtenidas de consideraciones microscópicas en la misma forma en que teorías macroscópicas de la materia son derivadas de sus descripciones teorético-moleculares.

Dentro de las aproximaciones microscópicas, las teorías cinemáticas modelan el tránsito como un gas en donde cada partícula representa un vehículo. La clase de modelos *follow-the-leader* representa a cada vehículo con una ecuación de movimiento en un sistema de partículas clásicas que interactúan entre sí. Los modelos de *Coupled-map-lattice* tratan el tiempo como una variable discreta y las ecuaciones dinamicas para cada vehículo se convierten en un mapa discreto-dinámico. Finalmente, los modelos de CA *(cellular automata)* juegan un rol prominente debido principalmente a su bajo costo computacional a comparación de los otros modelos microscópicos.

Los *traffic cellular automata* (TCA) son sistemas dinámicos discretos en tiempo y espacio cuya principal ventaja es un alto rendimiento y flexibilidad, basado en una baja precisión a nivel microscópico pero que puede generar mediciones macroscópicas realistas.

B CELLULAR AUTOMATA

Los orígenes de cellular automata pueden rastrearse hasta 1950, en que Von Neumann publicó los primeros conceptos de cellular automata tras su estudio de sistemas biológicos vivientes. En 1970 se po-

pularizaron ’juegos de simulación’ basados en cellular automata, siendo ”Game of Life”de Conway uno de los más conocidos. En 1980 se extendió el uso de cellular automata a una extensa clasificación de modelos estadísticos. Además se dio la publicación de *A New Kind of Science*, por Stephen Wolfram, en el que se aplicaban conceptos de cellular automata a múltiples disciplinas como la sociología, biología, física, entre otras. Un paso importante para la continuidad de la teoría de Cellular automata fue la demostración de Bill Gosper en que probaba que Game of Life es computacionalmente universal y por tanto puede imitar un algoritmo arbitrario (Maerivoet *et al.* , 2005).

V. SIMULACIÓN DE TRÁNSITO VEHICULAR A TRAVÉS DE UN MODELO DE CELLULAR AUTOMATON

A INTRODUCCIÓN

En el módulo de *Simulación de tránsito vehicular a través de un modelo de Cellular Automaton* se busca diseñar e implementar un sistema de simulación de tránsito basado en Traffic Cellular Automata. El producto debe poder ser utilizado por otros módulos de este megaproyecto para brindar datos a algoritmos de inteligencia artificial y poder a la vez comparar los resultados de ambos algoritmos aplicados en una simulación. El sistema debe ser también eficiente en el consumo de recursos computacionales y mantener un diseño que le permita ser extensible en la capacidad de mapas que pueda modelar.

B OBJETIVOS DEL MÓDULO

1 Objetivo general del módulo

Implementar un simulador de tránsito basado en Traffic Cellular Automata que represente una cuadrícula de calles y avenidas de doble carril con semáforos en las intersecciones.

2 Objetivos específicos del módulo

- Diseñar un sistema que permita la implementación de Traffic Cellular Automata capaces de modelar diferentes tipos de mapas.
- Definir un conjunto de reglas de Traffic Cellular Automata que extiendan las básicas para permitir calles multicarril e intersecciones.
- Implementar un simulador que aplique las reglas definidas y que sea computacionalmente eficiente.
- Implementar un visualizador que permita validar visualmente el funcionamiento de las reglas implementadas.

C JUSTIFICACIÓN DEL MÓDULO

Para poder realizar un análisis propio y el desarrollo de las diferentes implementaciones de algoritmos inteligentes, es necesario el módulo que contempla la creación de un simulador a través de la implementación de un Traffic Cellular Automaton—TCA—. Esto incluye la investigación e implementación de los elementos del TCA, así como la creación de reglas que determinen el comportamiento celular del mismo.

La implementación del TCA es vital para poder tener indicadores que provean noción sobre el comportamiento de los automóviles como un todo dentro de una cuadrícula de calles y avenidas. Mediante este módulo es posible evaluar el beneficio que un algoritmo puede influir en el flujo vehicular de una ciudad.

D DELIMITACIÓN

El simulador implementado contemplará unicamente calles de dos carriles y de una vía, con intersecciones sencillas entre ellas. A pesar de ello el simulador será capaz de ser extendido en un trabajo futuro.

El conjunto de reglas que serán implementadas será delimitado para que el comportamiento del vehículo se aproxime a un comportamiento normal. Esto incluye las reglas básicas del modelo de Nagel-Schreckenberg de aceleración, evasión de colisiones y desaceleración estocástica. También se contemplará el diseño e implementación de reglas que permitan un cambio de carril y cruces en intersección, contemplando el semáforo. Estas reglas simulan decisiones que puede tomar un conductor, aplicando estocasticidad en cada una de ellas. Los vehículos serán generados con valores aleatorios en constantes para que tengan cada uno personalidades de manejo distintas y simulen así la agresividad o falta de la misma que puedan tener.

E MARCO TEÓRICO

1 Traffic Cellular Automata.

En esta sección se describen las nociones básicas de un *Cellular Automaton*, que consituye el fundamento teórico y metodológico de un *Traffic Cellular Automaton* (TCA). Luego se explora la adaptación de conceptos de un modelo de *Cellular Automata* al contexto del tránsito. Finalmente se revisan técnicas para realizar evaluaciones macroscópicas en un *Traffic Cellular Automaton*.

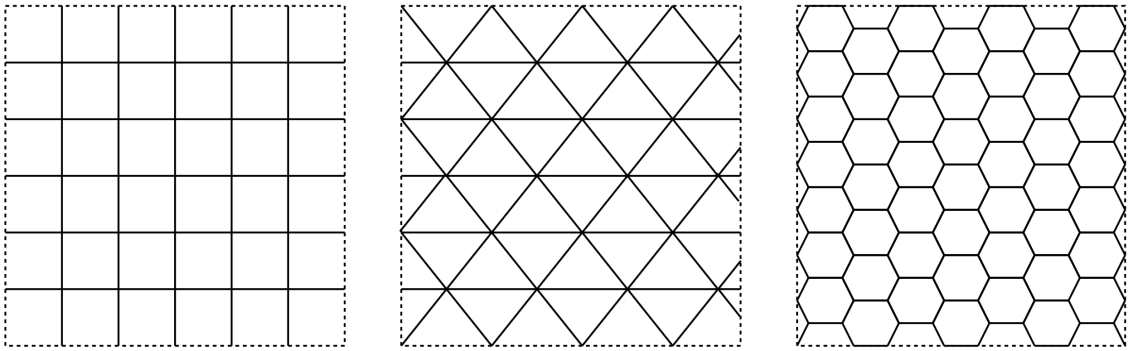
a Cellular Automata.

Desde un punto de vista teórico, los cellular automata tienen cuatro elementos fundamentales según sus primeras definiciones (Maerivoet *et al.* , 2005):

Ambiente físico define el universo en el que se computa el cellular automata. Este consiste en una estructura discreta, algunos ejemplos de muestran en la Figura 1.

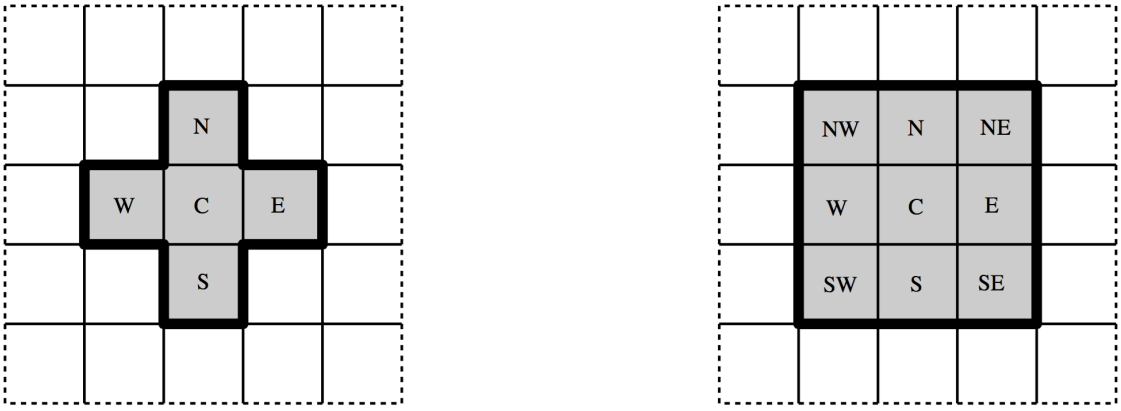
Estados de las celdas Cada celda puede tener un estado, típicamente representado por un entero o un bi-

Figura 1. Ejemplos de diferentes topologías euclidianas para un cellular automaton de dos dimensiones. *Izquierda:* rectangular. *Medio:* triangular/isométrico. *Derecha:* hexagonal.



(Maerivoet *et al.* , 2005)

Figura 2. Dos tipos de vecindarios comunes con radio 1. *Izquierda:* Vecindario de Von Neumann. *Derecha:* Vecindario de Moore.



(Maerivoet *et al.* , 2005)

nario. Sin embargo, los estados también pueden ser descritos de formas más complejas. Al colectivo de estados de todas las celdas de le conoce como *configuración global del cellular automaton*.

Vecindario de la celda Para cada celda se define un vecindario que define la evolución local de la misma. En la Figura 2 se muestran los dos tipos de vecindarios más comunes en dos dimensiones: el vecindario de Von Neumann y el de Moore. En el primero consiste en la celda central más cuatro adyacentes. El segundo está compuesto por 8 celdas adyacentes.

Regla local de transición Esta regla actúa en una celda y su vecindario, de modo que el estado de las celdas cambian en pasos discretos de tiempo. Un cellular automaton evoluciona en el tiempo y en el espacio mientras las reglas se aplican paralelamente en las celdas. Las reglas pueden tener componentes estocásticos o ser deterministas.

Al aplicar los elementos de un cellular automaton en el contexto del tránsito el ambiente físico repre-

senta la estructura de las calles, en la que una celda puede tener un vehículo y las reglas de transición determinan el comportamiento de los vehículos (Nagel , 1992).

b Componentes de un TCA. Cuando se aplican los conceptos de los cellular automata al universo de calles y vehículos en una ciudad o un sistema de carreteras es posible obtener información sobre el flujo vehicular en diferentes puntos y escenarios. Según describe Maerivoet *et al.* (2005), en un modelo de traffic cellular automaton clásico se consideran los elementos de los cellular automata como sigue:

Topología de calle El ambiente físico del cellular automaton representa la calle o el conjunto de calles sobre las que los vehículos transitan. Nagel (1992) define un modelo en el que se constituye una única calle de un sólo carril que puede tener los extremos abiertos o representar un circuito cerrado. Este modelo y los que se derivan del mismo definen una cuadrícula de celdas en una dimensión en donde cada celda puede o no contener un vehículo. Existen variaciones a estos modelos donde las calles pueden tener varios carriles y los vehículos ocupar varias celdas a la vez.

Vehículos Los estados de las celdas en un TCA se definen como la presencia o ausencia de un vehículo en ese espacio físico. Los vehículos tienen una velocidad representada por el número de celdas que pueden avanzar en el tiempo de una iteración del sistema.

Reglas de conducción Al tratarse de sistemas microscópicos, los cambios en la configuración de los traffic cellular automata se producen a partir del cálculo de reglas locales de transición que se aplican a cada vehículo. De modo que tales reglas definen de una forma burda el movimiento de los carros como moléculas, pero que en la visualización macroscópica del sistema permiten crear una aproximación realista del comportamiento del flujo vehicular. (Benjaafar , 1997)

c Evaluaciones macroscópicas de TCA. Los modelos de *traffic cellular automata* son capaces de reproducir correctamente comportamientos del tránsito basándose en las descripciones microscópicas descritas por las reglas de transición. Debido a esta característica, resulta válido extraer datos macroscópicos tanto locales como globales de diferentes detectores colocados dentro de un simulador de *traffic cellular automata*. Estas mediciones pueden incluir flujos, velocidades, distancias y densidades. (Maerivoet *et al.* , 2005)

Evaluaciones locales Las evaluaciones locales regularmente son realizadas por medio de un detector colocado en un sector específico de un carril. Este detector utiliza el intervalo de tiempo del *traffic cellular automata* como indicador para realizar la medición, por ejemplo en cada actualización de tiempo se evalúa si hay un vehículo dentro del sector especificado y la velocidad a la que se desplaza en caso se encuentre uno. (Maerivoet *et al.* , 2005)

Evaluaciones globales Las evaluaciones globales consideran *N* vehículos manejando en el sistema que contiene *M* celdas, por lo que la densidad *N/M* se mantiene constante durante todo el período de la medición. Este tipo de evaluaciones son viables únicamente cuando el simulador de *traffic cellular automata* sobre el cual se están realizando las mediciones genera tantos vehículos como los que consume dentro del sistema. (Maerivoet *et al.* , 2005)

Las evaluaciones macroscópicas de *traffic cellular automata* son útiles para generar estadísticas como la densidad global vehicular, número de vehículos que ha ingresado o velocidad promedio a la que se mueven los vehículos del simulador. Estas mediciones, como se mencionó, se realizan a nivel microscópico por lo que es necesario contar con un mecanismo de comparación con el tránsito real. En la comparación realizada por Nagel (1992) se utiliza una escala de longitud basándose en que al existir congestionamiento completo un vehículo promedio utilizaría aproximadamente 7.5 *m*, por lo tanto toman esta medida como la longitud de celda.

De forma similar se calcula el tiempo real que toma cada iteración de la simulación, para esta comparación Nagel (1992) toma como base la velocidad promedio a la que un vehículo circula en Alemania (120 *km/h*) y se compara con la velocidad promedio a la que un vehículo se mueve en el simulador cuando el carril está completamente libre (4.5 celdas por iteración). Al realizar las conversiones correspondientes, se llega a la conclusión que este modelo tomar alrededor de 1 segundo por cada iteración.

Existen otros métodos para estimar el tiempo que toma una iteración, por ejemplo considerando la capacidad máxima de las carreteras para el paso de vehículos por hora en cada carril o la cantidad de vehículos que pueden transitar simultáneamente en un kilómetro de carril de una carretera. Todas estas comparaciones tienen como base el hecho que la longitud de la celda es similar a la de un vehículo promedio por lo que se encuentra bastante relacionada a las capacidades del simulador respecto a diferentes longitudes de vehículos. (Nagel , 1992)

Las evaluaciones en los modelos de *traffic cellular automata* se basan en dos posibles condiciones iniciales que dependerán del estudio que se desea realizar. La configuración utilizada como base es la de *homogeneous initial conditions* que distribuye los vehículos de manera uniforme lo que implica que la separación entre un vehículo y otro será igual. La otra condición inicial, *compact superjam*, coloca vehículos en todo el simulador de tal forma que no existan separaciones entre un vehículo con otro. Para mantener estas condiciones a lo largo de la simulación, los métodos generadores y consumidores de vehículos deben ser consistentes con el estado en el que se encuentra el sistema. (Maerivoet *et al.* , 2005)

2 Clasificación de modelos.

El estudio de traffic cellular automata ha sido abordado por diferentes autores que han definido modelos con características y definiciones distintas. Esta diferenciación en los aspectos que puede tener un modelo permite clasificarlo bajo diferentes categorías. Los sistemas pue-

den ser unicelulares determinísticos y estocásticos, multicelulares y de múltiples carriles; la actualización de los mismos también puede darse de forma síncrona o asíncrona.

a Resolución del modelo. Los modelos se pueden clasificar como unicelulares o multicelulares. El término está asociado a la cantidad de celdas que hacen falta para ocupar el espacio de un vehículo. La principal diferencia entre ambas definiciones radica en la diversidad de vehículos que existen dentro del TCA, como es planteado por Maerivoet *et al.* (2005).

Modelo unicelular Bajo esta descripción, cada celda del sistema puede estar vacía u ocupada por exactamente un vehículo y todos los vehículos tienen la misma longitud de $l_i = 1$ celda. El tráfico también es considerado homogéneo, de forma que todas las características propias de los vehículos se toman como iguales.

Modelo multicelular A diferencia del anterior, en un modelo descrito como multicelular un vehículo es permitido a ocupar un número de celdas consecutivas en la dirección longitudinal, de forma que pueden exisir vehículos con distintas longitudes y sus acciones de aceleración y desaceleración tienen una más real representación.

b Estocastidad. Los modelos se pueden clasificar por el nivel de aleatoriedad que introduzcan en su simulación.

Modelo determinístico Las reglas que gobiernan este tipo de modelos sólo incluyen aleatoriedad en la definición inicial del sistema, de forma que una configuración inicial de vehículos y sus correspondientes velocidades alcanza rápidamente un patrón estacionario que se desplaza de forma uniforme en el sentido contrario a la dirección de los vehículos. (Nagel , 1992)

Modelo estocástico Estos modelos incorporan explícitamente reglas de aleatoriedad en sus procesos de cálculo de los cambios en la configuración del sistema de forma que permiten que fluctuaciones naturales ocurran en las velocidades de los vehículos debido a comportamientos humanos o a condiciones externas variantes. Estos modelos permiten la formación espontánea de colas fantasma en la naturaleza del tráfico. (Maerivoet *et al.* , 2005)

c Consideración de múltiples carriles. Los modelos de un sólo carril no son capaces de modelar tránsito real debido a una razón: un conjunto realista de vehículos es usualmente compuesto por vehículos que mantienen diferentes velocidades deseadas. Introducir esta variedad de vehículos en el modelo de un sólo carril resultaría en la formación de grupos de vehículos donde los más lentos serían seguidos por los rápidos que no podrían acelerar a su velocidad deseada. (Rickert *et al.*, 2008)

La introducción de multiples carriles en las calles de los modelos de TCA nos induce a identificar los diferentes tipos de cambios de carril que pueden representarse. Maerivoet y De Moor (2008) identifica dos tipos: *cambios de carril mandatorios* (MLC) y *cambios de carril discrecionales* (DLC).

Cambios de carril mandatorios Un vehículo puede verse obligado a realizar un cambio de carril por diferentes razones; por ejemplo, debido a que el vehículo debe tomar una salida de la carretera o por que la ley le indica que debe transitar en el carril más a la derecha siempre que sea posible, como es el caso en varios países europeos.

Cambios de carril discrecionales Un vehículo cambia de carril a su propia discreción, por ejemplo al aproximarse y rebasar a un vehículo que se moviliza con lentitud.

Esser (1997) también realiza una comparación sobre las reglas que definen un cambio de carril y las clasifica como cambios riesgosos o controlados, siendo los primeros aquellos donde no se consideran distancias de seguridad al cambiar de carril y pueden ocasionar un cambio abrupto en la velocidad de los vehículos consecutivos en dicho carril.

d Paralelismo del sistema. El proceso de cálculo y actualización del sistema puede ocurrir de forma síncrona o asíncrona dependiendo del momento en que los vehículos realicen el cálculo y su consecuente movimiento dentro de este proceso. Ambas formas de actualización proveen resultados muy similares. (Maerivoet *et al.* , 2005)

TCA síncrono Se describe como una actualización paralela en donde cada vehículo raliza su decisión basado en la configuración general del automaton al inicio del intervalo de tiempo.

TCA asíncrono Una actualización secuencial se puede realizar en un orden aleatorio sobre el conjunto de vehículos que se encuentran en el automaton, de esta forma la configuración del automaton está cambiando constantemente.

3 Modelo de Nagel-Schreckenberg. El modelo propuesto por Kai Nagel y Michael Schreckenberg (Nagel , 1992), conocido como *NaSch TCA*, fue originalmente definido en una calle de un solo carril. La calle se divide en celdas, que pueden estar vacías u ocupadas por un vehículo. Cada vehículo tiene un valor entero no negativo de velocidad. Para una actualización de la calle, los siguientes cuatro pasos son ejecutados de forma simultánea para todos los vehículos:

1. Aceleración: $v \longrightarrow \text{mín}(v+1, v_{max})$
2. Desaceleración debido a otros vehículos: $v \longrightarrow \text{mín}(v, gap)$

3. Aleatorización: if $rand() < p_{dec}$ then $v \longrightarrow \text{máx}(v-1,0)$

4. Actualización: Cada vehículo avanza v celdas en la calle.

En estas instrucciones, gap se refiere al número de celdas vacías al frente de un vehículo y v_{max} a la velocidad máxima. La aleatorización (paso 3) toma en cuenta que los comportamientos individuales de conducción de diferentes vehículos resultan en dinámicas no determinísticas en la vida real (Esser , 1997).

4 Extensión del modelo de *NaSch-TCA* para modelación de múltiples carriles e intersecciones.

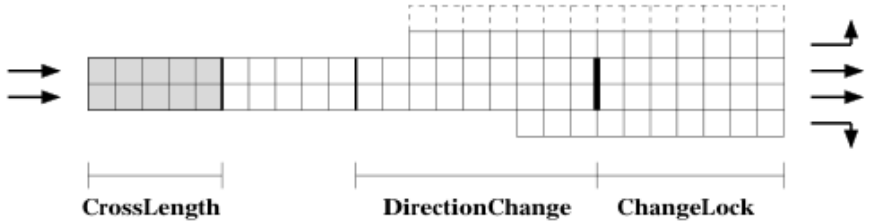
Esser (1997) se basan en el modelo de Nagel-Schreckenberg para crear un modelo para redes de calles en entornos urbanos. En su trabajo describen la red de calles como una composición de nodos y aristas que representan cruces y calles en un grafo dirigido. Definen diferentes tipos de aristas.

Aristas de un carril En estas aristas, simplemente se aplica el *cellular automaton* básico de *NaSch*. Al final de la arista pueden haber secciones de cruce para las diferentes direcciones. Los vehículos que utilizan estas secciones se movilizan al inicio de la misma y si el vehículo es obstaculizado para entrar, debe esperar en una posición predeterminada hasta que el cambio sea posible.

Aristas multi-carril Especialmente al final de las calles multi-carril, el comportamiento de cambios de carril depende en gran medida de la dirección que el vehículo desee tomar. Para este propósito, estas aristas son divididas en diferentes regiones, las cuales se muestran como ejemplo en la Figura 3. La figura muestra una arista de dos carriles que tiene dos secciones de cruce agregadas a los lados. En la sección inicial, *CrossLength*, los cambios de carril son libres. En la siguiente sección, *DirectionChange*, sólo los cambios acordes a la dirección de destino son permitidos y la probabilidad de realizar cambios de carril riesgosos, donde sólo la celda vecina es revisado, aumenta a medida que se aproxima a la última sección. En la sección *ChangeLock* los vehículos se detienen hasta que logran cambiar de carril si no se encuentran en el carril de la dirección deseada. En este momento puede ocurrir un *deadlock* si dos vahículos se obstaculizan entre sí para realizar cambios de carril, en ese caso se intercambian de celda para eliminar el *deadlock*.

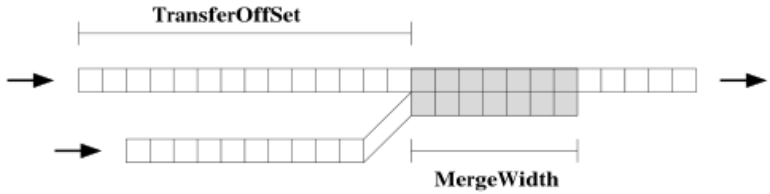
Aristas de transferencia Usualmente las carreteras juegan un rol importante en la descripción de tránsito urbano. Por ello, este elemento es incorporado para simular los acoplamientos a carreteras e intersecciones con gran extensión espacial. Las aristas de transferencia son aristas de uno o varios carriles que se unen a una arista de destino (Fig. 4). Los vehículos se movilizan hasta llegar a la sección de unión, *MergeSection*, donde pueden cambiar de carril hacia la arista de destino bajo la condición de que exista suficiente distancia al sucesor en la arista de destino. Si son obstaculizados, se mueven hasta el final de la calle de transferencia y de ser necesario esperan. En la Figura 5 se muestra una intersección como combinación de diferentes tipos de aristas.

Figura 3. Estructura de una arista de dos carriles con secciones de cruce adicionales.



(Esser , 1997)

Figura 4. Estructura de una arista de transferencia para representar acoplaciones a carreteras e intersecciones con una extensión espacial grande.



(Esser , 1997)

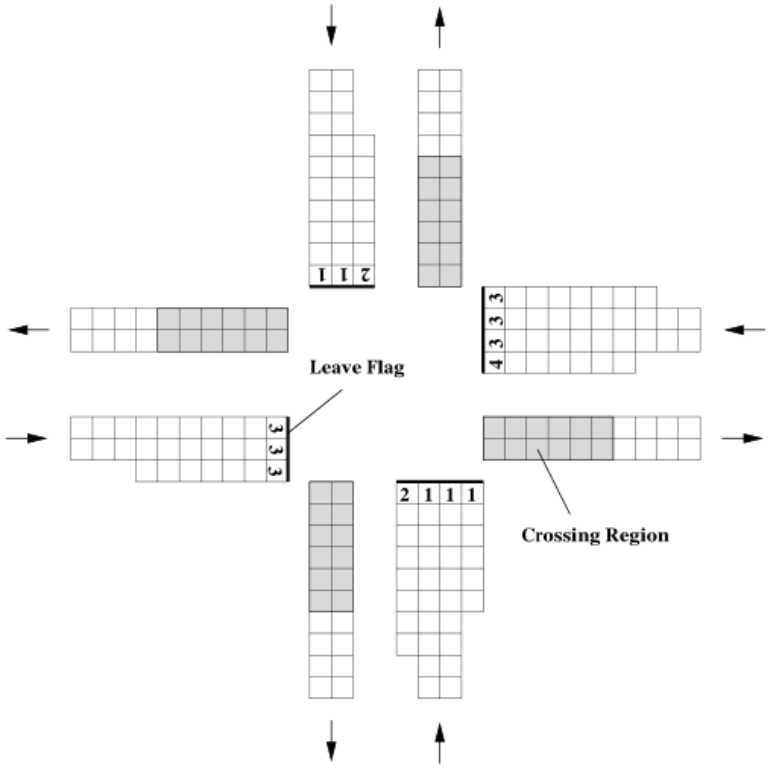
F METODOLOGÍA

Para el presente módulo se diseñará e implementará un simulador de tránsito vehicular en una zona urbana. El simulador se basará en el modelo de Traffic Cellular Automata de Nagel-Schreckenberg y aumentará el sistema de reglas para permitir la modelación de calles con múltiples carriles e intersecciones con semáforos. También será implementado un visualizador utilizando la librería PyGame para poder observar en tiempo real los cambios en el simulador implementado. A continuación se describe el diseño y los detalles de los componentes principales del simulador.

1 Implementación de TCA. Para el presente trabajo fue necesaria la implementación un Traffic Cellular Automaton—TCA— capaz de simular el comportamiento del tráfico en mallas de calles y avenidas de dos carriles cada una. Para lograrlo se tomó como base el modelo de Nagel-Schreckenberg y se extendieron las reglas y el modelo para permitir múltiples carriles e intersecciones. Estas extensiones del modelo *NaSch TCA* tomaron como base las mismas exploradas por Esser (1997) haciendo algunas modificaciones que se consideraron oportunas. El diseño también está considerado para que el modelo sea extensible.

La implementación del TCA se realizó basada en el paradigma de programación orientada a objetos.

Figura 5. Representación de cruces.



(Esser , 1997)

Estableciendo como uno de los principales requerimientos el rendimiento del sistema, el sistema cumple con ser un conjunto de objetos que interactúan entre sí realizando únicamente operaciones sencillas en sus cálculos.

Un objeto Automaton tiene dentro de sí una topología, una lista de carros y un método `update()`. La topología, por su parte, está compuesta por un conjunto de celdas de diferentes tipos que se encuentran referenciadas entre sí para formar calles e intersecciones. Durante el método `update()` del Automaton se instancian y ejecutan las reglas para cada carro que se encuentre dentro del simulador en ese momento. En las siguientes secciones se listan y explican los diferentes componentes que fueron diseñados. La Figura 6 muestra las relaciones ente los diferentes componentes del simulador.

a Automaton. El objeto `Automaton` representa una instancia del TCA. Su principal función es la de orquestar las instrucciones que deben seguirse en cada iteración del mismo. Este conjunto de instrucciones se realizan en un orden específico y fueron implementadas dentro de un método `update()`. El objeto guarda la instancia de una topología que define el entorno por el que los carros van a movilizarse.

El método `update()` realiza cuatro pasos para la actualización del automaton, como se indica en la

Figura 6. Diagrama general del diseño del simulador.

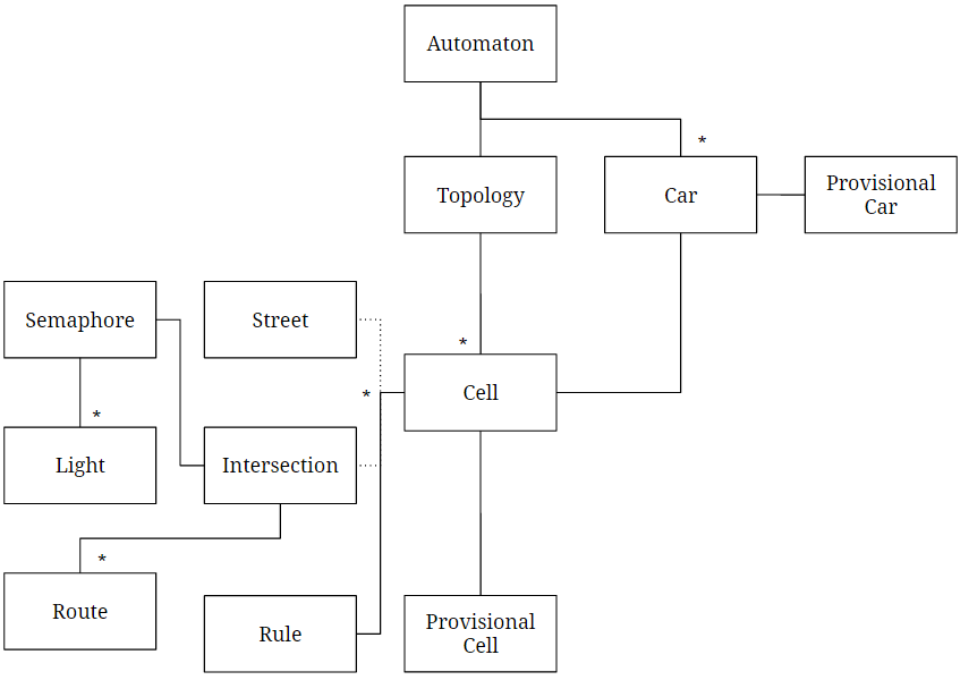


Figura 7(A):

1. **Generación y consumo de carros:** Se realiza un recorrido por todos las celdas limítrofes del mapa, es decir que pertenezcan a la lista `endpoint_cells` de la topología, y se aplican de manera asíncrona las reglas para la generación y el consumo de carros en esas celdas específicas.

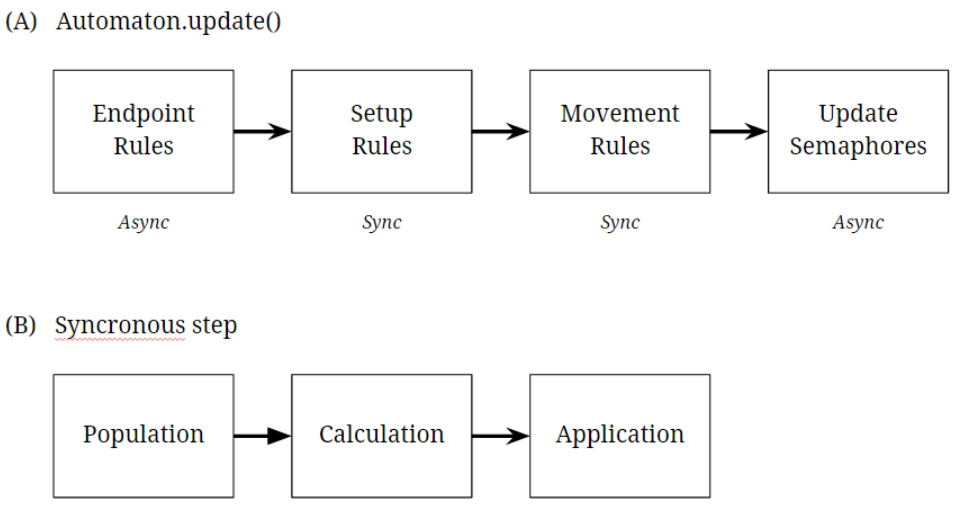
2. **Configuración del sistema:** Se realiza un recorrido por todas las celdas de la topología y se aplican las reglas definidas para el inicio del ciclo de `update()`, que definen los cambios que pueden ocurrir en el sistema previo a la movilización de los vehículos. Las reglas de cambio de carril se aplican en este momento.

3. **Movimiento de carros:** Se aplican las reglas básicas del movimiento de vehículos a todos los existentes dentro de la topología.

4. **Actualización de semáforos:** En todos los semáforos existentes en el sistema, se verifica si se debe hacer un cambio de las luces, según el cronograma asignado en cada uno de ellos.

Los pasos 2 y 3 del método se realizan de una forma síncrona en el sistema, mientras que los otros dos se realizan de forma asíncrona. Esto quiere decir que los primeros requieren que las reglas sean calculadas y aplicadas al mismo tiempo en todas las celdas para que los cambios de una no interfieran en el cálculo de

Figura 7. (A) Diagrama de flujo que sigue el simulador en la operación `update()` del Automaton y (B) diagrama de flujo que sigue un proceso síncrono de actualización de estados.



las reglas de otra. A diferencia de estos, en los pasos asíncronos se pueden realizar los cambios al mismo tiempo que se recorre la lista de objetos en los que se aplican, debido a que no afectan los cambios de uno en el otro. En la Figura 7(B) se muestra el ciclo necesario para el proceso de aplicación de reglas en los pasos síncronos.

b Topología. La clase `Topology` define la estructura que tiene un mapa sobre el cual corre el Automaton. Como principal función, almacena el conjunto de celdas que componen un mapa, así como las calles e intersecciones que estas representan.

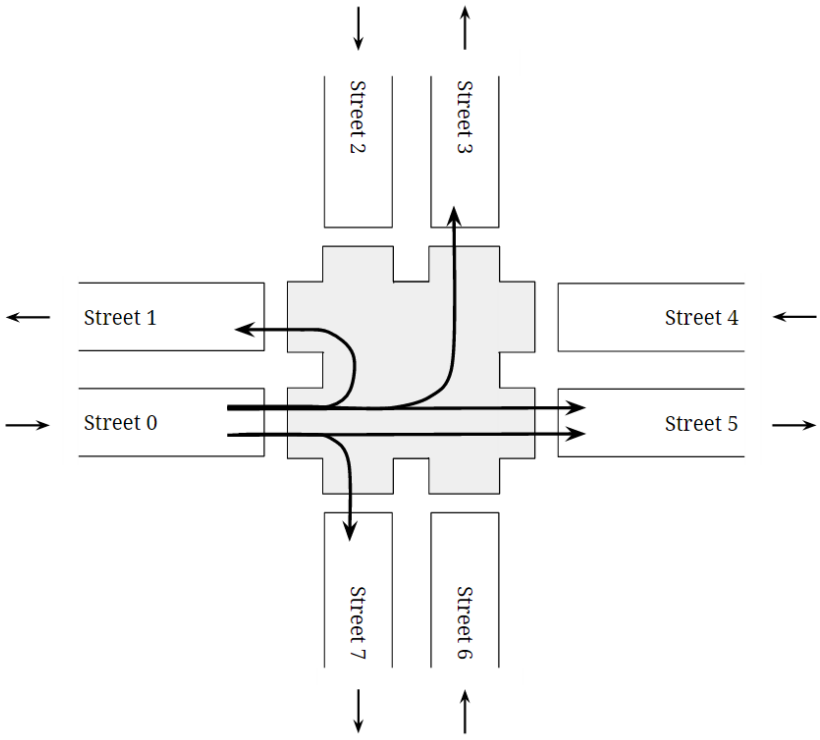
Celdas limítrofes A las celdas que se encuentran en los límites del mapa se les conoce como celdas de conexión o limítrofes. Estas celdas pueden ser generadoras o consumidoras, lo cual indica las reglas especiales que se les aplican en el primer paso del ciclo de `update()` del automaton.

Calles Las celdas dentro de la topología pueden definir calles formadas a partir de ellas. Estas calles guardan información importante para el cálculo de las reglas en las celdas que la definen, como lo son el ancho y largo de la calle y las rutas de salida. Las rutas de salida determinan el comportamiento de los vehículos en el cambio de carriles durante su recorrido por tal calle, debido a que buscan ubicarse en un carril dependiendo de la ruta de salida seleccionada.

Intersecciones y rutas Las celdas que se encuentran en los espacios que quedan entre las calles son conocidas como celdas de intersección y, de la misma forma que las de calle, definen objetos de intersección para contener información del mapa en ellos. Un objeto de intersección contiene dentro de sí las rutas posibles y el semáforo que rige los tiempos de las luces en la misma. Una ruta es una sucesión

de celdas dentro de una intersección que determinan una trayectoria permitida para la entrada y salida de un vehículo en una intersección. En la Figura 8 se muestra un ejemplo de intersección.

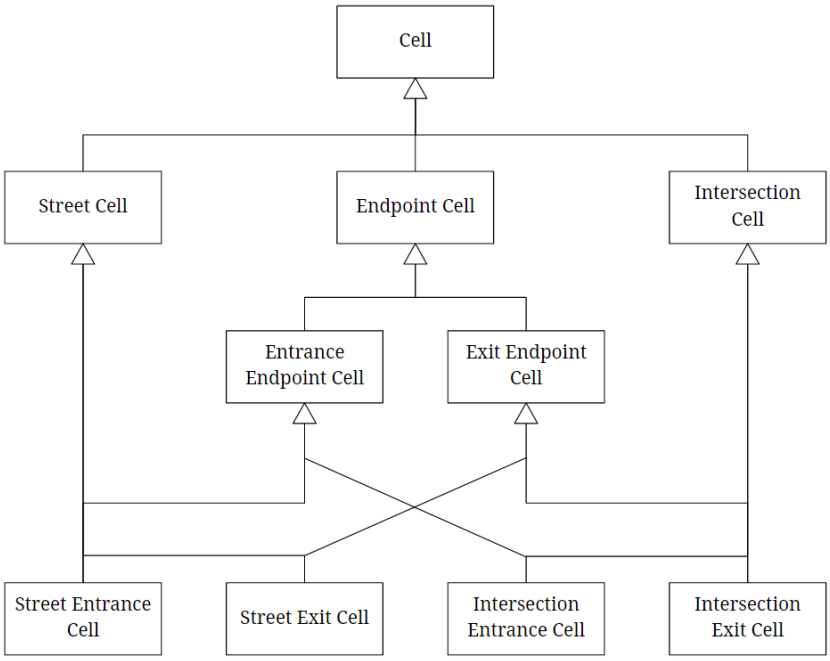
Figura 8. Estructura de una intersección de 8 calles con 5 rutas que salen de la calle 0 a manera de ejemplo.



c Celdas. La clase `Cell` es la principal en el diseño del Traffic Cellular Automaton. En ella se guarda toda la información relevante para las reglas que sean aplicadas a los carros que se encuentren sobre ella, incluyendo el conjunto mismo de reglas que será aplicado. Para describir los diferentes tipos de celdas que pueden existir se optó por definir un sistema de herencias. La taxonomía de los diferentes tipos se muestra en la Figura 9. El objeto `Cell` contiene dentro suyo un vehículo que puede ser `None` si no hay un vehículo sobre ella, una clase que hereda de `Rule`, la cual se instancia para calcular las reglas en cada ciclo de `update()` y una instancia de `ProvitionalCell`, la cual es utilizada para realizar las actualizaciones de forma síncrona. Los diferentes tipos de celda pueden implementar el método `get_front_cells(n, route)` para indicar una serie de n celdas que se encuentran delante de la celda actual, cuando el vehículo sigue una ruta (*route*) determinada.

Celdas de calle Contiene información de la calle a la que pertenece, así como el carril y el número de celda en el que se encuentra ubicada. También identifica cuáles son las celdas frontal y laterales e indica que la clase `StreetRule` debe ser implementada para el cálculo de las reglas. Se implementa el método de `get_front_cells(n, route)` que retorna las n celdas que se encuentren delante en el

Figura 9. Taxonomía de los diferentes tipos de celdas que existen en una topología.



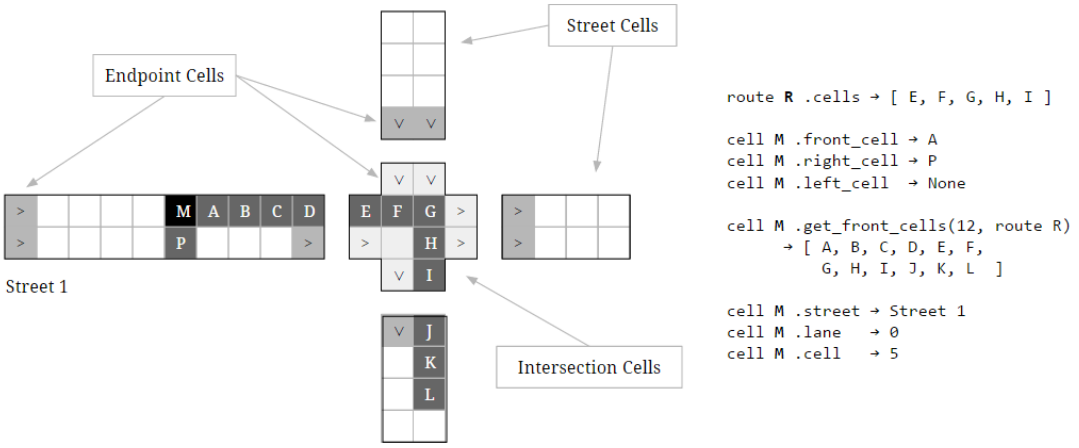
mismo carril y de llegar al final de la calle se llama al mismo método en la celda que esté conectada a la última, que será de tipo *Endpoint*. La Figura 10 muestra el comportamiento de este método para una celda cercana a la intersección.

Celdas de intersección Para estas celdas, la clase `IntersectionRule` debe ser implementada en el cálculo de las reglas. Las celdas tienen una referencia a la intersección a la que pertenecen y las rutas de las que hace parte. La implementación del método `get_front_cells(n, route)` determina a partir de una ruta indicada, cuáles son las siguientes celdas en el camino que desea tomar el vehículo.

Celdas de conexión En los límites de las calles e intersecciones, es importante manejar celdas de conexión para que estas puedan conectarse entre sí. Las `EndpointCell` son celdas que adicionalmente a las reglas que se les aplican si son de calle o de intersección, también se encargan de manejar la conexión con otra calle u otra intersección. Estas celdas tienen una clase especial de regla, `endpoint_rule_class`, que se instancia en el primer paso del ciclo de `update`. En la Figura 10 se muestra cómo es que estas celdas se conectan entre sí para realizar las conexiones entre calles e intersecciones.

d Vehículos. Los vehículos en el sistema son representados por la clase `Car` que contiene la celda en la que se encuentra ubicado, un valor entero no negativo de velocidad y una ruta. Adicionalmente, el objeto también guarda una serie de valores utilizados para la *personalización de conductores* que se busca que el simulador sea capaz de tomar en cuenta. Estos valores son calculados utilizando

Figura 10. Estructura de un conjunto de celdas relacionadas en calles e intersecciones. Se muestran las propiedades de una celda M que tiene contenido a un carro que seguirá una ruta R en la intersección.



una distribución normal en el momento de inicializar cada carro. El carro también tiene valores calculados dependiendo de su posición en el mapa.

Velocidad máxima Cada vehículo puede tener una velocidad máxima deseada, de forma que cuando las reglas sean aplicadas, algunos vehículos acelerarán a velocidades mayores que otros.

Tasa de desaceleración La probabilidad de que un vehículo reduzca espontáneamente su velocidad también es un valor distinto asignado a cada carro, tratando de emular el fenómeno que también ocurre en la vida real. Esta reducción de velocidad se implementa debido a la tercera regla del modelo de Nagel (1992) en el que indican que puede suceder por comportamientos humanos o condiciones externas variantes.

Tasa de cambios de carril Todos los vehículos guardan un valor base que indica la frecuencia con la que cambiarían de carril en caso de que las condiciones sean normales. Sin embargo se manejan otros dos valores que son modificados a partir de la ubicación y ruta que siga el carro. La tasa de cambio de carril aumenta conforme el carro se aproxima al final de la calle y no se encuentra en el mismo carril que en el que empieza la ruta que seguirá. También se maneja una tasa de cambio de carril hacia la derecha (y su complemento hacia la izquierda) que representan la probabilidad de que al hacer un cambio de carril, este sea para uno de los dos lados. El valor de esta última tasa se mantiene en 0.5 a menos que el carro se encuentre en un carril distinto al carril de su destino (ruta). En ese caso la tasa toma un valor de 0 o 1 dependiendo del lado al que se encuentre.

Espera máxima para cambio de ruta En el caso de que el vehículo se encuentre al final de la calle detenido por no encontrarse en el carril donde comienza su ruta, puede hacer esperar a los carros que le

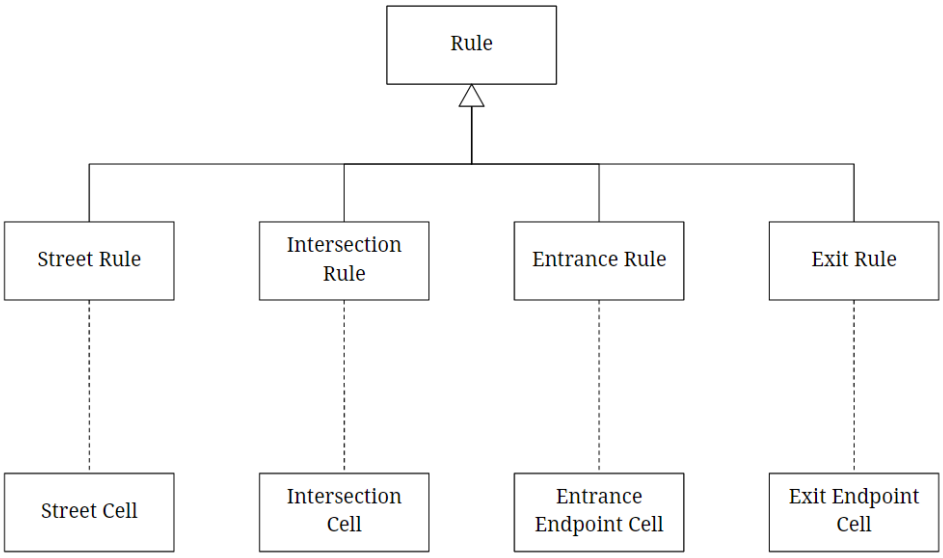
suceden por un tiempo determinado mientras intenta cambiar al carril que desea. Se define un tiempo máximo que el vehículo esperará detenido antes de decidir cambiar de ruta a una que aplique en el carril en el que se encuentra.

Los vehículos, al igual que las celdas, también cuentan con un objeto `ProvisionalCar` en el que se asignan todos los cambios calculados durante una actualización síncrona del mapa.

e Reglas. La clase `Rule` aplica para una celda las reglas determinadas. La aplicación de estas reglas ocurre en un proceso de tres fases. Primero, en la instanciación de la clase se realiza el proceso de **población** en donde se guardan los valores que se utilizan de input en la siguiente fase, en la que se hace el **cálculo** de los nuevos valores que se deben aplicar a las celdas y carros. Finalmente se **aplican** las reglas. Para estos tres procesos se implementaron los métodos dentro de la clase `Rule`: `populate()`, `calculate()` y `apply()`. Para el segundo paso del ciclo de `update()` del automaton, se llama al método `pre_setting()` de la regla, en el momento de hacer los cálculos, debido a que en este método se colocan los cálculos necesarios para aplicar las configuraciones del mapa en el ciclo.

Al igual que las celdas, las reglas también están constituidas en un sistema de herencias de clases como se muestra en la Figura 11. Las reglas `EntranceRule` y `ExitRule` son instanciadas en el primer paso del ciclo de `update()` del automaton y se encargan de generar o consumir carros según las tasas definidas en las celdas limítrofes. La regla `StreetRule` implementa el método `pre_setting()` para realizar los cálculos necesarios del cambio de carril.

Figura 11. Taxonomía de las clases de reglas.



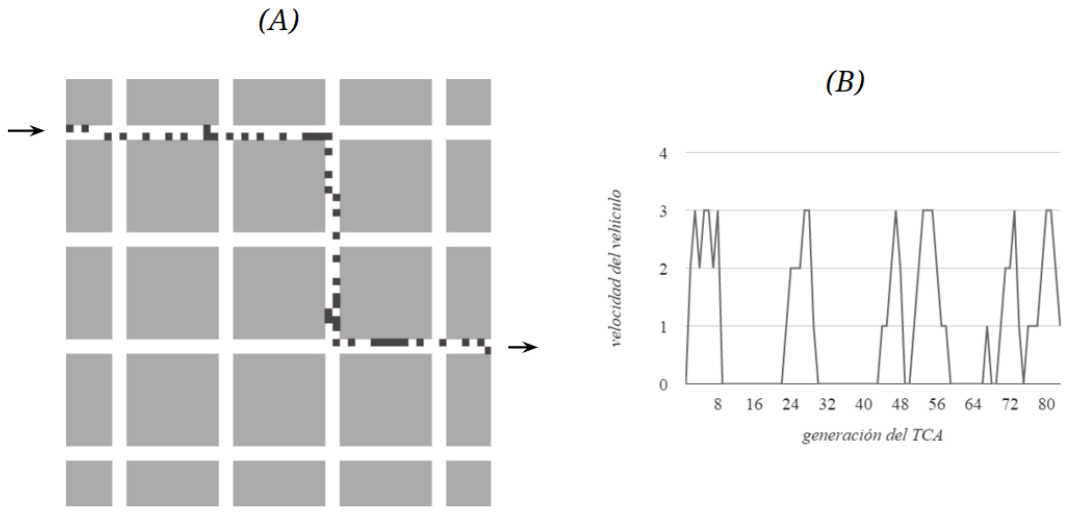
f Semáforos. Los semáforos en el sistema son elementos independientes que se encuentran en cada intersección. Un semáforo tiene un conjunto de luces. Una luz es un objeto que tiene un conjunto de rutas de la intersección asociadas a él, de manera que los vehículos pueden transitar por esas rutas si y sólo si la luz que las gobierna se encuentra en verde. En un semáforo puede haber sólo una luz verde a la vez, aunque dos luces pueden tener a la misma ruta relacionada. El semáforo maneja un diccionario *schedule* en el que tiene configurado el momento en el que cada luz tiene su turno de estar en verde mientras todas las demás están en rojo.

G RESULTADOS

Una vez implementado el simulador, se hicieron diferentes mediciones para comprobar que su funcionamiento fuera correcto. En esta sección se presentan los resultados obtenidos de esas mediciones. También se realizaron mediciones de tiempos de corrida para considerar si el rendimiento del simulador es adecuado.

En la Figura 12 se muestran todas las celdas en las que un vehículo estuvo colocado durante las 83 iteraciones que duró en el mapa y una gráfica que muestra la velocidad del mismo en cada iteración. Se puede observar que el vehículo tiene una velocidad máxima de tres celdas por iteración. El vehículo durante su recorrido hace pausas en semáforos y cambios de carril convenientes para la ruta que está siguiendo.

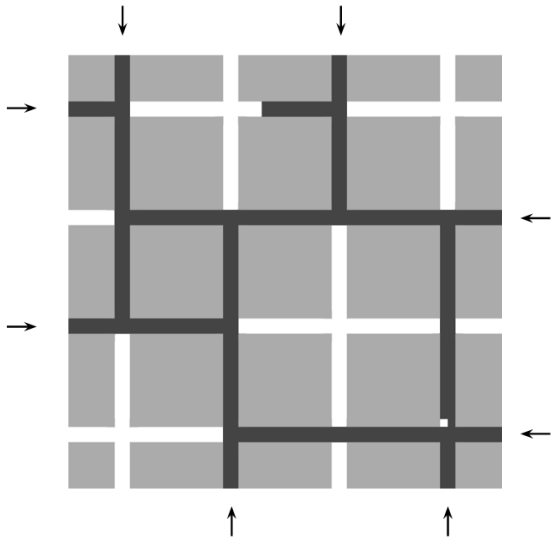
Figura 12. (A) Recorrido de un vehículo a través de una topología de 16 intersecciones desde su entrada en el mapa hasta su salida y (B) velocidad del vehículo en cada generación del TCA. El recorrido toma 83 iteraciones del simulador.



1 Saturación del mapa. Se observa en la Figura 13 que el sistema llegó a un estado en el que ningún vehículo puede avanzar debido a que las intersecciones se encuentran bloqueadas por vehículos que se movilizaban en la otra vía y no pudieron avanzar. Este es un fenómeno que ocurre cuando se satura

el sistema de vehículos, o el ciclo de un semáforo impide el tránsito por una vía durante mucho tiempo. La fila de vehículos detenidos por un semáforo en rojo llega a cubrir hasta la intersección que precede a la bloqueada por un tiempo suficiente para impedir que otros vehículos puedan cruzarla cuando su semáforo les da vía. Cuando un sistema se alcanza esta configuración se dice que está en un estado muerto o *deadlock*.

Figura 13. Captura de un sistema de 16 intersecciones que se encuentra en punto muerto en la iteración 1,200.



La Figura 14 muestra la cantidad de carros que se encuentran en el sistema a través del tiempo para topologías con los mapas ilustrados en la Figura 39. Puede observarse cómo ambos mapas llegan a un punto muerto en el que no pueden entrar más carros al sistema y los que están adentro no pueden moverse, y por lo tanto salir.

Figura 14. Cantidad de vehículos en un sistema de (A) 16 y (B) 64 intersecciones durante la evolución del simulador.

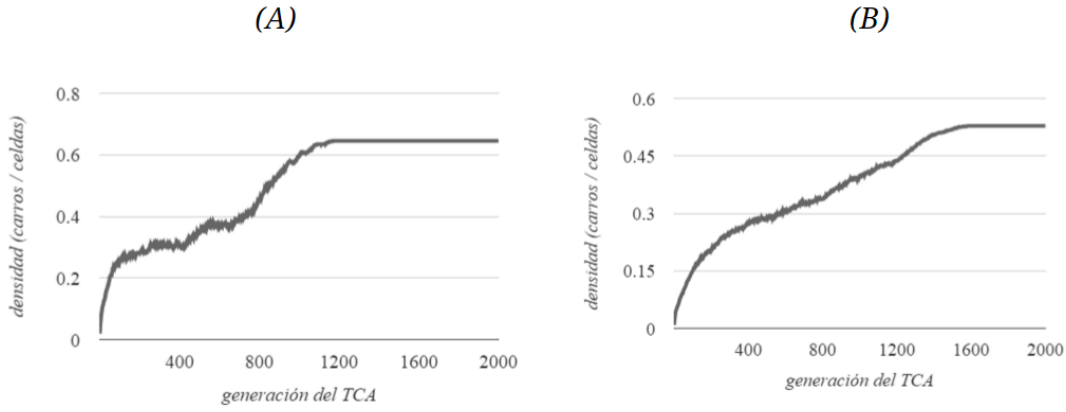
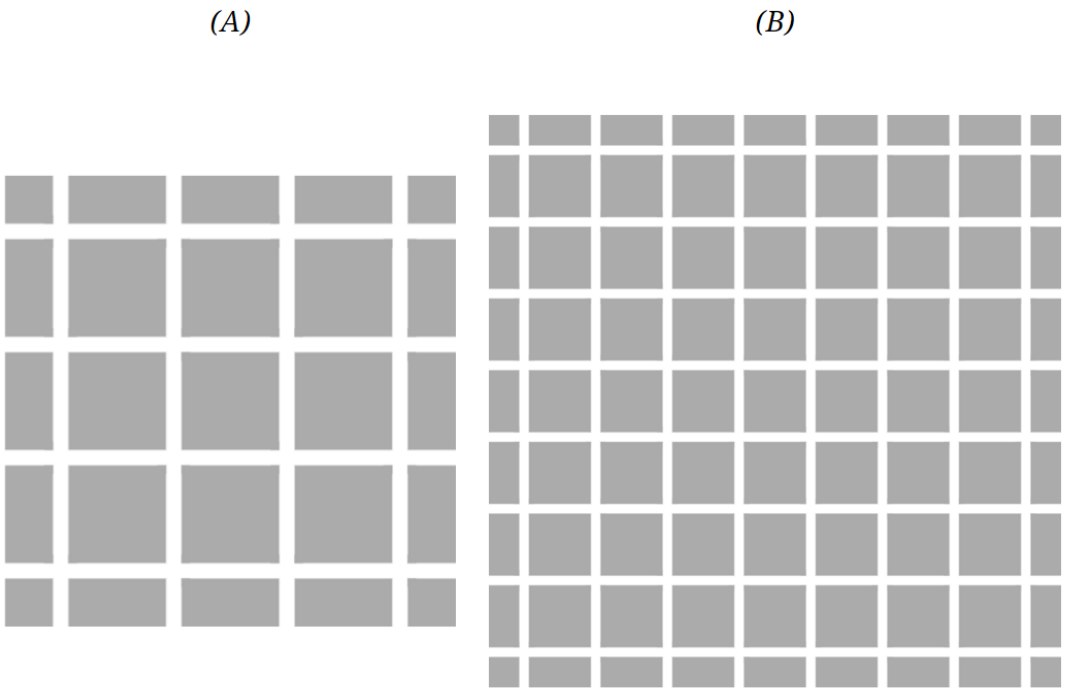


Figura 15. Topologías utilizadas para diferentes corridas en las mediciones realizadas. (A) Topología de 16 intersecciones con 832 celdas y (B) de 64 intersecciones y 3,328 celdas



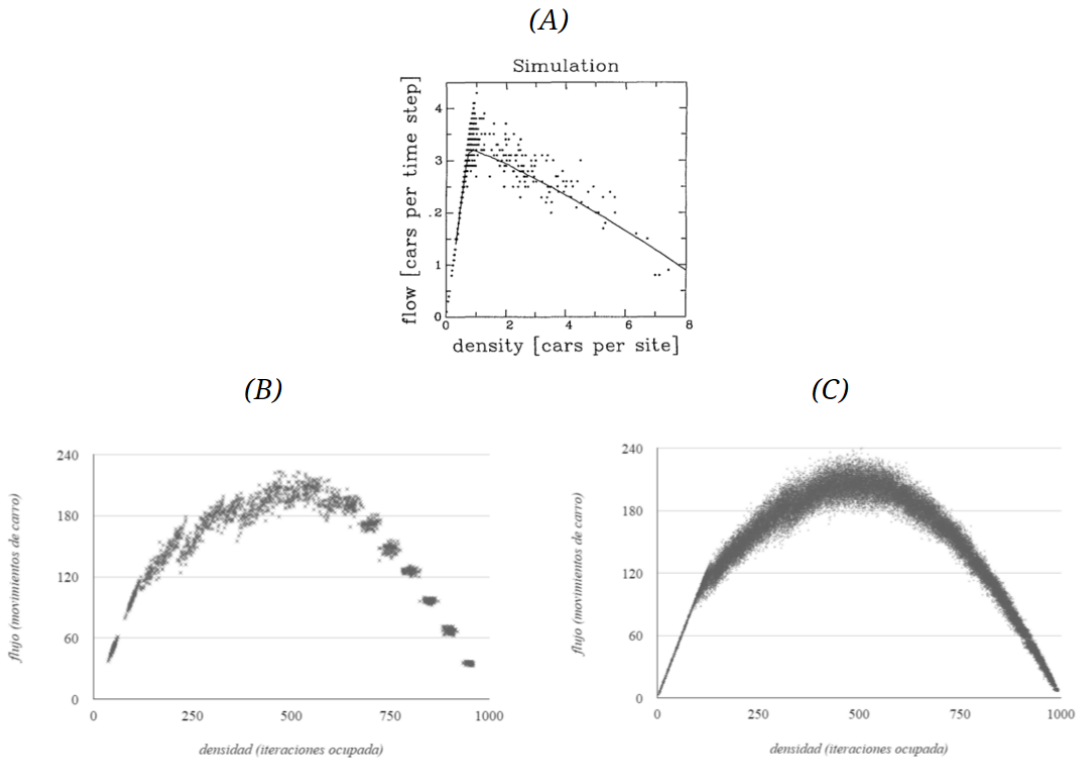
2 Reproducción del experimento de Nagel y Schreckenberg. El simulador implementado es un Traffic Cellular Automaton basado en el modelo definido por Nagel (1992) con una extensión de reglas que permiten la modelación de múltiples carriles e intersecciones. Por ello se intentó replicar el experimento que ellos realizaron en su trabajo. Las mediciones de densidad y flujo se realizaron para una calle de longitud *L* como se muestra en el siguiente bloque de pseudocódigo:

```
steps = (L // 5) - 1
for step in range(steps):
        n = 5 * (step + 1)
        instantiate TCA with street topology of size = L
        generate n Cars in random positions with speed = 0
        set density=0, flux=0 for each cell in topology
        for iter in range(1000):
                for each cell in topology.cells:
                        if cell is not empty:
                                add 1 to cell density
```

```
11                    if cell does not contains same car than in
                          iter -1:
12                            add 1 to cell flux
13            print (density, flux)
```

Existe una tupla de $(densidad, flujo)$ para cada celda de una calle circular de tamaño L en la que se corrió la simulación $\frac{L}{5} - 1$ veces, cada una con una cantidad de vehículos distinta y 1,000 iteraciones corridas. La cantidad de vehículos aumentaba en saltos de 5 en el intervalo $[5, L)$ para cada simulación. En la Figura 16 se muestran los resultados obtenidos por Nagel y Schreckenberg comparados con el experimento descrito para $L = 100$ y $L = 500$ respectivamente en (A) y (B). Se midió la densidad como la cantidad de veces que una celda contiene un carro durante el tiempo de cada simulación (1,000 iteraciones) y el flujo como la cantidad de veces que un carro se moviliza y deja de estar en la celda.

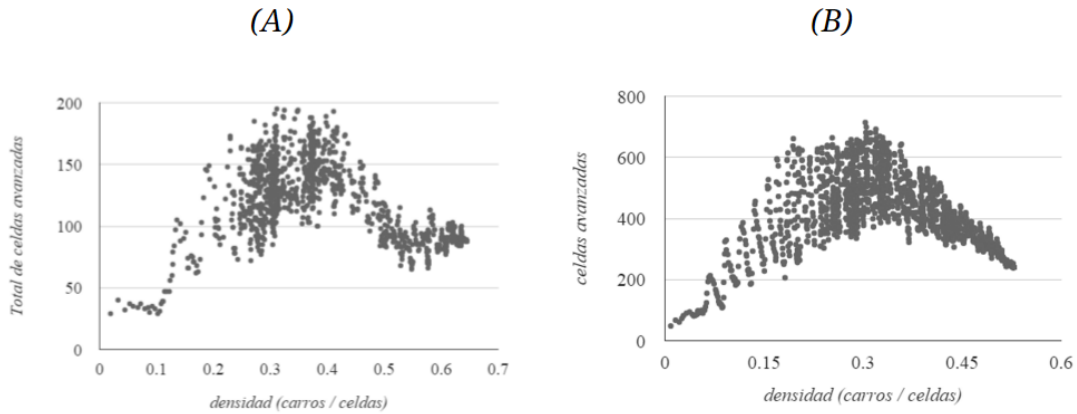
Figura 16. (A) Flujo del tránsito (en carros por unidad de tiempo) vs densidad (en carros por ubicación) de resultados de simulación para una calle de longitud 10^4 (Nagel , 1992) comparados con flujo vs densidad en el simulador implementado: los resultados de (B) son de 1,900 mediciones y los de (C) de 49,500.



3 NaSch en mapas complejos. La medición realizada en la sección anterior que emula el experimento realizado por Nagel y Schreckenberg se trasladó al modelo implementado en este trabajo. Para ello se midieron la densidades y flujos de una manera distinta. La densidad se tomó como la relación entre

la cantidad total de carros en el sistema y el total de celdas para un momento único en la simulación, así $densidad = \frac{total_carros}{total_celdas}$. El flujo se consideró como la suma de las velocidades de todos los carros en el sistema para un momento dado. Los resultados se tomaron de los mismos mapas de 16 y 64 intersecciones (Figura 39) y se muestran en la Figura 17.

Figura 17. Cantidad de celdas avanzadas por los carros de un sistema en comparación con la densidad que tiene el mismo. Resultados obtenidos de simulaciones con 2,000 iteraciones en mapas de (A) 16 y (B) 64 intersecciones



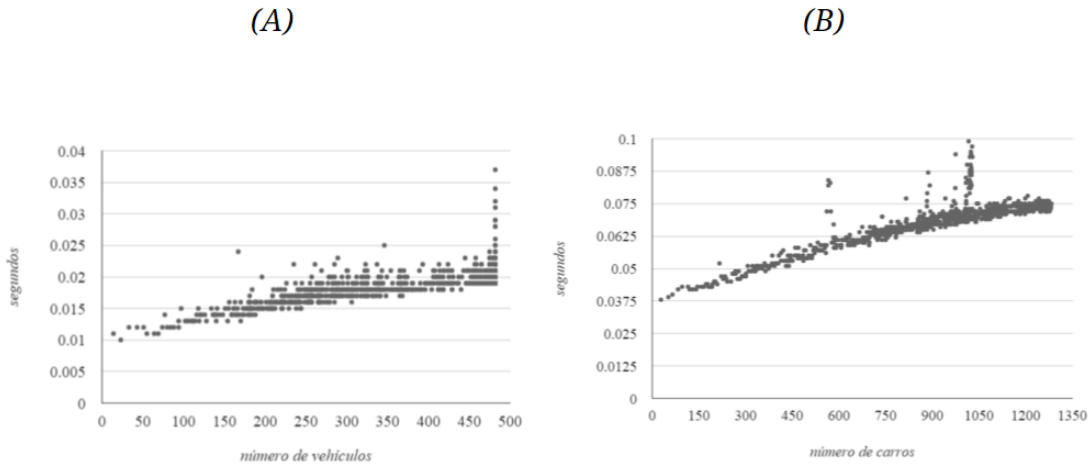
4 Rendimiento del simulador. El tiempo de ejecución de cada ciclo de `update()` del simulador puede verse afectado por dos factores fundamentales: la complejidad del mapa representado por la topología y la cantidad de carros interactuando dentro del mapa. Se realizaron mediciones de tiempo en dos diferentes mapas con tamaños distintos, mostrados en la Figura 39. Para cada mapa se realizaron 1,000 llamadas al método `update()` del Automaton, y para cada llamada se realizó una medición del tiempo tomado en su ejecución. Las mediciones de tiempo se realizaron utilizando la herramnienta cProfile de Python que realiza mediciones determinísticas. En la Figura 18 se muestran los tiempos medidos para las distintas cantidades de vehículos que se encontraban en cada uno de los mapas a lo largo de 1,000 iteraciones.

A partir de los resultados mostrados en la Figura 18 se realizó una regresión lineal para ambos escenarios, dando como resultado las ecuaciones $y = 1{,}862 \times 10^{-5}x + 0{,}012$ y $y = 2{,}709 \times 10^{-5}x + 0{,}043$ respectivamente para los casos (A) y (B).

H DISCUSIÓN

En esta sección se pretende analizar los resultados obtenidos en la sección anterior para poder determinar si el simulador implementado es capaz de simular de manera realista el tráfico en una cuadrícula de

Figura 18. Mediciones de tiempo del método `update()` para 1,000 iteraciones con diferentes cantidades de carros en mapas de (A) 16 y (B) 64 intersecciones.



calles y avenidas. Además también se analizará la viabilidad de utilizarse como entrada para otros algoritmos que intenten optimizar el flujo vehicular, considerando los tiempos de ejecución y la escalabilidad del sistema.

En la Figura 12 se muestra el recorrido registrado por un vehículo y la velocidad que tuvo en cada iteración. Es relevante notar que la velocidad fluctua bastante en intervalos cortos de tiempo, y que existen reducciones muy repentinas de velocidades. Lo primero puede ser debido a que la velocidad máxima de este vehículo en particular es muy baja y las pequeñas variaciones en la velocidad tienen un impacto mayor. Si consideramos que un vehículo promedio podría acelerar en un espacio sin obstáculos de 0 a 70 km/h en 15 segundos y que un vehículo detenido en el tráfico ocupa en promedio 7.5 metros incluyendo la distancia con el vehículo del frente, podemos determinar que una iteración en el sistema se aproxima a 2.6 segundos y que una velocidad máxima promedio aceptable en área urmana puede ser 62.3 km/h $\simeq$ 6 celdas/iteración. Entonces un cambio de velocidad $\Delta v = 1$ deja de ser tan representativo. Los frenos abruptos como se observan en los tiempos 8, 28, 47, 55 y 73 son aceptables bajo esta consideración, debido a que sí es posible frenar un vehículo que se traslada a 30 km/h en un tiempo igual o menor a 2.6 segundos. Sin embargo en una simulación que se permita llegar a mayores velocidades esto puede convertirse en una introducción de error debido a que se permite que el vehículo frene completamente en una sola iteración en lugar de hacerlo gradualmente.

En el recorrido registrado en la misma figura, se observa cómo en el primer semáforo en el que se detiene (del tiempo 9 al 22) realiza un doble cambio de carril estando con velocidad = 0. Este fenómeno no representa la realidad, debido a que para realizar un cambio de carril es necesario avanzar.

Otro aspecto relevante observado en el simulador es que siempre llega a un estado muerto. En la vida real estos puntos muertos sí ocurren y pueden afectar el flujo de vehículos en un sistema particular de calles, pero no son estados de los que sea imposible salir. El conductor de un vehículo que se encuentre obstruyendo una intersección puede —por presión o por desesperación— cambiar despues de cierto tiempo su ruta y permitir que los vehículos bloqueados puedan cruzar la calle.

Al comparar los resultados del experimento en el que se intenta replicar los resultados de Nagel (1992) es notorio que no se reproduce de forma exacta. Sin embargo con la simple observación de las gráficas obtenidas se pueden apreciar algunas tendencias que son comparables. El inicio de los datos, donde la densidad es muy baja, es muy similar: existe una relación lineal entre el flujo y la densidad, debido a que los pocos vehículos transitan libremente. En el experimento original ubican el cambio en el comportamiento en el punto donde $\rho = 0{,}08$, mientras que en nuestro experimento eso sucede entre 0.12 y 0.13. Las densidades que le suceden son las que más diferencia encuentran entre los experimentos. En nuestro simulador el flujo sigue aumentando hasta llegar a un punto máximo aproximadamente en $\rho = 0{,}5$, mientras que el original comienza a dismunuir casi inmediatamente después del cambio. Esto puede ser causado por las formas de obtener las mediciones, que se realizan de maneras ligeramente diferentes.

La aplicación del experimento en mapas complejos presenta resultados similares al modelo de una calle. Es notable que el comportamiento emula a una curva con un valor máximo cercano a la mitad del rango considerado. Estos resultados nos permiten generalizar que en un sistema de calles y avenidas el flujo de vehículos es mínimo cuando la densidad de estos en el mapa es muy baja, comparable a realizar mediciones de tráfico a las tres de la madrugada, o cuando es muy alta, i.e. cuando el tráfico es denso y no permite que los vehículos fluyan. Un flujo máximo se puede obtener cuando la densidad se aproxima a la mitad de densidad de saturación del sistema. Sin embargo también es importante resaltar que en ese punto los datos son más dispersos por lo que una medición de flujo en esta densidad puede resultar también en un flujo muy bajo.

Finalmente, si se observan los resultados de las mediciones de tiempos de ejecución es fácil concluir que sí existe una relación entre el tiempo de ejecución y el número de carros así como con el tamaño del mapa. Sin embargo la primera no es representativa para los dos casos observados ya que el crecimiento del tiempo de ejecución es muy lento. La diferencia, no obstante, entre los dos conjuntos de mediciones sí es representativa ya que es posible observar el desface que existe entre las mediciones de un mapa de 16 celdas con uno de 64. A pesar de que los tiempos aumentan con estos dos factores, siguen siendo tiempos bajos que son totalmente manejables en caso se desee realizar un análisis sobre estos que requiera miles de generaciones. El diseño del simulador también beneficia en este sentido debido a que existe una gran escalabilidad y el procesamiento no depende tanto de la complejidad del mapa como de su tamaño en número de celdas. Es posible implementar diferentes tipos de intersecciones y conexiones entre calles

manteniendo los tiempos de ejecución bajos.

I CONCLUSIONES

El Traffic Cellular Automaton desarrollado en el presente trabajo puede ser utilizado para realizar simulaciones de tránsito en un área urbana y los resultados macroscópicos del mismo darán una perspectiva cercana a la realidad de un arreglo de calles y avenidas. Para que el resultado sea más veraz se puede configurar una velocidad máxima promedio con valor 6 y describir los mapas calculando que las cantidades de celdas por calle sea dada por una longitud base de celda = 7.5 metros.

El simulador implementado alcanza siempre un estado muerto en el que ningún vehículo en el sistema puede movilizarse

La implementación del TCA en el presente trabajo emula el trabajo realizado por Nagel y Schreckenberg en 1992 y sus resultados son comparables con los obtenidos por los alemanes. El modelo implementado es una extensión del definido por ellos y tiene un comportamiento similar. Incluso cuando las mediciones no se realizan sobre una vía circular, comparten el mismo comportamiento en la relación flujo-densidad.

El tiempo de corrida del simulador depende del tamaño del mapa configurado y la cantidad de carros presentes dentro de su topología en un momento determinado. Este tiempo se mantiene bajo para cuadrículas grandes y densidades grandes de carros dentro de ellas. El diseño implementado permite una futura expansión de las reglas y de los mapas sin afectar este rendimiento.

J RECOMENDACIONES

Si se desea realizar simulaciones que contemplen el comportamiento del tránsito vehicular por tiempos largos, se recomienda implementar un método de *deadlock avoidance* que de alguna forma evite que se formen los puntos muertos en los sistemas, pero que no compromenta el buen funcionamiento y la eficiencia del simulador.

Se puede implementar un lenguaje propio o una interfaz gráfica para la creación de mapas si se desea construir mapas más sofisticados. De la misma forma, se invita a extender las reglas para permitir intersecciones sin semáforos y uniones y bifurcaciones de calles.

VI. DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO GENÉTICO PARA LA OPTIMIZACIÓN DE CICLOS DE SEMÁFORO EN INTERSECCIONES DE UNA CUADRÍCULA

A INTRODUCCIÓN

En el módulo de *Diseño e implementación de un algoritmo genético para la optimización de ciclos de semáforo en intersecciones de una cuadrícula* busca diseñar e implementar un algoritmo genético para la optimización de ciclos de semáforo. Esto implica la definición de los componentes que lo integran, a saber, codificación de soluciones candidatas, estrategia de selección y operadores de cruce y mutación. Las decisiones de diseño de cada componente están fundamentadas en experiencias encontradas en la literatura, pero fueron refinadas tras obtener resultados empíricos.

El algoritmo genético propuesto introduce una estrategia de selección híbrida, en la que un porcentaje de la población se elige determinísticamente con un método elitista y el resto de la población se selecciona con un método estocástico. Se empleó un operador de cruce de intercambio parametrizado uniforme y un operador de mutación de intercambio de índices.

Tras analizar el rendimiento del algoritmo genético al variar diversos parámetros, se concluyó que el éxito del algoritmo para encontrar soluciones óptimas depende de la elección adecuada de la cantidad de individuos en la población y la cantidad de generaciones. Para elegir estos parámetros se debe tomar en cuenta el tamaño del espacio de soluciones candidatas. Esto puede calcularse según la cantidad de intersecciones en el mapa y el largo del período a optimizar.

B OBJETIVOS DEL MÓDULO

1 Objetivo general del módulo.

Diseñar e implementar un algoritmo genético para la optimización de ciclos de semáforo en intersecciones de una cuadrícula.

2 Objetivos específicos del módulo.

- Diseñar la codificación de soluciones candidatas de acuerdo al modelo de TCA.
- Diseñar la estrategia de selección del algoritmo genético.
- Diseñar el método de cruce del algoritmo genético.
- Diseñar el operador de mutación para el algoritmo genético.
- Diseñar las funciones de evaluación y aptitud para el algoritmo genético.
- Implementar el algoritmo diseñado para que funcione con el simulador de TCA.
- Evaluar el rendimiento del algoritmo genético.

C JUSTIFICACIÓN DEL MÓDULO

Estudios en el comportamiento del tránsito sugieren que estrategias de tiempos fijos para la regulación de circulación de vehículos no resultan eficientes al aumentar la afluencia vehicular. Por tanto, abordar la programación de circulación de vehículos a través de semáforos y de los agentes de tránsito utilizando un algoritmo genético resulta atractivo. Turky et al y Sánchez-Medina et al en sus investigaciones han obtenido resultados favorables al implementar algoritmos genéticos para regular el tiempo de semáforos en intersecciones. Como parte del megaproyecto se propone implementar un algoritmo genético para optimizar el tráfico en el modelo de TCA desarrollado, que se alimente de datos provistos por el simulador.

D ANTECEDENTES

En este módulo se busca diseñar e implementar un algoritmo genético para la optimización de ciclos de semáforo utilizando un simulador basado en *Traffic Cellular Automata*. A continuación se presenta un recorrido exhaustivo sobre estudios específicos en algoritmos genéticos para la optimización de ciclos de semáforo.

1 Algoritmos genéticos. Entre 1950 y 1960 diversos científicos de la computación estudiaron, independientemente, sistemas evolutivos aspirando a que pudieran servir como herramientas para resolver problemas de optimización en ingeniería. Sin embargo, en 1970, John Holland formuló los *algoritmos genéticos* para estudiar propiamente el fenómeno natural de adaptación y los mecanismos por los que estos podrían incorporarse a las ciencias de la computación (Mitchel, 1998).

En la propuesta original de Holland, se planteaba un algoritmo genético que podía pasar de una población de *cromosomas* a una nueva población aplicando una especie de *selección natural*, así como un conjunto de operaciones inspiradas en la genética, como la mutación y el cruce. Cada cromosoma está

compuesto de *genes*, donde cada gen es una instancia de un *alelo*. La operación de selección elige los cromosomas de la población a los que se les permitirá reproducirse, entre los cuales los más aptos generarán más descendientes. El operador de cruce intercambia subpartes de dos cromosomas y la mutación cambia alelos aleatoriamente en un cromosoma.

El trabajo de Holland fue una sólida base teórica sobre la cual numerosos investigadores han estudiado hasta que hoy en día, aunque las nociones se conservan, los conceptos han avanzado y se han diversificado los estudios dentro de este campo. Ya que actualmente no existe una definición rigurosa sobre lo que constituye un algoritmo genético, se presentarán los componentes y técnicas

2 Optimización de ciclos de semáforo utilizando algoritmos genéticos. Los problemas de optimización de ciclos de semáforo son NP-hard, por lo que algoritmos que utilizan métodos exactos de búsqueda pueden ser poco viables por dos razones, a saber, la alta demanda computacional que requieren y la estocacidad propia de las dinámicas del tráfico. Por eso se ha investigado sobre Algoritmos Genéticos para obtener soluciones a la optimización de ciclos de semáforo. En esta sección se presenta un recorrido histórico de algunos enfoques y avances en el uso de algoritmos genéticos y se hace una revisión de su uso en conjunto con un simulador basado en TCA.

La primera publicación sobre el uso de algoritmos genéticos para la determinación de tiempos de señales fue por Foy *et al.* (1992). En esta se trató una cuadrícula de cuatro intersecciones para una contexto con tráfico fijo. Para evaluar los resultados de los tiempos de señales se utilizó un simulador simple en el que se buscaba minimizar el tiempo en espera de los vehículos. Sus resultados se acercaban a los óptimos. En esta misma línea Sun *et al.* (2003) estudiaron el uso de algoritmos genéticos multi-objetivo para minimizar el tiempo de espera de los vehículos y la cantidad de paradas de los vehículos. En vez de utilizar una simulación, utilizaron un conjunto de funciones estadísticas para calcular el rendimiento de su modelo. Braun *et al.* (2005) integraron su algoritmo genético en un sistema de ingeniería de tráfico para llevar a cabo un estudio de campo en Regensbug, Alemania, mostrando mejoras en el tráfico del área.

Por otro lado, Zhiyong *et al.* (2006) propusieron un algoritmo genético modelo jerárquico para un área urbana coordinada por un sistema de control de tráfico. Se busca optimizar los ciclos de semáforo a nivel de los controladores de cada intersección y a nivel global por un controlador central. El sistema propone una arquitectura de dos niveles con parámetros que son optimizados en intervalos entre 5 y 30 minutos. Utilizaron simulaciones para mostrar la viabilidad de su sistema, obteniendo resultados positivos.

Los trabajos hasta ahora mencionados se han enfocado en encontrar soluciones para controladores fijos. Se obtiene una solución óptima utilizando un algoritmo genético utilizando datos históricos y simulaciones para luego fijar estos tiempos en el sistema de tránsito. Sin embargo, autores han estudiado sistemas de tránsito accionados por sensores en tiempo real. Yun *et al.* (2005) consideraron sistemas de señales ac-

cionados utilizando heurísticas y los compararon al resultado de herramientas de optimización existentes, obteniendo resultados prometedores. Posteriormente, Stevanovic *et al.* (2007) propusieron un algoritmo genético que utiliza señales para optimizar ciclos de semáforo, haciendo uso de VISSIM como ambiente de simulación y evaluación.

Sánchez-Medina *et al.* (2004) propusieron una arquitectura para la optimización de ciclos de semáforo basada en traffic cellular automata, algoritmos genéticos y un sistema de computación paralela. Los autores no se limitaron a presentar la propuesta una vez, sino que han publicado otros tres estudios basados en la misma arquitectura. En la primera publicación posterior, Sánchez-Medina *et al.* (2005) demuestran que entrenar el algoritmo genético con un simulador determinístico es tan válido como con uno estocástico, pero con notables ventajas en tiempos de ejecución y cantidad de pruebas. Posteriormente, Sánchez-Medina *et al.* (2010) publicaron un artículo sobre la implementación real en La Almozara, España, bajo condiciones de tránsito congestionado con resultados positivos bajo condiciones extremas. Recientemente, Sánchez-Medina *et al.* (2015) publicaron un estudio de a escalabilidad de su arquitectura, haciendo pruebas de rendimiento en escenarios diversos.

Por su parte, Turk *et al.* (2009) presentaron una propuesta de sistema para control de tráfico utilizando algoritmos genéticos y un simulador basado en TCA. Sin embargo, su propuesta no es escalable y se limita a cubrir el caso de una intersección con cinco luces de semáforo, puesto que incluyen un paso peatonal en una calle dentro de su modelo. Utilizaron un simulador de TCA, mas como método de evaluación hicieron uso de heurísticas construidas con datos del tráfico en la intersección.

E DELIMITACIÓN

El algoritmo genético utilizará los datos provistos por el simulador basado en TCA para la evaluación de aptitud de las soluciones candidatas. Se investigará sobre las estrategias y métodos que puedan hacer el algoritmo más efectivo. Con base en estos resultados se diseñarán los componentes del algoritmo apegados a la lógica del modelo de tránsito, buscando maximizar la velocidad promedio de los vehículos y minimizar el tiempo en el que están detenidos.

F MARCO TEÓRICO

1 Algoritmos genéticos.

A partir de las nociones básicas de Holland, expuestas en la sección de Antecedentes, se ha explorado en la literatura, diversas técnicas y estrategias para conformar un algoritmo genético. A continuación se exponen algunas opciones de diseño e implementación para cada uno de los componentes. En esta sección se seguirá el recorrido de Mitchel (1998), quien ha publicado el libro de referencia más común y amplio en algoritmos genéticos hasta la fecha. También se introducen

aportes más recientes a la literatura.

a Codificación de soluciones candidatas. La parte más fundamental de un algoritmo genético es la forma en que se codifican las soluciones candidatas en cromosomas. La mayoría de algoritmos utilizan una codificación binaria que asume un largo fijo y un orden fijo, pero se han presentado otro tipo de estructuras. Adicionalmente, algunos autores han experimentado con utilizar codificaciones adaptativas.

La codificación binaria es la más utilizada, en gran parte por razones históricas. En el trabajo original de Holland, se utilizaban cadenas de bits de largo y orden fijos. Por esta razón, el desarrollo teórico y práctico de los operadores de cruce y mutación se ha dado alrededor de codificaciones binarias. Sin embargo, en múltiples casos forzar una codificación binaria podría resultar en ordenes arbitrarios y poco naturales.

Para muchas aplicaciones es más oportuno utilizar un alfabeto de varios caracteres o números reales. Bajo el esquema de Holland una codificación con alfabetos o números reales tendría un menor rendimiento a uno binario, pero esto ha sido cuestionado por autores como Antonisse (1989). Por otro lado, estudios empíricos han exhibido mejor rendimiento con codificaciones en alfabetos. Sin embargo, el rendimiento del algoritmo depende del problema a resolver y otras características del algoritmo y no tanto de la codificación.

También es común codificar la solución como un árbol, como lo hizo John Koza (1994) en su algoritmo para representar programas de computadora. El problema con esta técnica es que las soluciones pueden evolucionar a tamaños incontrolables. Carrano *et al.* (2007) compararon el rendimiento de codificación por árbol, observando que su rendimiento incrementaba en problemas orientados a nodos.

A pesar de las diversas técnicas de codificación conocidas, es difícil encontrar la forma ideal en un principio por lo que se ha explorado la posibilidad de usar cromosomas adaptativos. Como es propio de los algoritmos genéticos, no se tiene suficiente información del problema o de la solución cuando se diseña, por lo que es posible que una coidificación arbitraria impida un rendimiento óptimo. Por esta razón se han construido técnicas para trabajar con cromosomas de largo variable, adaptando el operador de cruce y agregando operadores como el de *inversión*, que cambia los índices de las soluciones candidatas. (**?**)

b Métodos de selección. El método de selección determina qué individuos de la población podrán pasar a la siguiente generación y cuántos descendientes tendrán. Se busca elegir los mejores individuos con el objetivo de mejorar las soluciones. Elegir un método de selección puede determinar el éxito: una selección muy estricta podría llevar a una convergencia prematura, mientras que una muy laxa haría el proceso de evolución muy lento. Existen numerosos métodos de selección en la literatura de algoritmos genéticos, mas no hay una forma rigurosa de elegir el método selección más apropiado para el problema. En esta sección se presenta una lista con algunos de los métodos más comunes.

Selección proporcional a la aptitud Holland en su propuesta original utilizaba una selección proporcional a la aptitud de la soluciones. Una de las implementaciones más comunes de este método es el de *ruleta*. A cada individuo se le asigna una porción de la rueda en proporción a su aptitud. Luego se hace girar la ruleta la cantidad de veces que sea necesaria para tener la siguiente generación. Estadísticamente se espera cierta cantidad de descendientes por individuo, pero en la práctica puede que no se obtengan los valores esperados y que incluso se elija una mayoría de individuos poco aptos.

Escala Sigma Para resolver los problemas de convergencia prematura de la selección proporcional a la aptitud, se ha propuesto el método Escala Sigma en el que se define una función de valor esperado que toma en cuenta el valor de aptitud del individuo y el promedio de aptitud de la población. De esta forma, se mantiene una presión de selección constante a lo largo de la evolución en vez de depender de la varianza de cada generación.

Elitista Con una estrategia elititsta se fuerza a que el algoritmo pase los mejores individuos a la siguiente generación, para evitar que se pierdan en caso de no ser seleccionados por métodos estocásticos. Los investigadores han notado que el rendimiento de los algoritmos genéticos aumenta con esta técnica.

Boltzmann Selection La Escala Sigma mantiene una presión constante de selección a lo largo de la ejecución, pero a menudo es deseable que la presión varíe con el tiempo. Esto es porque al inicio de la evolución, los individuos tienen aptitudes variadas y por eso podría ser mejor tener una selección más abierta. Conforme avanza el proceso, la varianza de la aptitud de los individuos disminuye y es por tanto conveniente aumentar el rigor de la selección.

Selección por rango Es otro método para reducir la convergencia prematura. En este se asigna un rango a los individuos según su aptitud y se seleccionan en base a este rango. De esta forma se ocultan los valores absolutos de aptitud, reduciendo el riesgo de una convergencia prematura. Sin embargo, dependiendo el problema que se quiere resolver este puede o no ser efectivo, pues en algunos casos conocer el valor real de la aptitud es crucial.

Selección por torneo Los métodos descritos anteriormente requieren dos pasadas por la población: una para determinar el valor promedio de aptitud y otro para determinar el valor esperado de cada individuo. El de rango requiere ordenar toda la población, lo cual puede ser costoso computacionalmente. El método de torneo es más eficiente en cuanto a recursos, pues elige un par de individuos aleatoriamente y luego se elige el más apto si un número aleatorio pasa un umbral definido. Ambos elementos son devueltos a la población de modo que pueden elegirse de nuevo.

Selección de estado fijo En las técnicas anteriores se describen métodos en las que se intercambian las generaciones. Sin embargo, como en la estrategia elitista, es posible retener individuos de generaciones pasadas. En una selección de estado fijo solo son reemplazados algunos individuos de los menos aptos, mientras el resto se conservan.

c Operador de cruce. El operador de cruce (*crossover*) intercambia segmentos de dos individuos para generar uno nuevo. Una de las formas más simples de cruce es la de punto único: se elige un punto aleatorio de intercambio entre los padres y sus posiciones se intercambian. Esta estrategia asume esquemas cortos con bajo ordenamiento, lo que puede llevar al acarreo de bits subóptimos entre soluciones.

También es posible utilizar un cruce de dos puntos, en el que dos posiciones son elegidas aleatoriamente para ser intercambiadas. Esto ayuda en esquemas de mayor escala. Es incluso posible realizar un intercambio parametrizado uniforme, en el que cada bit se intercambia con una probabilidad p.

Finalmente, no existe un criterio determinístico para elegir un tipo de operador de cruce, pues depende en gran parte de la función de aptitud, de la codificación de la solución y del problema mismo.

d Operador de mutación. El operador de mutación (*mutation*) introduce cambios de bits aleatorios un un individuo según una probabilidad p. La utilidad del operador de mutación ha sido ampliamente discutido en la literatura. Algunos autores argumentan que el operador de cruce puede opacar la utilidad de la mutación en su capacidad de generar nuevos esquemas. Sin embargo, esta utilidad de la mutación varía según el tiempo de ejecución del algoritmo.

G METODOLOGÍA

Para el presente módulo se diseñará e implementará un algoritmo genético para optimizar los tiempos de semáforo de una zona urbana. El algoritmo utilizará un simulador basado en Traffic Cellular Automata para evaluar la aptitud de las soluciones. A continuación se describen los componentes del algoritmo genético a utilizar y se presentan detalles de la arquitectura de la implementación en Python utilizando DEAP.

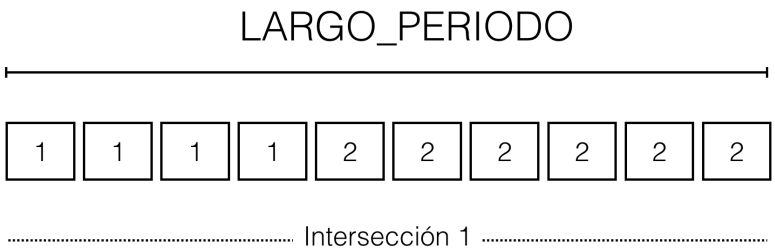
1 Diseño de algoritmo genético. Para solucionar un problema utilizando un algoritmo genético es necesario considerar cuidadosamente los elementos que se utilizarán, de modo que puedan encontrarse respuestas adecuadas al problema. Sin embargo, tras hacer una revisión de la literatura específica de investigación en algoritmos genéticos, Mitchel (1998) explica que no existen métodos determinísticos para elegir acertadamente los componentes del algoritmo. Esto es debido a la alta dependencia del problema que se intenta resolver y la naturaleza de las soluciones candidatas, además del ambiente en el que se desempeñan. Por tanto, para este trabajo se ha hecho un estudio de lo que otros autores han implementado como compontentes de un algoritmo genético para optimizar ciclos de semáforo. Adicionalmente, las decisiones de diseño del algoritmo también fueron refinadas tras un tratamiento empírico al medir su rendimiento en conjunción con el simulador basado en traffic cellular automata.

a Codificación de soluciones candidatas. La codificación de la soluciones candidatas en un *cromosoma* es uno de los componentes más fundamentales de un algoritmo genético y puede

determinar su éxito. Para ello se consideró el acercamiento de Turk *et al.* (2009) en el que se utilizaba una codificación binaria para representar tiempos de luces rojas y verdes, pero se concluyó que la solución provista no era escalable a un número arbitrario de intersecciones de arbitraria forma. La codificación de soluciones candidata propuesto en este proyecto está derivada del trabajo continuo de Sánchez-Medina *et al.* (2004), que ha realizado estudios posteriores sobre su arquitectura para la optmización de luces de semáforo, entre los cuales está una prueba de escalabilidad en 2015 (Sánchez-Medina *et al.* , 2015).

El cromosoma representa un ciclo de semáforo de periodo T para las i intersecciones con n_i luces de la misma. Los genes del cromosoma representan unidades de tiempo secuenciales, donde sus posibles alelos son una de las luces de la intersección. En la Figura 19 se ilustra un cromosoma para una intersección.

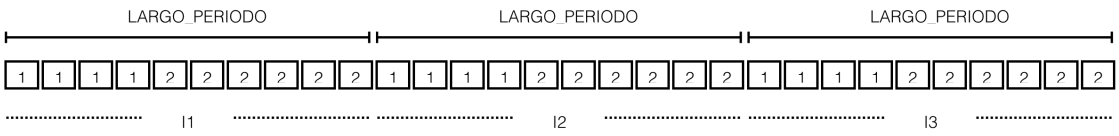
Figura 19. Cromosoma del proyecto para una intersección.



La codificación de soluciones candidatas tiene un largo variable, que depende de la cantiad de semáforos que quieran coordinarse y el periodo que se quiere optimizar. Por tanto, el tamaño del cromosoma también varía, afectando el tiempo de ejecución que requerirá el algoritmo para encontrar una solución.

El diseño de esta codificación de solución candidata permite optimizar ciclos de semáforo para contextos con múltiples intersecciones con cofiguraciones arbitrarias, pues no depende de la forma en que se relacionan las calles sino únicamente de las luces de cada intersección.

Figura 20. Cromosoma del proyecto para tres intersecciones.



b Método de selección. Se optó por una estrategia híbrida de selección entre Elitista y Ruleta, pues de este modo las mejores soluciones candidatas se pasan a la siguiente generación para evitar perderlas por métodos estocásticos y el resto de soluciones se recombinan estocásticamente para evitar convergencias prematuras. Sánchez-Medina *et al.* (2004) obtuvieron mejore resultados empíricos utilizando una estrategia elitista de selección, pero al intentar esa estrategia con el simulador basado en

TCA de este proyecto se observó una convergencia muy prematura.

c Operador de cruce. Primero se consideró el uso de un cruce simple de un punto, en el que los cromosomas intercambiarían sus bits a partir de cierto punto aleatorio. Sánchez-Medina *et al.* (2004) y Turk *et al.* (2009) utilizaron este operador de un solo punto con buenos resultados. Sin embargo, desde un punto de vista conceptual esta aproximación podría no ser la mejor, pues introduce el acarreo de bits subóptimos en las soluciones candidatas. Esto se cumple especialmente por el diseño del cromosoma, pues con facilidad puede hacerse extenso.

Por estas razones, se optó por un operador de intercambio parametrizado uniforme. Este operador podría mostrar un mejor rendimiento, sobre todo, en intersecciones con más de dos luces. Con este operador se establece un umbral que determina si los bits de dos cromosomas se intercambian o no.

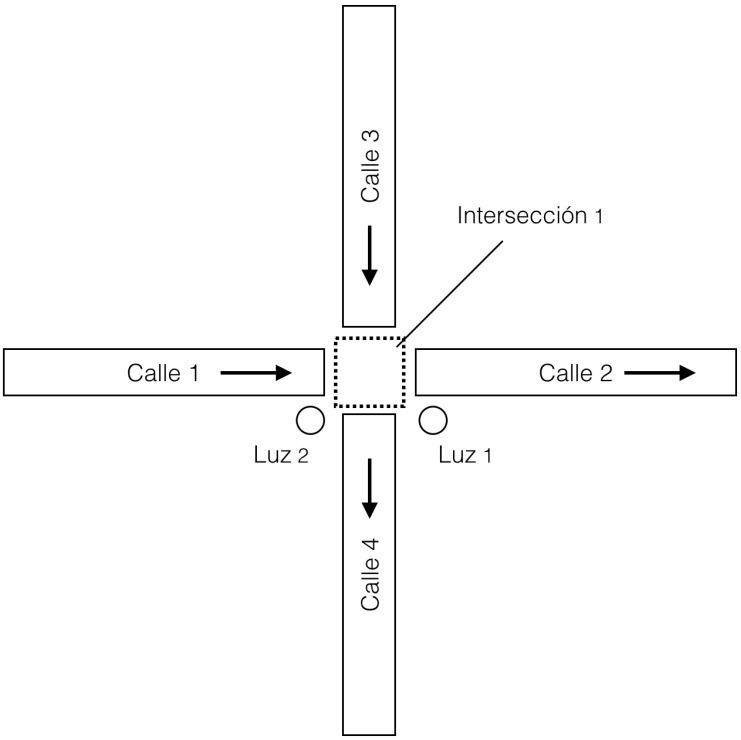
d Operador de mutación. Debido a que el operador de cruce realiza intercambios a nivel de bits, el efecto de la aleatorización se ve opacado. Sin embargo, se optó por introducir mutaciones como intercambios de índices para que las variaciones permitan explorar más soluciones candidatas y eviten una convergencia prematura. Esta técnica resulta conveniente para el problema porque lo que se representa dentro de la codificación de las soluciones candidatas son índices que hacen referencia a luces de semáforos en intersecciones, por lo que no tendría sentido solo cambiar aleatoriamente estos valores. La mutación por intercambio de índices en vez de hacer cambios aleatorios en los valores del cromosoma, intercambia posiciones dentro del mismo.

e Evaluación de aptitud. Se calculará la aptitud de las soluciones candidatas ejecutando una simulación en el simulador basado en TCA con los tiempos de luces para cada intersección codificda en la solución. Se utilizará una optimización multiobjetivo, en la que se busca maximizar la velocidad promedio de los vehículos durante la simulación y la minimización del tiempo promedio que estuvieron detenidos los vehículos bajo la configuración de semáforos de la solución candidata. Por tanto, la función de evaluación asigna la aptitud de la solución candidato como dos resultados: la velocidad promedio y el tiempo promedio en que los vehículos estuvieron detenidos.

2 Uso de simulador basado en TCA. El cálculo de la aptitud de las soluciones candidatas se obtendrá obteniendo datos de un simulador basado en TCA construído en el *Módulo de Simulación de Tránsito Vehicular a Través de un Modelo de Cellular Automaton*. Para ello, el módulo de *Módulo de Integración de Algoritmos con Traffic Cellular Automaton, Evaluación y Análisis de Resultados de Inteligencia Artificial* ha provisto una serie de métodos para ejecutar simulaciones y obtener macroestadísticas de sus resultados. A continuación se describen los detalles, a alto nivel, de estos módulos relevantes para el funcionamiento del algoritmo genético.

a Semáforos y luces de semáforos. El simulador provee un mapa arbitrario en el que se encuentran definidas calles, intersecciones, semáforos y luces de semáforos, cada uno con un identificador único (ID). En la Figura 21 se muestra la distribución conceptual de los componentes del simulador basado en TCA. Cada intersección está compuesta por dos o más calles que se entrecruzan. Además, cada intersección tiene un semáforo con tantas luces como calles se cruzan en la intersección.

Figura 21. Disitribución conceptual de componentes en el simulador basado en TCA



b Interacción con el simulador. Para interactuar con el simulador se debe utilizar los métodos provistos por el *Módulo de Integración de Algoritmos con Traffic Cellular Automaton, Evaluación y Análisis de Resultados de Inteligencia Artificial*, que provee un método para configurar los tiempos de luces de semáforos en el simulador, uno para ejecutar la simulación y diversos métodos para obtener las estadísticas de la simulación efectuada.

Los métodos de `get_traffic_lights` y `set_traffic_lights` son los de mayor incidencia en el desarrollo del algoritmo genético. Se definió el intercambio de información sobre luces de semáforo a través de diccionarios con la siguiente forma:

```
1 [{
2   'id': 0,
3   'lights': [0, 1],
4   'schedule': {0: 0, 5: 1}
```

```
5    }]
```

La configuración de tiempos, *schedule*, utiliza el formato {`start: id`} donde *start* es la iteración dentro del ciclo en la que la luz con identificador *id* va a cambiar a verde.

3 Uso de librería DEAP. DEAP, *Distributed Evolutionary Algorithms in Python*, es un framework de desarrollo para algoritmos evolutivos. Se diferencia de otras librerías análogas en que busca hacer los algoritmos explícitos y las estructuras de datos transparentes. Se ha utilizado para múltiples estudios que utilizan algoritmos evolutivos en variedad de contextos académicos (Fortin *et al.*, 2012). En esta sección se hace un recorrido por los paradigmas que sugiere DEAP y su uso para el desarrollo del algoritmo genético.

a DEAP `creator`. Este módulo de DEAP facilita la gestión de tipos de datos que se dedicarán al algoritmo genético. Al registrar una clase, DEAP lo decora con parámetros variados de modo que pueda ser utilizado fácilmente con los otros métodos de DEAP.

b DEAP `tools`. DEAP provee un módulo con utilidades que representan patrones comunes dentro de la literatura de algoritmos genéticos para el manejo de cromosomas. Provee implementaciones de diferentes métodos de crossover, mutación, selección, entre otras implementaciones. Asume que las codificaciones de los cromosomas son instancias de `Sequence` para realizar las manipulaciones.

c DEAP `toolbox`. Una utilidad provista por DEAP es la clase `toolbox`. En una instancia de `toolbox` se pueden registrar funciones para abstraer los conceptos de algoritmos genéticos de la implementación concreta del algoritmo. De este modo se facilita la reutilización de algoritmos evolutivos, pues se construyen sobre los conceptos registrados en la `toolbox`.

4 Implementación de algoritmo genético. Se implementó el algoritmo genético en Python 3.4, en concenso con los otros módulos del megaproyecto para reducir la fricción de la integración. En esta sección se describen detalles de la implementación del algoritmo.

a Normalización de IDs. Como se describió anteriormente, cada luz de semáforo tiene un identificador único. Sin embargo, para poder implementar el algoritmo es necesario que la codificación de soluciones candidatas sea de tal manera que las operaciones de cruce y de mutación sigan generando soluciones válidas. Por lo tanto es necesario normalizar las luces de semáforo. Para ello se siguió el siguiente algoritmo:

```
1 intersections = {}
2 real_intersections = {}
3 for intersection in traffic_lights:
4     intersections[intersection.id] = [lights.id % max_lights_num]
```

```
5       real_intersections[intersection.id] = [lights.id]
```

El método extrae también un diccionario con los IDs originales de las luces de modo que puedan mapearse desde los IDs normalizados de las soluciones candidatas a los IDs reales para ejecutar la simulación.

b Construcción de cromosomas. El algoritmo se basa en la evolución de una población de individuos con una codificación se solución candidata. Es necesario inicializar esta población con individuos válidos, pero aleatorios, de modo que puedan luego evolucionar hacia mejores soluciones. Se utilizó el siguiente algoritmo para la construcción de un cromosoma:

```
1 chromosome = []
2 for id, lights in intersections.items():
3     for _ in range(period):
4         chromosome.append(getrand(lights))
```

El algoritmo construye un cromosoma a partir la descripción de intersecciones. El cromosoma está constituido por ID's normalizados aleatorios. Para cada intersección genera tantos ID's normalizados aleatorios como requiera el periodo establecido.

c Decodificación de cromosoma. La descripción de luces del semáforo del API no es compatible directamente con el esquema de codificación de las soluciones candidatas. Es por eso que es necesario construir un objeto a partir de la solución candidata para simularla. El algoritmo de decodificación se muestra a continuación:

```
1  api_inters = []
2  for inter in normal_intersections:
3      api_inter = {"id": inter_id, "lights": real_inters[inter.id]}
4      light_past = -1
5      schedule = {}
6      for t in period:
7          light_t = individual[t]
8          #Register the scheduled light only when there is a change
9          if light_t is not light_past:
10             schedule[t] = real_id
11             light_past = light_t
12     api_inter['schedule'] = schedule
13     api_inters.append(api_inter)
```

Esta construcción debe mapear los IDs normalizados de las luces a los reales y formar una lista de

diccionarios, uno por cada intersección, para conformarse con la descripción del API.

d Evaluación de aptitud de soluciones candidatas. Para evaluar la aptitud de las soluciones candidatas se hace uso del API para ejecutar simulaciones en el simulador basado en TCA. A continuación se presenta la función de evaluación:

```
#Map normalized ids to real ids and calculate times
api_inters = decode_chromosome(individual)
simulator.reset_statistics()
#Change traffic lights and configuration and run simulation
simulator.set_traffic_lights(api_inters)
simulator.fixed_time_start(period * 5)
#Return fitness values
return simulator.get_average_speed(), simulator.get_stopped_time()
```

Primero se decodifica el cromosoma y se resetean las estadísticas del API. Entonces se fijan los tiempos de luces de semáforo de la solución candidata en el API y se ejecuta por un tiempo que equivale a cinco veces el periodo. El valor de aptitud consiste en una tupla de la velocidad promedio y el tiempo promedio en que los vehículos estuvieron detenidos.

La función de aptitud asigna pesos a estos dos objetivos. Es más apta una solución con mayor velocidad promedio y menor tiempo promedio de vehículos detenidos. El primer parámetro tiene el doble de preminencia que el segundo. Esta medida se tomó para orientar la búsqueda de optimización por un camino más determinístico, pues los resultados solían dispersarse cuando ambos parámetros tenían la misma prioridad.

e Algoritmo genético. Como se mencionó en el diseño, se implementó un algoritmo genético con una estrategia híbrida de selección, un operador de cruce de intercambio parametrizado uniforme y un operador de mutación de índices. A continuación se presenta el algoritmo utilizado:

```
while g < max_gen:
    # Select the best individuals for the next generation
    best_num = int(len(population) * 0.10)
    offspring = toolbox.selectBest(population, best_num)
    # Select the rest of individuals for the next generation
    offspring2 = toolbox.selectRest(population,
        len(population)-best_num)

    # Apply crossover on the offspring
    for child1, child2 in zip(offspring2[::2], offspring2[1::2]):
```

```
10          if random.random() < CXPB:
11              toolbox.mate(child1, child2, 0.2)
12              del child1.fitness.values
13              del child2.fitness.values
14
15      # Apply mutation on the offspring
16      for mutant in offspring2:
17          if random.random() < MUTPB:
18              toolbox.mutate(mutant, 0.1)
19              del mutant.fitness.values
20
21      # Evaluate the individuals with an invalid fitness
22      invalid_ind = [ind for ind in offspring2 if not ind.fitness.valid]
23      fitnesses = list(map(toolbox.evaluate, invalid_ind))
24      for ind, fit in zip(invalid_ind, fitnesses):
25          ind.fitness.values = fit
26
27      offspring.extend(offspring2)
28      population[:]= offspring
29      g += 1
```

Los métodos de crossover y de mutación se aplican si un número aleatorio generado es menor al umbral fijados por las variables `CXPB` y `MUTPB`, respectivamente. Para los resultados de este informe se utilzó `CXPB = 0.5` y `MUTPB = 0.2`.

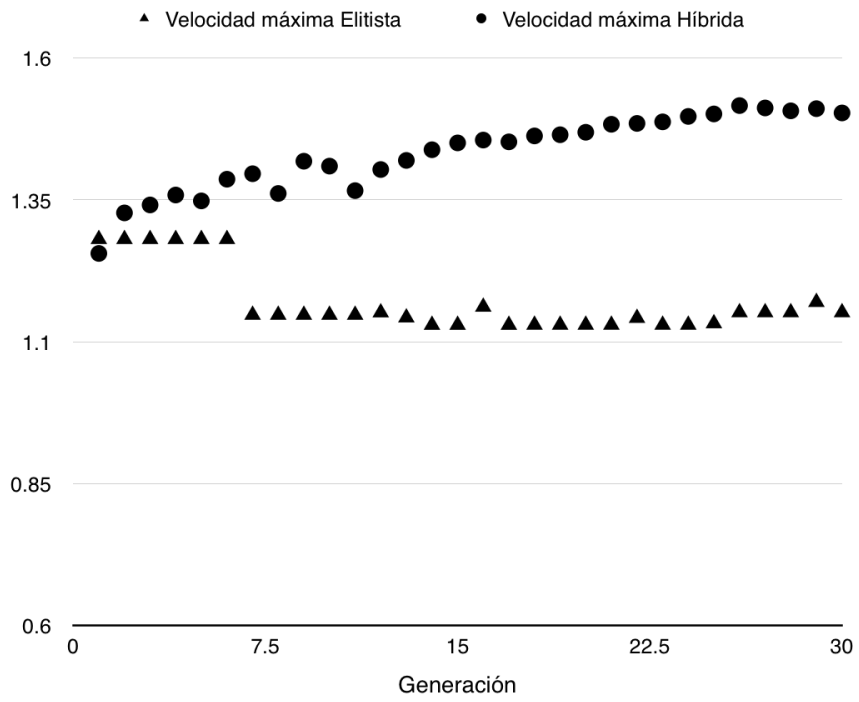
Como puede verse, en este algoritmo se reemplaza la población totalmente por los hijos con cruces y mutaciones.

f Elección de solución. Al terminar de evolucionar la población la cantidad de veces requerida se obtiene una población optimizada. Entonces se selección la mejor de las soluciones candidatas como solución óptima para la configuración de luces de semáforo.

H RESULTADOS

En el desarrollo de un algoritmo genético se suele tener información escaza sobre el problema y las dinámicas que pueden presentar sus soluciones candidatas. Es por eso que un diseño fundamentado únicamente en proposiciones teóricas no es sufiente para ser efectivo. Es necesario refinar y ajustar los compo-

Figura 22. Comparación de la evolución de la velocidad máxima en la población, por generación, utilizando una estrategia de selección elitista contra una estrategia híbrida

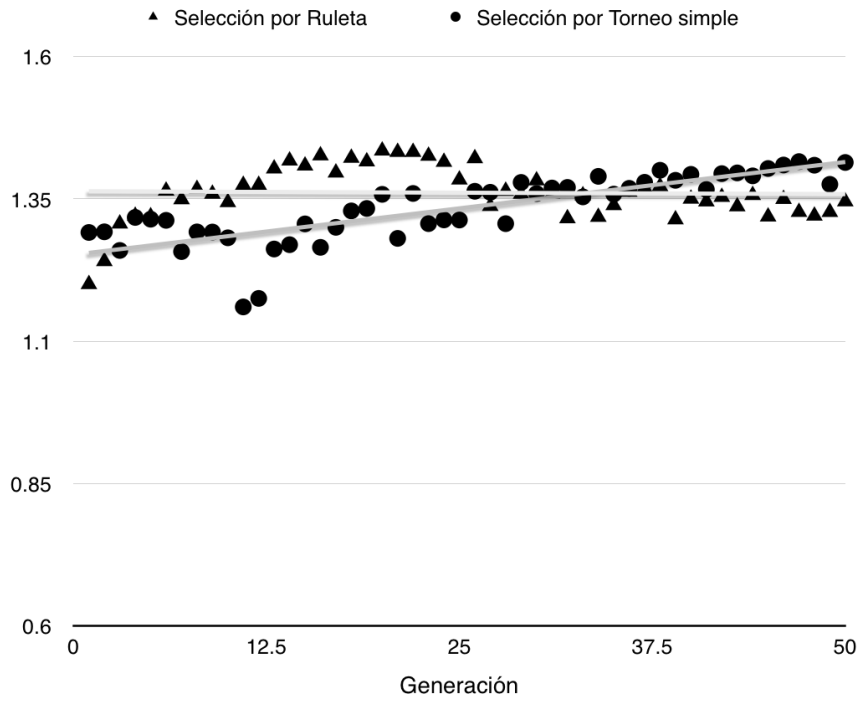


nentes del algoritmo genético tras analizar los resultados variando parámetros. En esta sección se presentan los resultados que llevaron a fijar decisiones y ajustes de diseño, como la estrategia híbrida de selección, y se hace un análisis del desempeño del algoritmo genético en función de la evolución de la población. También se estudia el rendimiento del algoritmo genético en tres mapas distintos.

1 Convergencia prematura. La convergencia prematura es un concepto importante en la evaluación de algoritmos genéticos. Se refire al comportamiento que puede presentar un algoritmo genético al estancarse en un mínimo o máximo local subóptimo. Este es un comportamiento no deseado en la mayoría de casos, pues evita el descubrimiento de mejores soluciones (Mitchel, 1998). Para determinar si un algoritmo genético converge prematuramente, en este análisis se ha comparando el valor de aptitud promedio con el valor de aptitud máximo en la población y se ha considerado la mejora de la aptitud respecto del objetivo de optimización.

2 Estrategia de selección. La estrategia de selección se planteó desde un punto de vista teórico, con la experiencia de trabajos anteriores, en una estrategia de selección elitista. Bajo este esquema, solo las soluciones candidatas que presentaran mejor aptitud podrían reproducirse y pasar a la siguiente generación. Sin embargo, los resultados empíricos sugieren que el rendimiento del algoritmo se ve minado por esta estrategia. En esta sección se contrastan los resultados de la estrategia de selección híbrida contra otras implementaciones posibles.

Figura 23. Comparación de la evolución de la velocidad máxima en la población, por generación, utilizando una estrategia de selección por torneo contra una estrategia de ruleta



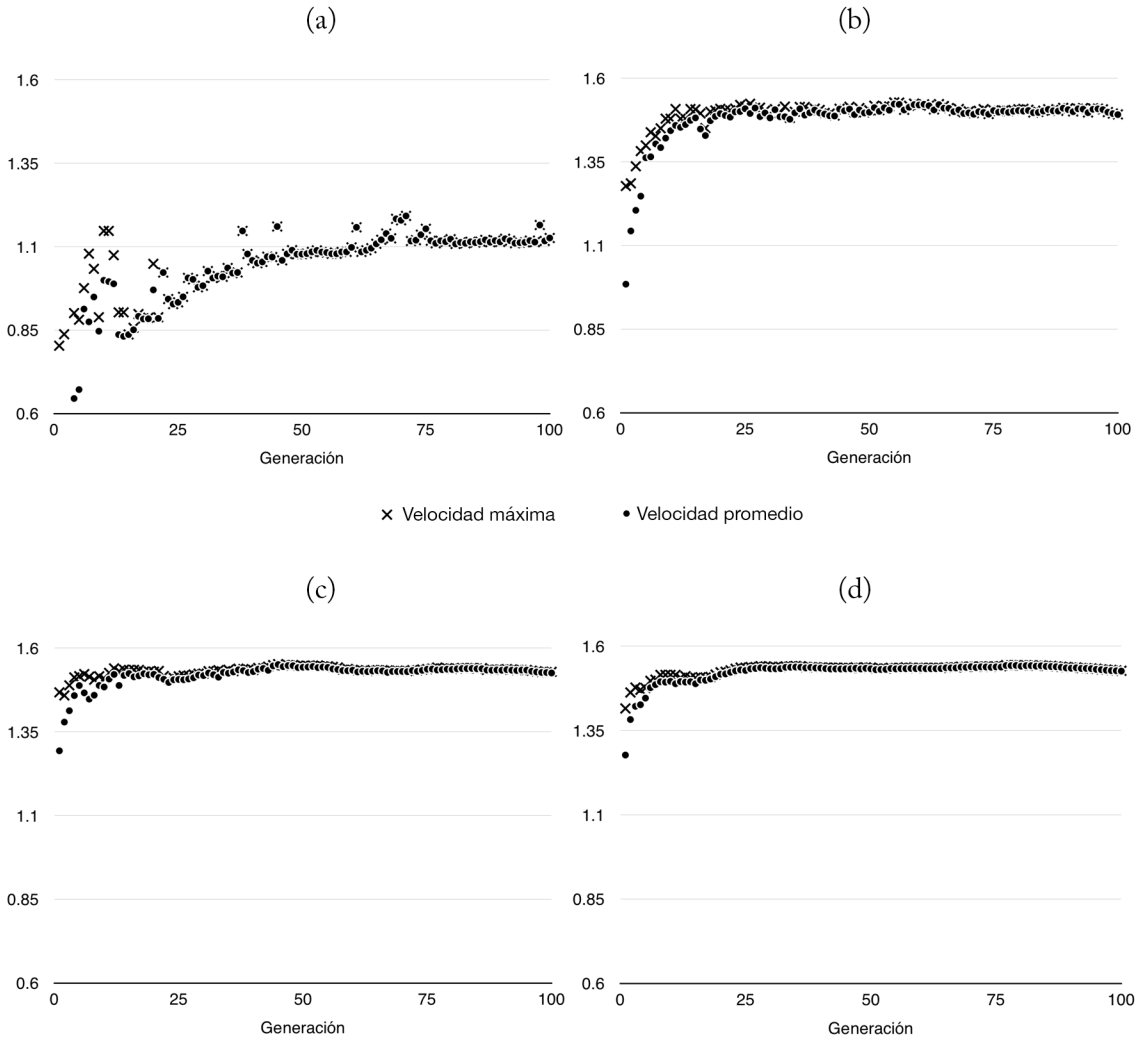
Tras observar que la estrategia elistista llevaba a convergencias prematuras en esta implementación, se realizó una simulación de 30 generaciones, con una población de 100 individuos y un largo de período de 10 iteraciones para ilustrar el problema. En la Figura 22 se muestra la evolución de la aptitud máxima de cada generación tanto para la estrategia elitista como la estrategia híbrida propuesta en la subsección a.

Como ya se mencionó, la estrategia híbrida utilza una selelección elitista para pasar un porcentaje de las soluciones más aptas a la siguiente generación y emplea un método estocástico para el resto de la población. Se probaron dos opciones de implementación para el resto de la población: selección por ruleta y selección por torneo simple. Se observó el comportamiento del algoritmo con ambas implementaciones y se anotó su evolución.

En la Figura 23 se compara la velocidad máxima por generación del algoritmo con estrategia híbrida empleando el método de selección por torneo simple contra el algoritmo híbrido empleando el método de selección de ruleta. Para obtener estos datos se generó una simulación de 50 generaciones con una población de 100 individuos y un período de 10 iteraciones.

Además, en la Figura 23 se han trazado regresiones lineales de mejor ajuste para mostrar claramente la tendencia de ambas estrategias. Para la regresión lineal de la estrategia de selección de torneo es $R^2 = 0{,}661$ contra la estrategia de selección por ruleta con $R^2 = 0{,}001$. Esto indica que la estrategia de selección por ruleta es más apropiada para introducir variaciones a la población.

Figura 24. Gráficas de velocidad máxima y velocidad promedio por generación según tamaño de la población. (a) Población = 10 (b) Población = 50 (c) Población = 100 (d) Población = 200



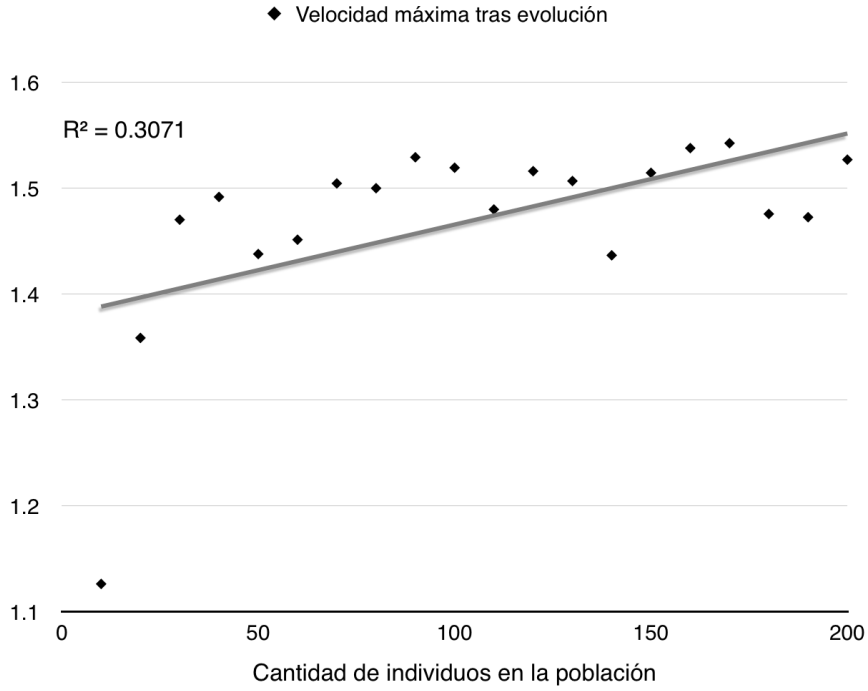
3 Rendimiento en proporción a la población. Se evaluó el rendimiento del algoritmo según el tamaño de población. Concretamente, se registró la evolución de la aptitud en poblaciones desde 10 individuos hasta 200 con un período de 10 iteraciones, en intervalos de 10. El objetivo de este procedimiento es observar el comportamiento del algoritmo genético según la cantidad de individuos en la población. En la Figura 24 se muestra un diagrama de dispersión de velocidad promedio de la población y velocidad máxima de la población (en celdas por iteración) por generación para poblaciones de 10, 50, 100 y 200 individuos.

Un dato a notar en estas gráficas de la Figura 24 es la convergencia prematura en la evolución de la población de 10 individuos. A partir de la generación 22 todos los individuos son iguales, lo cual puede observarse en que el valor de la velocidad promedia es igual al de la velocidad máxima de la población. Esto significa que el progreso en la evolución de la aptitud a partir de este punto se debe únicamente a procesos

estocásticos de cruce y mutación sobre estos individuos. Se debe resaltar que no se presentó este caso en ninguna otra simulación de las ejecutadas.

Respecto a las velocidades máximas que alcanzaron las soluciones candidatas puede observarse una tendencia ascendente en proporción al tamaño de la población. Para una población de 10 individuos la velocidad máxima en la última iteración fue de 1,125988184444188 celdas por iteración; para una de 50 individuos, 1,4917090279593956 celdas por iteración; para una de 100 individuos, 1,5292017341259452 celdas por iteración; para una de 200 individuos, 1,5269481141198997. En la gráfica 25 puede observarse las velocidades máximas tras el proceso evolutivo por cantidad de individuos en la población. Se ha añadido una regresión lineal a la gráfica para mostrar la tendencia de crecimiento.

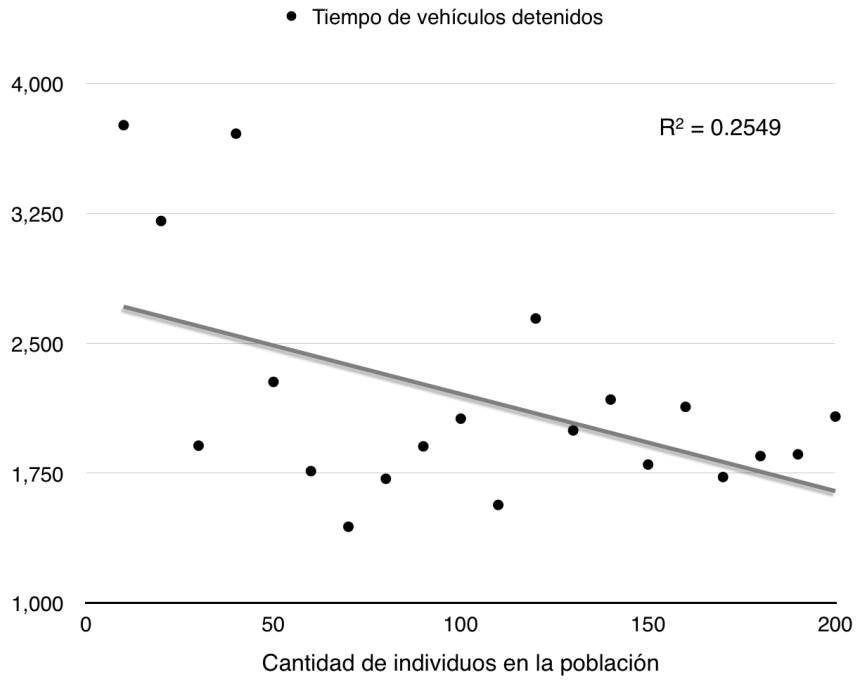
Figura 25. Velocidad máxima de solución candidata seleccionada tras proceso de evolución según cantidad de individuos en la población.



Por otro lado, también es importante observar el comportamiento del otro parámetro de optimización dentro de las simulaciones: el tiempo promedio en que los vehículos están detenidos durante la simulación. En la Figura 26 se muestran los mínimos de tiempo en que los vehículos estuvieron detenidos bajo la solución candidata seleccionada. También se añadió una regresión lineal a la gráfica para mostrar la tendencia.

4 Análisis de espacio de búsqueda. Como ya se ha expuesto en secciones anteriores de los resultados, el rendimiento del algoritmo depende de los parámetros con los que se ejecute: tamaño de la población y cantidad de generaciones. Sin embargo, existe un parámetro crucial inscrito en el problema a resolver: el tamaño del espacio de búsqueda. A continuación cómo varía el espacio de búsqueda en relación

Figura 26. Tiempo promedio en que los vehículos están detenidos en la solución candidata seleccionada tras proceso de evolución según cantidad de individuos en la población.



a los otros parámetros.

Si el probleama busca optimizar para un periodo T, en un mapa de i intersecciones con k luces cada uno, entonces el espaciod de búsqueda S es de:

$$S = \binom{T \times i}{k} = \frac{(T \times i)!}{(k!(T \times i) - k)!} \tag{VI..1}$$

Para modelar la forma en que incrementa el espacio de búsqueda según incrementa el período o la cantidad de intersecciones en el mapa, sea $n = T \times i$, entonces:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{VI..2}$$

$$\binom{n+1}{k} = \frac{(n+1)!}{k!((n+1)-k)!} \tag{VI..3}$$

$$= \frac{n!(n+1)}{k!(n-k)!(n-k+1)} \tag{VI..4}$$

$$\binom{n+1}{k} = \binom{n}{k} \times \frac{n+1}{n-k+1} \tag{VI..5}$$

Así, si se tiene $T = 10$, $i = 4$ y $k = 2$, como en las simulaciones anteriores, el espacio de búsqueda es $S = 780$. Si se incrementa el período a optimizar, por ejemplo, a $T = 15$, $i = 4$ y $k = 2$, $S = 1770$. Así, el espacio de búsqueda se incrementa notablemente al incrementa el período o la cantidad de intersecciones. Para tener como referencia, utilizando $T = 15$, $i = 8$ y $k = 2$, entonces $S = 7140$; y $T = 15$, $i = 16$ y $k = 2$, entonces $S = 28680$

5 Rendimiento de algoritmo en tres mapas distintos. Se realizaron pruebas de mayor escala utilizando tres diferentes mapas, que se muestran en la Figura 27. Se ejecutó una simulación con una población de 100 individuos, 50 generaciones y un período de 15 iteraciones. Estas restricciones se debieron a la falta de infraestructura adecuada para realizar mejores búsquedas. A continuación se muestran los resultados en estos mapas.

Todas las pruebas se ejecutaron en una computadora con sistema operativo Mac OSX 10.11, Procesador Intel de 2.5GHz Core i7, y memoria RAM de 16 GB 1600 Mhz DDR3, dedicada exclusivamente al algorimo genético. Se presentan los tiempos de ejecución registrados por `CProfiler` de Python como referencia únicamente.

En el mapa (a) de la Figura 27 se presenta el mapa básico con que se realizaron los análisis y pruebas previamente presentados, pues es el que requiere menos poder computacional para calcular la aptitud. El mapa tiene cuatro intersecciones y sus calles son de un solo carril. Sin embargo, el periodo para el que se optimizó era de 15 iteraciones en vez de 10, lo cual tiene un impacto grande en el rendimiento del algoritmo, como se presentó en la sección 4. Los resultados de velocidad máxima y velocidad promedio por generación pueden verse en la Figura 28. El tiempo de ejecución fue de 1180.377 segundos.

Para la simulación del mapa (b) de la Figura 27 el mapa presenta 8 intersecciones y dos carriles en cada calle. Esto introduce dinámicas más sofísticadas al comportamiento del simulador. Los resultados se presentan como velocidad máxima y velocidad promedio por generación en la Figura 29. El tiempo de ejecución fue de 2679.73 segundos.

Para la simulación del mapa (c) de la Figura 27 el mapa presenta 16 intersecciones y dos carriles en cada calle. Los resultados se presentan como velocidad máxima y velocidad promedio por generación en la Figura 30. El tiempo de ejecución fue de 5142.968 segundos.

I DISCUSIÓN

Según Mitchel (1998), no se tiene información abundante de cualquier problema que sea suficientemente complejo como para intentar resolverlo con un algoritmo genético. La optimización de ciclos de semáforo es un problema *NP-hard* que ha sido abordado con diferentes enfoques y técnicas, entre las cuales los algoritmos genéticos han sido la elección de numerosos autores, como ya se mencionó en antecedentes. Sin

embargo, los detalles de diseño e implementación del algoritmo genético varían según las condiciones que se busca optimizar y el método para calcular la aptitud. En el presente trabajo se optó por realizar una evaluación de la aptitud a partir de un simulador basado en TCA desarrollado en otro módulo del megaproyecto, lo cual propone desafíos específicos de diseño. En esta sección se exponen las decisiones de diseño guiadas por resultados del algoritmo y se explican dificultades al aplicar el algoritmo con parámetros más demandantes.

Como se mencionó en el marco teórico, dos de los componentes más importantes de un algoritmo genético son la codificación de soluciones candidatas y la estrategia de selección. Para el primero, se siguió el esquema de Sánchez-Medina *et al.* (2004) que permitía la creación de cromosomas válidos aún después de aplicar operadores de cruce y de mutación. Durante la implementación solo fue necearia la normalización de los IDs de las luces de semáforo del simulador para adoptar esta codificación sin dificultades.

Por otro lado está la estrategia de selección. Primero se planteó utilizar una estrategia elitista, según la propuesta de Sánchez-Medina *et al.* (2004), quien también implementó un algoritmo genético que evaluaba la aptitud de las soluciones candidatas utilizando un simulador basado en TCA. Bajo este esquema, solo las soluciones candidatas que presentaran mejor aptitud podrían reproducirse y pasar a la siguiente generación. Sin embargo, esto llevaba a convergencias prematuras en esta implementación. Como puede observarse en la Figura 22 la estrategia elitista tiende a estancarse en soluciones candidatas subóptimas.

Mitchel (1998) planteaba que varios autores han obtenido buenos resultados seleccionando a un porcentaje de los mejores individuos de la población para la siguiente generación, de modo que no se pierdan por procesos estocásticos. Por su lado, Turk *et al.* (2009) utilizó una selección tradicional de ruleta. Por tanto, se propuso una estrategia híbrida que empleará tanto un método determinístico de selección elitista para una minoría de la población y un método estocástico para el resto de la población.

Para el método de selección estocástico, en un inicio se había contemplado una selección por torneo simple, pues requiere menos computo que un método de selección proporcional a la aptitud. Sin embargo, en ocasiones se observó que contribuía a una convergencia prematura al reducir la variedad de la población. En la Figura 22 que la selección elitista tiene a estancarse en soluciones candidatas subóptimas. Por el contrario, el permitir deliberadamente la variedad genética en la población es posible explorar más parte del espacio de soluciones candidatas para así encontrar soluciones más óptimas.

Respecto a las pruebas de rendimiento del algoritmo según la población, cabe resaltar la relación entre el número de individuos en la población y la solución candidata óptima encontrada por el algoritmo, que se ilustra en la Figura 25. Mientras mayor es la población, es más probable que se encuentre una solución candidata con una mejor aptitud. Sin embargo, puede también observarse que elegir una población demasiado grande podría conducir a un gasto de recursos computacionales innecesario, como puede observarse en la

similtud de resultados del algoritmo con población de 100 individuos y con 200 individuos en la Figura 24.

Otra relación relevante que puede observarse en estas pruebas de rendimiento es la relación entre la aptitud de la población y la generación de la población. Es una relación más delicada, pues aumentar la cantidad de generaciones no asegura una mejor aptitud. En la Figura 25 es posible observar que conforme se avanza en las generaciones se acerca más la población a la solución óptima. Sin embargo, es posible que el algoritmo experimente una convergencia prematura, como es el caso de la simulación con una población de 10 individuos.

Por otro lado, las pruebas en tres diferentes mapas dieron resultados poco prometedores utilizando los parámetros de simulación con una población de 100 indidividuos, 50 generaciones y período de 15 iteraciones. Tras analizar el crecimiento del espacio de búsqueda que conllevan estas configuraciones no sorprende que los resultados encontrados hayan estado lejos de ser óptimos. En la Figura 28 se logró encontrar una leve mejora de la población conforme pasan las generaciones, pero de forma poco efectiva. En las Figuras 29 y 30 se evidencia un estancamiento de la población en valores subóptimos, un fenómeno parecido a los ocurrido en la población de 10 individuos expuesta anteriormente.

Para las simulaciones los resultados anteriores se había utilizado una configuración con 780 soluciones candidatas. En estos tres mapas el espacio de búsqueda es de 1770, 7140, y 28680 soluciones candidatas, respectivamente. Por tanto, era necesario ejecutar el algoritmo genético en cada una con poblaciones más numerosas y por más generaciones para encontrar soluciones óptimas.

J CONCLUSIONES

Respecto al diseño del algoritmo genético, es posble concluir que la selección de componentes debe hacerse de acuerdo a la naturaleza del problema de solución y el método para evaluación de la aptitud. En el caso concreto de este algoritmo para la optimización de luces de semáforo, fue decisivo elegir una codificación de las soluciones candidatas que sea compatible con el simulador basado en TCA.

Adicionalmente, la estrategia de selección ha sido ajustado tras obtener resultados empíricos, de modo que se introducen variaciones genéticas en la pobación para explorar más el espacio de soluciones candidatas. Por ello la estrategia híbrida ha provisto un rendimiento favorable.

Por otro lado, no es suficiente diseñar y refinar un algoritmo genético apropiado para obtener resultados óptimos. Se deben tomar en cuenta tres parámetros que definen en buena medida el éxito de la búsqueda de una solución óptima: el tamaño del espacio de las soluciones candidatas, el tamaño de la población y la cantidad de generaciones. Se ha observado que el primer parámetro guía la elección de los otros dos, pues este es inherente al problema y no se tiene total control sobre el mismo.

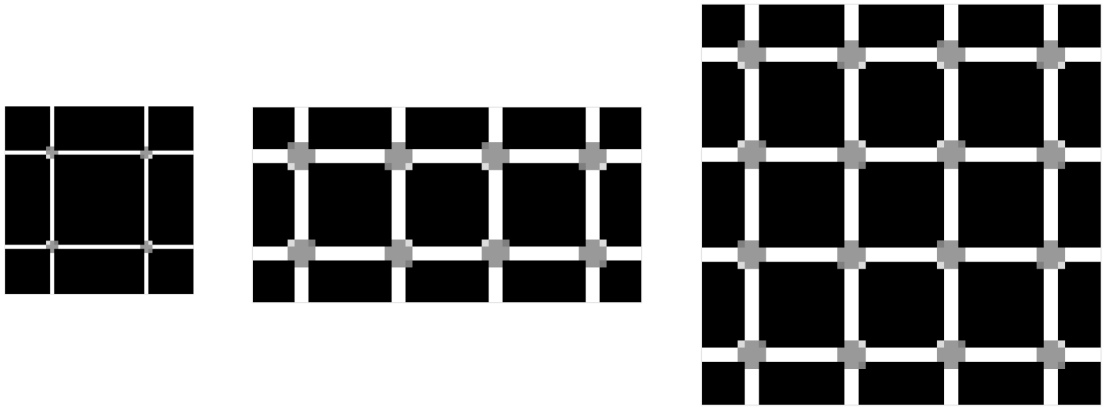
K RECOMENDACIONES

El costo de evaluación de la aptitud es muy elevado, por lo que podrían descartarse soluciones candidatas que son teóricamente válidas pero inviables en el manejo del tránsito real. Un ejemplo de esto sería una calendarización en la que se cambie de luz cada iteración. Utilizando conocimientos del dominio del tránsito pueden filtrarse dichas soluciones para descartarlas antes de evaluar su aptitud, de modo que solo se exploren las que tienen más probabilidad de ser exitosas.

La elección del tamaño de población y cantidad de generaciones en el algoritmo respecto al tamaño del espacio de soluciones candidatas es crucial para el éxito el algoritmo. Por eso se sugiere formalizar una técnica para determinar la cantidad de individuos en la población y número de generaciones más acertado para obtener soluciones más aptas y evitar desperdiciar recursos.

Por otro lado, a pesar que un algoritmo genético es una forma más rápida de explorar el espacio de soluciones candidatas de un problema, su ejecución requiere recursos notables de computación para obtener soluciones óptimas. Se recomienda implementar una infraestructura que evalué aptitudes de individuos paralelamente. No sería necesario re-implementar el algoritmo pues DEAP provee herramientas que permiten la paralelización utilizando SCOOP (*Scalable COncurrent Operations in Python*) (Yannick *et al.*, 2014).

Figura 27. (a) Mapa de 4 intersecciones con un solo carril. (b) Mapa de 8 intersecciones multicarril. (c) Mapa de 16 intersecciones multicarril. En todos hay un semáforo con luces para cada calle en cada intersección.



(a) (b) (c)

Figura 28. Gráfica de velocidad máxima y velocidad promedio por generación en un mapa de 4 por 4 intersecciones

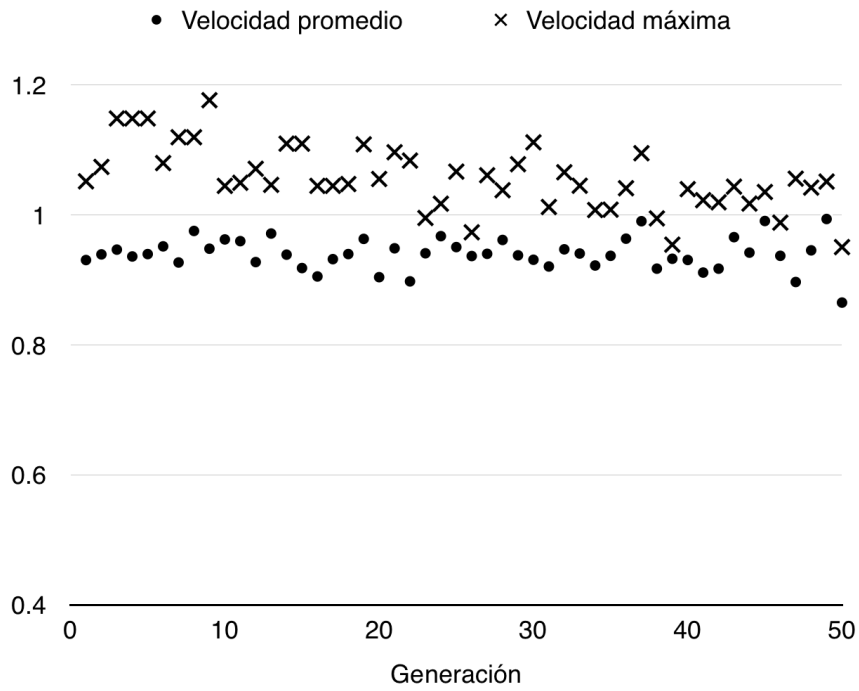


Figura 29. Gráfica de velocidad máxima y velocidad promedio por generación en un mapa de 8 por 4 intersecciones.

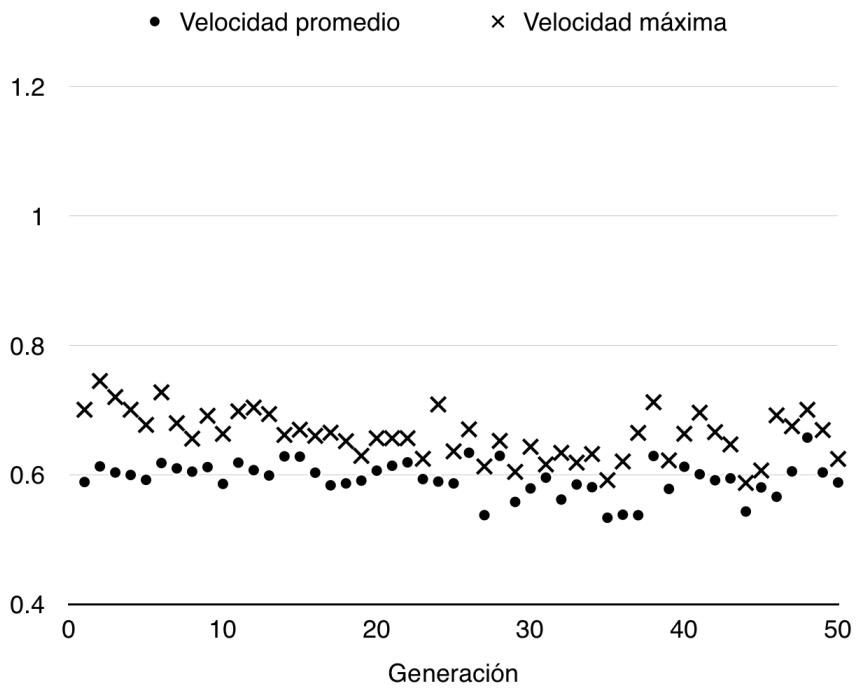
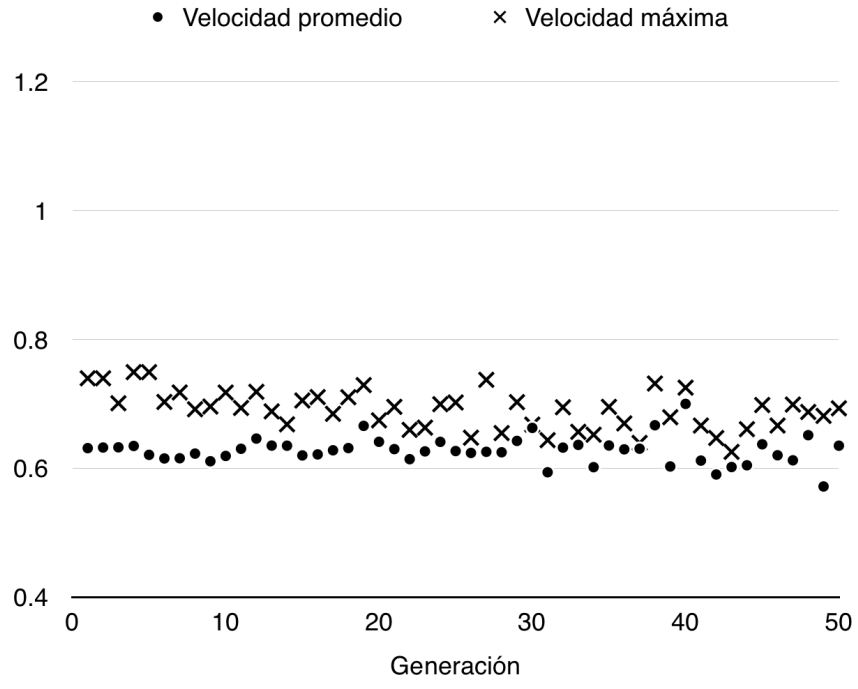


Figura 30. Gráfica de velocidad máxima y velocidad promedio por generación en un mapa de 8 por 8 intersecciones.



VII. DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA MULTIAGENTE CON APRENDIZAJE POR REFUERZO PARA OPTIMIZAR EL TRÁFICO

A INTRODUCCIÓN

El incremento poblacional en las ciudades urbanas alrededor del mundo ha ocasionado también el aumento de vehículos en circulación. En el caso de la ciudad de Guatemala, el tráfico es un problema severo a causa de las aglomeraciones urbanas y la infraestructura suficiente para soportar altas cantidades de vehículos. Se ha intentado manejar el congestionamiento a través de la Policía Municipal de Tránsito (PMT), asignando agentes de tránsito para dirigir en semáforos a las horas pico de la ciudad. Los agentes deben ingeniarselas para dar vía el tiempo suficiente a cada calle, además de coordinarse con agentes de otras intersecciones a través de radiocomunicación.

Este megaproyecto tiene como visión apoyar a estos agentes en la toma de decisiones para la dirección de tránsito vehicular. El megaproyecto consiste en dos partes: hardware y software. La parte de hardware involucra un prototipo de la infraestructura que se debería implementar para apoyar a los agentes, como son los sensores de tránsito, un sistema centralizado con todas la información del tráfico en la ciudad y la comunicación con los agentes. La parte de software se exploran diferentes algoritmos de inteligencia artificial para optimizar el tráfico. Además, se presenta un basado en *Traffic Cellular Automata (TCA)*, que será utilizado para la realización de prueba para los algoritmos de inteligencia artificial.

En el presente módulo se busca diseñar e implementar un sistema multiagente, cuya función es aprender el comportamiento del entorno y reaccionar realizando cambios en las luces de los semáforos. El sistema se basó en una perspectiva jerárquica donde existe un agente inteligente en cada intersección estos son coordinados por otro agente. Los agentes inteligentes utilizan un tipo de algoritmo de intelingecia artificial llamado aprendizaje por refuerzo. Con base en la acciones que realiza el agente en un estado del entorno, el agente recibe recompensas que le ayudan a determinar las mejores acciones a realizar en el futuro en caso las mismas circunstancias del entorno vuelven a presentarse.

Se analizó el rendimiento del algoritmo tratando de optimizar la velocidad promedio y se concluyó que los agentes logran reducir la cantidad de tiempo de espera en sus intersecciones, sin embargo el agente

coordinador al no tener un método de selección riguroso para determinar qué acciones logran optimizar la velocidad promedio de todo el mapa.

B OBJETIVOS DEL MÓDULO

1 Objetivo general del módulo

Diseñar e implementar un algoritmo de inteligencia artificial que mejore el flujo de los automóviles dentro de una cuadrícula.

2 Objetivos específicos del Módulo.

- Investigar casos de éxito en sistemas multiagentes aplicados al tránsito urbano
- Investigar y elegir el algoritmo de aprendizaje por refuerzo para los agentes
- Diseñar los diferentes tipos de agente y la jerarquía en el sistema
- Establecer las acciones permitidas de cada agente
- Diseñar la comunicación entre los agentes del sistema
- Implementar el agente con un algoritmo de aprendizaje por refuerzo
- Implementar el sistema multiagente sobre un simulador de TCA
- Implementar rutinas de entrenamiento para los agentes.

C JUSTIFICACIÓN DEL MÓDULO

Diseñar un sistema multi agente para esta coordinación es conveniente ya que cada semáforo asumiría el papel de un agente que esté informándose del paso de vehículos de su intersección. Pero para que haya cooperación entre los diferentes agentes, se necesita a un coordinador que esté comunicándose con todos. Por eso se busca que el sistema tenga una arquitectura jerárquica, donde el coordinador será un agente que esté calculando cómo optimizar los resultados de las métricas y cada semáforo será un agente que tomará sus decisiones pero siempre debe darle prioridad a las decisiones del coordinador (France y Ghorbani , 2003). La ventaja de esta metodología es la capacidad del sistema de aprender con el tiempo por lo que con el tiempo irá mejorando en la toma de decisiones y mejorando el flujo de vehículos.

D ANTECEDENTES

1 Aprendizaje en sistemas multiagentes.

La importancia del aprendizaje reforzado en la teoría de aprendizaje psicológica se basa en aprendizaje de prueba y error. Ante varias respuestas a la misma

situación, las que vienen acompañadas de satisfacción al organismo estarán firmemente conectadas, así en caso la sitaución se repite, es más probable que actúen de la misma manera. Las que vienen acompañado de disgusto, el organismo, este debilitará su conexión a la situación y así cuando vuelva a estar en esa situación evitará actuar igual. Edward Thorndike definió esta esencia de la prueba y error en el comportamiento de un animal y lo llamó la ”Ley de efecto”porque describe el efecto de reforzar eventos en la tendencia de elegir acciones (Marsland , 2014). La idea que en programación una computadora aprenda por prueba y error existe desde 1950 con trabajos de Minsky y de los autores Farley y Clark (Sutton y Barto , 1998). Misnky en su disertación discute sobre modelos computacionales de aprendizaje reforzado. Farley y Clark describieron una máquina de aprendizaje de redes neurales diseñado para aprender con prueba y error. Misnky en 1961, publica un artículo sobre inteligencia artificial donde discute los diferentes problemas de aprendizaje reforzado, incluyendo el ”problema de asignación de crédito”. Este problema trata sobre cómo distribuir crédito por éxito entre las diferentes decisiones involucradas para producirlo

Otro camino de investigación de importancia son los métodos de diferencias temporales. Estos métodos se distinguen por que se basan en la diferencia entre estimaciones sucesivas temporales, como la probabilidad de obtener cierta recompensa. Estos métodos se basan en un término psicológico denominado reforzador secundario, que es un estímulo pareado con un reforzador primario como comida y dolor y, como resultado, viene con propiedades de refuerzo similares. Arthur Samuel en 1959 fue el primero en proponer e implementar un método de aprendizaje que incluye ideas de diferencias temporales. No hubo muchos estudios sobre aprendizaje basado en prueba y error hasta en la década de 1970, donde Klopf incorpora el aprendizaje de prueba y error con componentes de aprendizaje de diferencias temporales. A Klopf le interesaba los principios que terminarían escalando en aprendizaje en sistemas grandes, la noción de refuerzo local donde subcomponentes pudieran reforzarse entre ellos. La idea de refuerzo generalizado, por lo cual cada componente observaba todas sus entradas en términos de refuerzo. Sutton desarrollo estas ideas de Knopf describiendo reglas de aprendizaje basado en cambios en predicciones sucesivas temporales.

Por último, existía otra área de investigación sobre problemas de control óptimo y su solución utilizando funciones y programación dinámica. Este campo viene desde los años 1950 para describir problemas de diseñar un controlador para minimizar una cantidad de comportamiento de un sistema dinámico. Un enfoque de este problema fue desarrollado por Richard Bellman y otros autores. Este enfoque consiste en usar conceptos del estado dinámico de un sistema y una función de valor que define una ecuación funcional. Esta ecuación es llamado actualmente como la ecuación de Bellman. La clase de métodos para resolver problemas de control óptimo utilizando esta ecuación se conoce como programación dinámica. Bellman también introdujo una versión discreta estocástica del problema de control óptimo actualmente llamado procesos de decisión de Markov (MDP). En 1960 Ron Howard diseñó el método de iteración de la política para MDPs.

Estos tres campos terminaron uniéndose para dar origen al campo moderno de aprendizaje por refuerzo.

Los problemas de aprendizaje por refuerzo tienen relación con problemas de control óptimo, particularmente los modelados como MDP. En 1989, las investigaciones en los campos de control óptimo y diferencias temporales fueron unidas por Chris Watkin con el desarrollo de Q-learning. Su trabajo contribuyó masivamente en la investigación de aprendizaje por refuerzo y otros campos de inteligencia artificial.

2 Optimización de tráfico utilizando agentes que aprenden. En muchas ciudades del mundo, el problema de tráfico y congestión en las intersecciones ha ido en aumento debido al crecimiento poblacional y la migración a estos centros urbanos. El crecimiento puede ser tan rápido que no se logra expandir y mejorar la infraestructura y la capacidad de vehículos por lo que se tiene que buscar opciones para reducir el tráfico con los recursos presentes. Una alternativa es la instalación de una sistema de manejo en las señales de las intersecciones para reducir los tiempos de espera y aumentar el flujo de vehículos en las calles. Muchos sistemas de control de tráfico urbano en intersecciones han sido desarrolladas. Entre los aspectos que se manejan son las decisiones sobre la indicación de señales y su secuencia. Distintos enfoques de diseño, arquitectura y técnicas de sistemas de control de tráfico. Tahilyani *et. al.* (2013) presentaron un estudio de diferentes técnicas de Soft Computing.

Entre las técnicas más comúnes es la basada en aprendizaje y cognición. Diferentes metodologías en la utilización agentes han sido presentadas, siendo la de multiagentes la más común. France y Ghorbani (2003) propusieron un diseño de sistema multiagente jerárquico donde las intersecciones son optimizadas por agentes que aprenden y son coordinadas por un agente común que determina el óptimo global respecto a los óptimos locales. El coordinador compara la densidad de vehículos en cada intersección para evaluar y evitar que se acumule el tráfico. Mannion *et. al.* (2015) por otra parte presentaron un sistema multiagente donde el aprendizaje es paralelo. Los agentes en las intersecciones son influídos por la información que otros agentes comparten.

Junto al diseño de jerarquía y roles de sistemas multiagente, también se debe presentar los algoritmos el cual el sistema utiliza para aprender. balaji presenta un sistema distribuido multiagente para optimizar los tiempos de señal verde para reducir el tiempo de viaje y retrasos de los vehículos. Los autores utilizaron el software PARAMICS para simular intersecciones en Singapur. Chin *et. al.* (2011) y Gregoire *et. al.* (2007) presentaron su solución utilizando el algoritmo de Q-learning con búsqueda codiciosa. Pham *et. al.* (2013) también presentan diferentes metodologías de aprendizaje para conocer y optimizar el flujo de tráfico. Entre los algoritmos que trabajaron está SARSA, un algoritmo de aprendizaje por refuerzo, y el marco de trabajo DCEE, que busca un balance entre utilizar las mejores configuraciones y buscar posibles mejoras. Wiering (2000) presentó un sistema multiagente implementando aprendizaje por refuerzo. Por su parte, los vehículos tendrían la capacidad para comunicarse con los semáforos y compartir información de ubicación y destino. La función de valor es en base al carro y los tiempos de espera esperados. Todas estas propuestas buscaban reducir los tiempos de espera que cada vehículo dado la congestión de intersecciones

o semáforos en rojo. Durante la investigación de este trabajo, no se encontró una propuesta que cambie la función de valor en base a la velocidad promedio de vehículos en las calles.

E DELIMITACIÓN

Este megaproyecto consiste en el desarrollo del prototipo de una herramienta de apoyo para los policías municipales de tránsito, no del control automático de dispositivos electrónicos relacionados al flujo vehicular. Esto es, el proyecto se limita a la implementación del sistema de sensado, dispositivo portátil, desarrollo de simulador y comunicación entre los diversos módulos. Debido a esto, la parte de computación se desarrolla en paralelo a la de electrónica, sin interconectar ambas partes. En esta última, se seleccionará un sensor, un medio de comunicación y sistemas de control de bajo costo y que permitan el funcionamiento adecuado del sistema total.

El sistema multiagente será desarrollado en base a un modelo implementado por otro módulo del megaproyecto, que establece las reglas de comportamiento de vehículos y los semáforos que regulan el tránsito. Se investigará diferentes estrategias de sistemas multiagente y de aprendizaje para determinar los más efectivos y los que puedan ser ejecutados en la práctica. Se diseñará la estructura del sistema y el algoritmo de aprendizaje en base a un modelo de tránsito con el objetivo de reducir la cantidad de tiempo de espera perdido por los diferentes vehículos.

F MARCO TÉORICO

1 Agentes inteligentes. La inteligencia artificial puede definirse como el estudio y diseño de máquinas o software especializado denominado como *agente inteligente*. Un agente se define como cualquier objeto en un entorno con capacidad de percibir diferentes aspectos de dicho entorno a través de sensores. Además, tiene la capcidad para realizar acciones en el entorno e interactuar, ya sea con las diferentes características de los entornos o con otros agentes que también participan. Al momento de percibir, denominamos como .entradas"todas esas características que el agente toma en consideración para procesar y realizar una acción. Esta acción al momento de realizarla le asignamos el término de "salida". Durante este trabajo, se referirá a un agente inteligente simplemente como agente.

Cuando mencionamos el entorno nos referimos a diferentes aspectos que el agente va a estar sometido, ya sea en una simulación o en producción en un ambiente real. Wooldridge (2009) define las siguientes características que están presentes en los diferentes tipos de ambientes:

Accesibilidad: Generalmente los agentes que son implementados en ambientes del mundo real no tienen acceso completo de todo su entorno. Muchos dependen de sensores para obtener cierta información para lograr acercarse a los cumplimientos de los objetivos que se quieren cumplir.

Determinístico o estocástico: Nos referimos a un sistema determinístico en el cual al realizar una acción obtendremos siempre el mismo resultado.

Estático o dinámico: Los ambientes estáticos no cambian a menos que el agente realice una acción mientras que los dinámicos estarán sujetos a diferentes cambios sin importar si el agente actúa o no.

Discreto o continua: Un entorno discreto se limita a un número de acciones o estados mientras que en continuos las posibles acciones o estados son infinitas.

Los agentes inteligentes para ser considerados como tal necesitan mostrar ciertos comportamientos. Los agentes inteligentes son autónomos por lo que no necesitan intervención humana para funcionar. También presentan la noción de comportamiento. Es decir, no solo llevan a cabo diferentes acciones sino que sugieren y seleccionan qué acciones llevar a cabo. Además de autónomos, los agentes presentan otras características para que puedan ser considerados como inteligentes:

Reactivos Los agentes perciben el entorno y responden de para que se den cambios.

Proactivos Los agentes no solo actúan en respuesta al ambiente, sino que también deben actuar ante situaciones convenientes que puedan mejorar y acercarse a los objetivos que el agente debe cumplir.

Sociales Los agentes deben poder interactuar (en caso sea necesario) con otros agentes, o bien, con seres humanos para cumplir objetivos a través de la cooperación o competitividad.

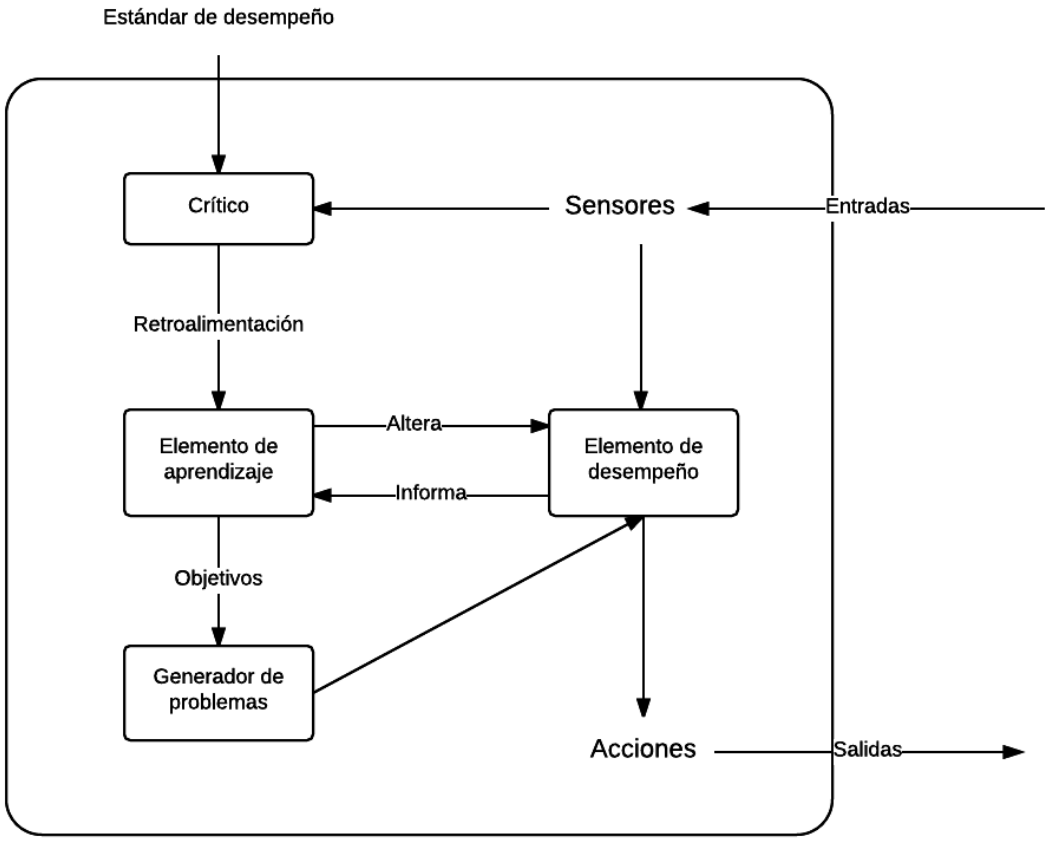
Los agentes son programados con una función que tiene como propósito mapear las entradas a acciones. Estas entradas son recibidas por medio de sensores u otros medios para obtener esa información. Dependiendo del tipo de función que se utilice será el comportamiento del agente y su forma de razonamiento. En este proyecto se trabajó con agentes de aprendizaje que tienen la capacidad de medir el rendimiento de sus componentes para realizar mejores acciones.

a Agentes de aprendizaje. Normalmente los agentes empiezan sin tener conocimiento del entorno en el que están. Los agentes de aprendizaje pueden empezar sin conocer nada y conforme van explorando van a desempeñar mejor. Estos agentes logran aprender cómo el entorno evoluciona y a través de la observación de sus acciones el agente logra aprender sus consecuencias. En la Figura 31 se muestran los diferentes componentes que se divide el agente de aprendizaje. El agente tiene un elemento de aprendizaje que se encarga de realizar mejoras y el elemento de desempeño se encarga de elegir una acción. Este último recibe entradas y decide qué acción realizar. El elemento de aprendizaje recibe retroalimentación del crítico acerca de cómo se está desempeñando y determina el elemento de desempeño debe modificarse para rendir mejor en el futuro.

El crítico informa al elemento de aprendizaje qué tan bien está desempeñando respecto al estándar de desempeño. El estándar de desempeño distingue las entradas como una recompensa o una penalización que provee retroalimentación del comportamiento del agente. Por último está el generador de problemas que se responsabiliza de sugerir acciones que puedan llevar a nuevas experiencias. Esto evita a que el elemento de desempeño siempre elija la misma acción que considere mejor. Si el agente explora puede llegar a encontrar mejores acciones con mejores resultados.

El objetivo de estos agentes es optimizar sus objetivos a través de recompensas

Figura 31. Diagrama donde se describe la interacción de componentes en un agente de aprendizaje



(Russell y Norvig , 2003)

2 Sistemas multiagente. Ciertos problemas no pueden ser resueltos por un solo agente o se requiere de inmensas cantidades de computaciones para poder elegir la acción a realizar. Se busca reducir la carga de información que un agente deba aprender o que un agente se especialice en ambientes reducidos. Por lo tanto, crear más de un agente para resolver un problema resulta conveniente al poder distribuir tareas

y lograr decentralizar los cálculos necesarios para obtener una solución. Los agentes en entornos reales no trabajan por sí solos. Wooldridge (2009) expone diferentes situaciones en donde la aplicación de agentes es apropiado:

- En entornos muy grandes, abiertos, inciertos o dinámicos, en donde sistemas capaces de ejecutar acciones de forma flexible y autónoma son la única solución.
- En entornos donde son modelados como una sociedad de agentes que están cooperando o compitiendo entre ellos.
- En muchos ambientes donde los medios de control y distribución de datos resulta complicado si no imposible.

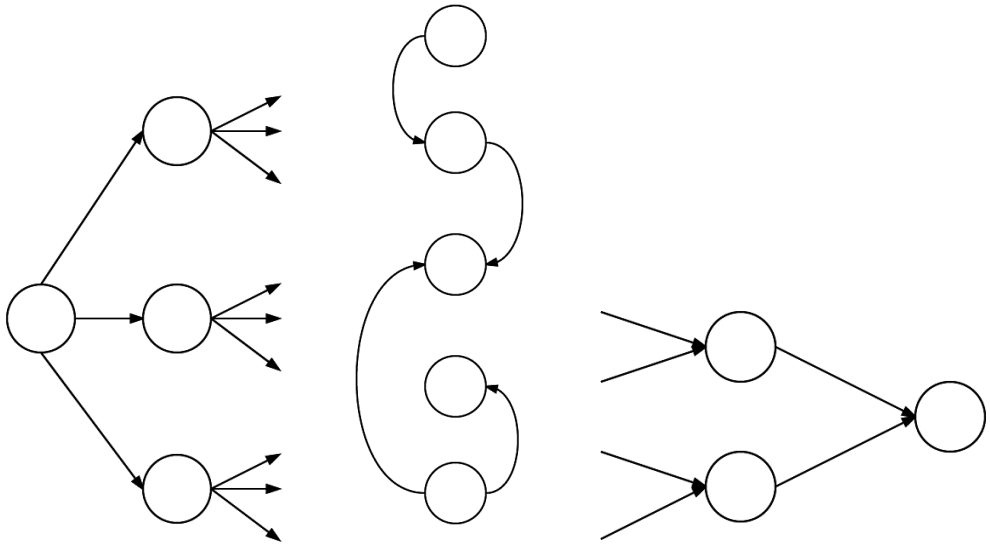
Los agentes pueden cooperar y resolver problemas en paralelo. En el diseño de sistemas multiagente se necesita tomar en cuenta dos problemas principales:

- Se necesita medir el sistema sobre ciertas métricas. La coherencia puede medirse por calidad de la solución, eficiencia en el uso de recursos o la degradación del desempeño ante la incerteza.
- Se busca que los agentes no intervengan el desempeño de otros cuando se quiera alcanzar un objetivo personal cuando se quiere cumplir el objetivo global.

Para que los agentes logren cooperar se necesita primero subdividir el problema en pequeños subproblemas. Esta descomposición generará una jerarquía para que subproblemas consecuentes continuen subdividiendo hasta llegar a cierta granularidad adecuada para que agentes individuales puedan trabajar. Hay diferentes paradigmas en sistemas multiagentes que proponen metodologías para la subdivisión de problemas y generalmente estas tareas son realizadas por los mismos agentes. Los subproblemas identificados son solucionados individualmente por un agente. Para cumplir con el objetivo principal de todo el sistema, estos agentes deben compartir información que pueda resultar útil en buscar la solución al subproblema que se está resolviendo. Cuando los agentes resuelven el problema individual, se transmite la solución jerárquicamente y se va sintetizando en diferentes niveles hasta llegar al último nivel donde se juntan todas las soluciones. En la Figura 32 se resume las tres fases de la resolución de problemas y la interacción entre angente de forma jerárquica.

La resolución de problemas a través de la cooperación entre agentes da lugar a problemas de inconsistencia. Cada agente tiene su propio conocimiento del entorno y del objetivo a cumplir o maximizar. Hay diferentes formas de planificar un sistema multiagente para resolver estos problemas de coordinación. Sen y Weiss (1999) distinguen los enfoques de planificación de sistemas y categorización de aprendizaje en

Figura 32. Las tres fases para la resolución de problemas de forma distrbuida



(a) Descomposición del problema (b) Solución del subproblema (c) Síntesis de la solución

dos: aprendizaje centralizado y descentralizado. El primero, también llamado como aprendizaje aislado, consiste en centralizar todo el proceso de aprendizaje en todas sus partes en un solo agente. Este agente es completamente independiente de otros agentes. Un aprendizaje descentralizado (o aprendizaje interactivo) toma varios agentes y los somete a un mismo proceso de aprendizaje, por lo que las actividades que constituyen el proceso de aprendizaje son ejecutadas por los diferentes agentes. La diferencia consiste en que el aprendizaje descentralizado requiere de la presencia de distintos agentes capaces de llevar a cabo las actividades establecidas. En un sistema multiagente diferentes agentes con aprendizaje centralizado que intentan obtener diferentes o los mismos objetivos pueden estar activos al mismo tiempo. Además, pueden haber diferentes grupos de agentes que están involucrados en diferentes procesos de aprendizaje descentralizado. Asimismo, los objetivos de aprendizaje planteados por estos grupos pueden ser diferentes o iguales. También es importante ver que un agente puede estar involucrado en distintos procesos centralizados y/o distribuidos al mismo tiempo.

Los agentes deben aprender a coordinar sus actividades. Los tres principales tipos de aprendizaje son supervisado, no supervisado y por refuerzo. Estos métodos se distinguen por el tipo de retroalimentación quer el crítico provea al elemento de aprendizaje. En aprendizaje supervisado el crítico provee la salida correcta. En no supervisado, no se provee ninguna retroalimentación. En aprendizaje por refuerzo, el crítico provee una evaluación (la recompensa) de la salida del agente. También hay otro tipo de aprendizaje llamado evolucional. Basado en el concepto de evolución biológica, tal como los organismos se adaptan para mejorar sus tasas de supervivencia y la posibilidad de tener descendencia en su entorno (Marsland ,

2014). El aprendizaje evolucional es similar al aprendizaje por refuerzo dado que ambos se basan en un aprendizaje orientado a una recompensa. Tal como Panait y Luke (2005) exponen en su trabajo, la mayoría de investigación en aprendizaje de sistema multiagente se enfocan en métodos basados en recompensa. El aprendizaje por refuerzo estiman funciones de valor mientras que la búsqueda estocástica de la computación evoucional aprenden comportamientos directamente sin utilizar funciones de valor.

3 Aprendizaje por refuerzo. Los métodos de aprendizaje por refuerzo son útiles donde la información de refuerzo (recompensa o penalización) se proveen después de que se haya llevado a cabo una secuencia de acciones en el entorno.Panait y Luke (2005) Entre el aprendizaje supervisado y no supervisado, los algoritmos informan cuando una solución no es correcta pero no informa cómo corregirlo. Se debe explorar e intentar diferentes posibilidades hasta que se encuentre cómo obtener la solución correcta. La exploración es la parte fundamental de cualquier aprendiz por refuerzo. El entorno debe proveer información de qué tan buena es la estrategia del agente que está aprendiendo a través de una función de recompensa. Marsland (2014)

En aprendizaje por refuerzo se mapean los estados a acciones para maximizar una recompensa. El algoritmo recibe la recompensa como retroalimentación. Una vez el algoritmo haya decidido en la recompensa, se necesita escoger la acción que debe llevarse a cabo en el estado actual. Esto es conocido como política. Un desafío del algoritmo por refuerzo es la combinación entre exploración y explotación, es decir, si elegir una acción que lleve a una alta recompensa o realizar otra acción e intentar encontrar otra recompensa mejor. Estas recompensas se conocen como recompensas atrasadas, donde no es posible obtenerlas hasta mucho tiempo después.

Al inicio, se tiene el conjunto A de acciones posibles a ejecutar en un estado y S como el conjunto de estados que el agente puede explorar. Entonces, para un tiempo t el agente puede estar en s_t, donde $s_t \in S$, y al realizar una acción a_t, donde $a_t \in A(s_t)$, se obtendrá una recompensa r_t. Como se desea maximizar las recompensa R_t, la recompensa esperada es la suma de recompensas obtenidas en secuencia:

$$R_t = r_{t+1} + r_{t+2} + \ldots r_T \tag{VII..1}$$

donde T es el tiempo terminal. Dado que frecuentemente se trabajo en entorno infinitos, se aplica un descuento a recompensas futuras. Por lo tanto, el agente buscará maximizar la recompensa descontada esperada:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{VII..2}$$

donde γ es la tasa de descuento y $0 \leq \gamma \leq 1$. Si γ es muy cercano a cero, se dice que el agente es miope y solo se preocupa en maximizar la recompensa inmediata.

En aprendizaje por refuerzo, se desea tener una buena base para realizar predicciones. Por ello se utilizan MDPs. Un sistema de decisi[on tiene la propiedad Markov si el estado actual es suficiente para saber qué acción tomar. La acción que se debe tomar en el siguiente paso no depende de lo que haya pasado anteriormente, solamente en el estado que se encuentra el sistema.

a Procesos de decisión de Markov. Los agentes necesitan un marco de trabajo formal para la "hacer"decisiones. En los últimos años ha aumentado el uso de Procesos de Decisión de Markov en el campo de aprendizaje por refuerzo. Los MDP son eficientes para representar toma de decisiones secuenciales donde la meta es encontrar la mejorar acción para recibir la máxima utilidad. Pierre

Un MDP se define como una tupla (S, A, T, R), donde:

- S es el conjunto de estados.
- A es el conjunto de acciones.
- $T : S \times A \times S \to [0, 1]$ es una función de transición definida como una distribución de probabilidad sobre los estados. La función se define como $T(s, a, s') = P(s_{t+1}|s_t, a_t)$, donde s_{t+1} es el estado en el tiempo $t + 1$ tras haber realizado la acción a_t en el tiempo t en el estado s_t
- $R : S \times A \times S \to \mathbb{R}$ se refiere a la recompensa adquirida tras realizar una transición de estados. De manera similar, $R(s, a, s')$ es la definición de esta función.

La idea de los MDP es que el agente actúa en el entorno con una acción a, en el estado s, y verifica la respuesta del entorno, de la forma del siguiente estado s' y un número real que representa la recompensa que el agente recive de haber escogido dicha acción en el estado s. Para decidir qué acción tomar en cada estado se necesita de la función de la política. Una política es un colección de distribuciones de probabilidad, una para cada paso del proceso: $\pi(s_t, a_{t-1}, s_{t-1}, a_{t-2}, \dots) \in DP(A)$, donde DP es la distribución de probabilidad. Entonces, se define la probabilidad para cada acción que será elegida para cada paso particular del sistema.

La mayoría de algoritmos de aprendizaje por refuerzo están basado en la estimación de funciones de valor, es decir, funciones de estados que estiman que *tan bueno* es para el agente estar en ese estado, o bien, que tan bueno es realizar una acción en ese estado. Esta verificación se define en términos de recompensas futuras que son esperadas. La política π mapea de cada estado $s \in S$, y acción $a \in A(s)$, hacia la probabilidad $\pi(s, a)$ de realizar la acción a estando en s. El *valor* de estar en s bajo la política π, definida como $V^{\pi}(s)$, es el retorno esperado cuando se inicia en s y posteriormente seguir π. La definición formal del valor esperado

es:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1}\middle| s_t = s, \pi\right\}, \tag{VII..3}$$

donde E_{π} denota el valor esperado dado que el agente sigue la política π. Por lo tanto, V^{π} es la función de valor-estado para la política π.

De igual forma, se define el valor de realizar la acción a en el estado s y de seguir la política π se define como:

$$Q^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1}\middle| s_t = s, \pi\right\} \tag{VII..4}$$

Q^{π} es la función de valor-acción para la política π. También se conoce como la función-Q y sus valores como valores-Q.

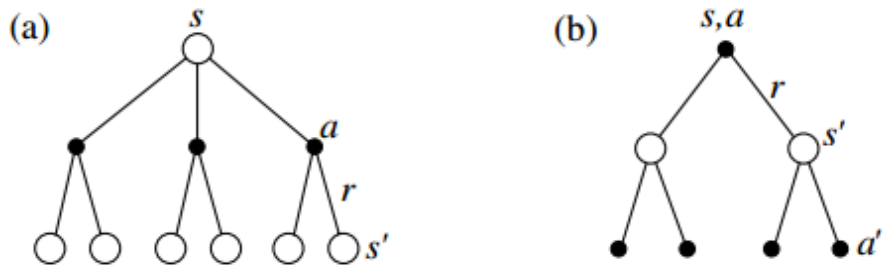
Las funciones de valor en algoritmos de aprendizaje por refuerzo sastifacen relaciones particulares de recursividad. La ecuación VII..3 puede definirse recursivamente en terminos de la ecuación de Bellman (Schwartz , 2014):

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s\right\} \\ &= E_{\pi}\left\{r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s\right\} \\ &= \sum_{s'} T(s,\pi(s),s')\left(R(s,a,s') + \gamma V^{\pi}(s')\right) \end{aligned} \tag{VII..5}$$

La ecuación de Bellman demuestra la relación entre el valor de un estado y los valores de los estados sucesores. En la 33, cada círculo blanco del diagrama (a) representa un estado y cada negro una pareja estado-acción. Desde s, el agente puede tomar una acción de las tres disponibles. Cada una de estas, el entorno responedará con uno de los estados s', junto con la recompensa r. La ecuación de Bellman promedia todas las posibilidades, pesando cada una una por la probabilidad de ocurrencia. Indica que el valor del estado inicial debe ser igual al valor descontado del siguiente estado esperado, además de una recompensa obtenida en el camino.

1) Aprendizaje con métodos libre de modelo. Un aprendizaje por refuerzo basado en modelos busca aprender un modelo explícito y luego utiliza métodos de programación dinámica para calcular la política óptima respecto al estimado del modelo. Mientras tanto, aprendizaje libre de modelo

Figura 33. En estos diagramas se muestran las relaciones que forman la base de una actualización que es el núcleo de los métodos de aprendizaje. (a) V^{π} (b) Q^{π}



se concentra en aprender la función de valor-estado (función-Q) directamente y obtener la política óptima en base a estas estimaciones. La clase algoritmos que utilizan métodos libres de modelo para aprender en MDP se conocen como métodos de diferencias temporales. (Sutton y Barto , 1998)

2) Aprendizaje por diferencias temporales. El aprendizaje con diferencias temporales estiman valores basados en otras estimaciones. Esta técnica se llama *bootstrapping*. Cada paso en el entorno genera un *ejemplo de aprendizaje*, que se utiliza para agregar valor acorde a la recompensa inmediata y el valor estimado del siguiente estado o pareja acción-estado. El primer algoritmo propuesto por Sutton y Barto (1998) es el TD(0). Este algoritmo estima V^{π} para una política π, en línea (observando directamente el entorno) y de forma incremental. TD(0) puede ser usado para evaluar una política y trabaja utilizando esta regla de actualización:

$$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right] \qquad \text{(VII..6)}$$

donde $\alpha \in [0,1]$ es la taza de aprendizaje que determina que tanto se actualizan los valores. Este *backup* se realiza después de experimentar la transición del estado s al s' basado en la acción a. Solo el valor de un estado sucesor se utiliza. El valor de α debe reducirse apropiadamente para que el aprendizaje converja. Los siguientes dos algoritmos aprenden funciones-Q directamente de muestras, eliminando la necesidad de utilizar un modelo de transición para la selección de una acción.

3) Q-Learning Este algoritmo es de los más utilizados en aprendizaje por refuerzo debido a su fácil implementación. Es un método *off-policy*. Esto significa que busca optimizar valores-Q en lugar de parejas valors-estado, y además determinar la política óptima del MDP. Parecido a TD, en cada iteración solo se conoce dos estados, s y su sucesor. El algoritmo consiste en ir estimando valores-Q para las acciones, basado en las recompensas y la función-Q del agente. La regla de actualización es una variación de TD, usando valores-Q y una función *max* sobre los valores-Q del próximo estado para

Algorithm 1 Aprendizaje TD

1: Inicializar $V(s)$ con valores arbitrarios
2: Escoger una política π a evaluar
3: Definir estado inicial s
4: **loop**
5: $a \leftarrow$ resultado probabilístico de $\pi(s)$
6: Realizar acción a, obtener recompensa r y cambiar al estado s'
7: $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
8: $s \leftarrow s'$

actualizar Q_t hacia Q_{t+1}:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)\right) \quad \text{(VII..7)}$$

Q-learning es especializa en la exploración. Significa que convergerá a la política óptima sin importar la política de exploración que se elija, bajo el supuesto que cada pareja acción-estado sea visitado infinitas veces, y la tasa α haya disminuido correctamente. (watkins)

Algorithm 2 Q-learning

1: Inicializar $Q(s,a)$ con valores arbitrarios
2: Definir estado inicial s
3: **loop**
4: $a \leftarrow$ resultado probabilístico basado en la política de comportamiento $\pi(s)$ y $Q(s,a)$
5: Realizar acción a, obtener recompensa r y cambiar al estado s'
6: $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma max_{a'} Q(s',a') - Q(s,a)\right)$
7: $s \leftarrow s'$
8: $\forall s \in S, \pi(s) \leftarrow argmax_a Q(s,a)$

4) SARSA. A diferencia de Q-learning, SARSA es un algoritmo *on-policy*. Esto significa que aprende la función-Q para la política que el agente está ejecutando actualmente. El nombre viene de las siglas de la secuencia de palabras *State-Action-Reward-State-Action*. Utiliza la siguiente regla de actualización:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right) \quad \text{(VII..8)}$$

donde la acción a_{t+1} es la acción a ser ejecutada por la política actual para el estado s_{t+1}. La operación *max* es sustituída por la estimación del valor de la siguiente acción de acuerdo a la política. Este algoritmo también convergerá en el límite hacia la función de valor y política óptima bajo la condición que todos los estados y acciones han sido exploradas infinitas veces y que la política converja en el límite a una política codiciosa, es decir, ya no se continua explorando. SARSA es útil en entorno no estacionarios dado que nunca se podrá alcanzar una política óptima.

Algorithm 3 SARSA

1: Inicializar $Q(s,a)$ con valores arbitrarios
2: Definir estado inicial s
3: $a \leftarrow$ resultado probabilístico basado en la política derivada de $Q(s,a)$
4: **loop**
5: Realizar acción a, obtener recompensa r y cambiar al estado s'
6: $a' \leftarrow$ resultado probabilístico basado en la política derivada de $Q(s,a)$
7: $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$
8: $s \leftarrow s'$ y $a \leftarrow a'$

b Explotación versus exploración. Los algoritmos de aprendizaje están diseñados para converger a la política óptima, con el supuesto que todos los posibles estados y acciones hayan sido exploradas infinitas veces y evitar caer en la convergencia de políticas sub-óptimas. Los métodos off-policy pueden lograrlo dado que todas las acciones pueden ser ejecutadas mas de alguna vez dado que no hay acciones con cero probabilidad de ser elegida. Mientras que los métodos on-policy requieren que la política converja a uno codicioso, mientras se está explorando. El problema de estos métodos radica en la decisión si se continúa explorando o solamente utilizar la información ya adquirida. Existén tipos de políticas que tratan de buscar un balance entre explorar nuevas experiencias y explotar las mejores acciones con la información que se ha adquirido hasta el momento.

ε-greedy Este tipo de políticas es una mezcla de una política codiciosa(para explotar) y una uniforme(para explorar). El parámetro ε puede ser arbitrariamente pequeño y conducir a la convergencia hacia una política codiciosa. Se debe realizar una acción a definida por la política, pero hay una probabilidad ε de elegir otra acción.

Softmax Las políticas ε-greedy son las más comunes. Sin embargo, traen una desventaja. Estas políticas puede dar pesos iguales a las acciones no codiciosas, de las cuales, varios podrían desempeñar mejor que otras, a pesar de no ser codiciosas. Para evitar esto, se utiliza una función de utilidad sobre estas acciones para poderlas distinguir mejor. Este método llamado *softmax* todavía asigna la mayor probabilidad a la acción codiciosa pero no trata a las demás de la misma forma. El método más utilizado se basa en una distribución de Boltzmann(Ecuación y controla el enfoque de exploración a través de un parámetro $\tau > 0$.

$$\pi(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}} \tag{VII..9}$$

Al igual que en políticas codiciosas, cuando τ se aproxime a cero la política se vuelve codiciosa haciendo este método adecuado para usarlo en algoritmo on-policy. (Marsland , 2014)

c Funciones de valor aproximadas. La función de valor es utilizado para planificar o aprender en MDPs. Sin embargo, si el espacio de posibles estados en muy, tabular la representación de una

función de valor será complicado. El consumo de memoria sería altísimo y para aprender de forma óptima requeriría grandes cantidad de computaciones y datos. Este problema se denomina como la *maldición de la dimensión*. El crecimiento exponencial de un estado o conjunto de acciones como una función de la dimensión del estado o acción. Para resolver problemas del mundo real que son estados muy grandes, ha llevado a la creación de técnicas de aproximación. Estos métodos usan una función de aproximación para aproximar funciones de valor por lo que sacrifican exactitud para poder representar los estados de forma más manejable.

Sea $\hat{V}^{\pi}(s;w)$ la aproximación de la función valor-estado $V^{\pi}(s)$ representado por una arquitectura de aproximación paramétrica con parámetros libres w. Los parámetros w son ajustados para que los valores aproximados sean muy parecidos a los valores originales:

$$\hat{V}^{\pi}(s;w) \approx V^{\pi}(s) \tag{VII..10}$$

Por lo tanto, $\hat{V}^{\pi}$ puede ser usado en lugar de la función de valor exacta $V^{\pi}(s)$. Igualemente, $\hat{Q}^{\pi}(s,a;w)$ sería la aproximación de la función de valor acción-estado $Q^{\pi}(s,a)$. Se ajustan los parámetros w:

$$\hat{Q}^{\pi}(s,a;w) \approx Q^{\pi}(s,a), \tag{VII..11}$$

por lo que $\hat{Q}^{\pi}$ puede se usado en lugar de la función de valor exacta Q^{π}.

En general, para la mayor[ia de arquitecturas de aproximación, las necesidades de almacenamiento son independientes del tamaño del espacio de estados y/o acciones. Además, la mayoría de arquitecturas de aproximación no tienen restricciones sobre un espacio de estados de cantidad finita. Por lo tanto, se debe realizar una buena generalizaicón dado que la función de aproximación de valor proveerá valores correctos sobre el todo del estado/espacio estado-acción, usando soloamente información limitada de un subconjunto del espacio.

Una categorización amplia para de arquitecturas para la función de aproximación de valor son las arquitecturas lineales y no lineales. Esto depende de la forma que los parámetros entran a la arquitectura y no a la habilidad de aproximación de una arquitectura. Las no lineales son más expresivas debido a las interacciones complejas entre sus parámetros libres, aunque para refinar los parámetros en muy elaborado y complejo. Las lineales son las más populares. Esta arquitectura aproxima la función $Q^{\pi}(s,a)$ o $V^{\pi}(s)$ como

una combinación lineal de pesos de diferente características:

$$\hat{V}^{\pi}(s;w) = \sum_{j=1}^{k} \phi_j(s) w_j = \phi(s)^{\mathrm{T}} w \tag{VII..12}$$

$$\hat{Q}^{\pi}(s,a;w) = \sum_{j=1}^{k} \phi_j(s,a) w_j = \phi(s,a)^{\mathrm{T}} w \tag{VII..13}$$

Los parámetros libres de la arquitectura son los coeficientes w_j de la combinación (denominados como *pesos*).

Lagoudakis (2011) presenta diferentes arquitecturas de aproximación lineal:

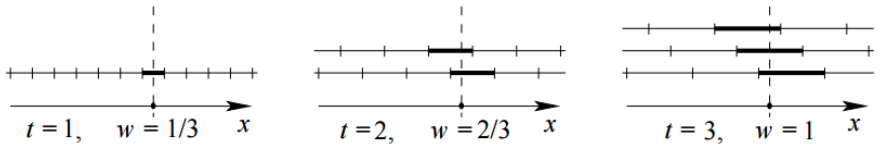
- **Look-up table**: es la representación tabular tradicional, por lo que no hay aproximación. Cada función base es una función de indicación cuyo valor es 1 solo para un punto de entrada discreto específico (estado o estado-acción), y 0 en caso contrario. Cada parámetro almacena un valor o entrada en la tabla.
- **Discretización**: técnica que consiste en convertir un espacio continuo a uno discreto usando una cuadrícula uniforme o de resolución variable. Una función base indicador es asignado a cada celda de la discretización y el parámetro correspondiente guarda el valor de esa celda
- **Tile coding (CMAC)**: este esquema utiliza diferentes discretizaciones (tilings) para mejorar exactitud. Genera funciones base indicadoras para cada celda de cada tiling y concatena las funciones bases de todas las tilings. Cada parámetro corresponde a una celda en un tiling, pero el valor de cada punto de entrada es computada aditivamente de los valores de todas las celdas de todas las tilings.
- **Agregación de estados**: es una familia de esquemas donde estados similares (en base a cierta métrica) son agrupados y son tratados como un solo estado. La métrica de semejanza es formado a través de técnicas de reducción de dimensión para identificar las dimensiones más significativas en un espacio multi-dimensional, a diferentecia de métodos de proximidad convencional en el mismo espacio. Hay una función base indicadora para cada grupo y un solo valor para todos los estados de ese grupo.
- **Polinomial**: esquema de aproximación genérica adecuado para problemas con varias variables continuas del estado. Cada función base es un término polinomial compuesto de variables de estado hasta cierto grado.
- **Funciones base radiales (RBF)**: esquema de aproximación para variables continuas. Cada función base es un gaussiano con una media y varianza fija. Los gaussianos son topologías organizadas de tal forma que pueden cubrir el espacio de entrada con algún traslapo.

Las arquitecturas lineales ofrecen varias ventajas: son fáciles de implementar y utilizar, y su comportamiento y transparente, tanto desde la perspectiva del análisis como la de depuración (debugging) e ingeniería.

(Lagoudakis , 2011)

1) Tile Coding. Un método común para encontrar características para en una función lineal, el aproximador divide el conjunto de estados continuos en segmentos separados y une una característica hacia cada segmento. Una característica está activa(valor igual a 1) si el estado relevante cae entre el segmento correspondiente. De lo contrario, está inactiva(igual a 0). Este método de discretización es utilizado para la arquitectura de tile coding. Se basa en la estructura CMAC propuesta por (albus). El conjunto de estados se divide en un número de subconjuntos disjuntos. Estos subconjuntos son llamados *tiles* o celdas. La idea detrás de tile coding es utilizar múltiples tilings superpuestas y para cada tiling, mantener los pesos de las celdas. El valor aproximado en un punto es la suma de pesos de las celdas, una por tiling, que la contienen. Conforme va entrenando, el método ajusta los pesos de las celdas involucradas por la misma cantidad para reducir el error. **??** hay un ejemplo de tile coding, de una dimensión. El conjunto de estados consiste solamente de la variable continua x. Las celdas son del mismo tamaño y las tilings adyacentes tienen un *offset* por la misma cantidad, este tipo de organización de tilings se llama *canónico*. En la figura **??** también se muestra la computación de valores estimados. Una organización de tiling como los de esa figura se compone de un valor dado w que es el tamaño de la celda y el número de tilings t. La proporción de w/t es la *resolución* de una tiling. Entonces, se refiere al número de tilings como *amplitud de generalización* debido a que las organizaciones de tilings con más tilings logran generalizar más ampliamente. Una forma degenerada de tile coding es utilizar discretización directa, es decir, utilizar solo 1 tiling y no generalizan más allá de las fronteras de las celda. (Sutton y Barto , 1998)

Figura 34. Utilizando una resolución $r = 1/3$, se muestan diferentes organizaciones canónicas de una, dos y tres tilings, y cada uno con su respectiva longitud de celdas w. Las líneas resaltadas representan las celdas(tiles) que discretizan el estado. El valor estimado es la suma de los pesos de estas celdas.



(Sherstov y Stone , 2005)

G METODOLOGÍA

En este módulo se presentará una propuesta de un sistema de asistencia de tránsito que busca ser flexible y escalable, de modo que pueda ser utilizada para futuras investigaciones o para una implementación real

en una zona urbana. Se diseñará un sistema multiagente y se implementará sobre un simulador basado en Traffic Cellular Automata para realizar las pruebas necesarias y evaluar eficiencia de la solución. Los agentes de este sistema se programarán con un algoritmo de aprendizaje por refuerzo y se especificará cómo el agente explora las diferentes opciones para optimizar el flujo de tránsito.

1 Diseño de sistema multiagente. El tránsito en la ciudad es un entorno donde hay poca accesibilidad de toda la red calles, no es determinístico y es dinámico. Para proponer una solución que mejore el tránsito se necesita que maneje varios criterios: cambiar dinámicamente los patrones de tránsito, ocurrencia de eventos impredecibles y un ambiente de tránsito no finito. Dado que un agente es incapaz de percibir todo el ambiente por completo, se busca que hayan más de un agente y que entre todos logren buscar un consenso para mejorar el flujo vehicular.

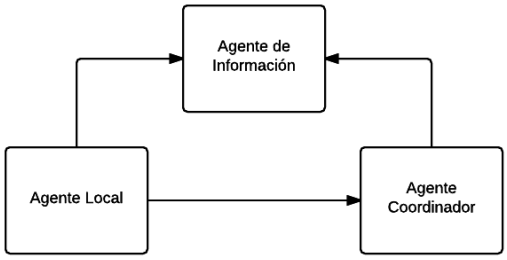
Para encontrar un balance entre las necesidades de una intersección y las necesidades a nivel global, se sugiere que haya una estructura que sea capaz de optimizar cada escenario local, en este caso las intersecciones, y un ente superior que monitoree los agentes locales y pueda modificar decisiones en base a necesidades globales. Entre los grandes desafíos de la congestión vehicular están los cuellos de botella de la red, por lo que un objetivo a mejorar es reducir estos atascos internos. Utilizando sensores le dan capacidad a los agentes conocimiento del ambiente inmediato y le proporciona mayor flexibilidad para que pueda cambiar sus patrones de tiempo. Pero al ser incapaz de percibir las otras intersecciones, puede dar lugar a que varios agentes permitan un flujo perjudicial hacia otra intersección y generar un cuello de botella.

Un sistema multiagente jerárquico permite atacar tres situaciones:

- Puede estar ejecutándose en tiempo indefinido al no existir un proceso de finalización, lo cual es pertinente para una red de tránsito.
- Puede manejar eventos inesperados de forma dinámica y sin necesidad de programar un plan preexistente
- Puede conseguir un balance entre el nivel global y los niveles locales.

La jerarquía que se propone está basado en el trabajo de France y Ghorbani (2003), donde se definen tres roles para mantener control de los diferentes espacios de la ciudad. En la Figura 35 se resume la interacción de estos tres roles. El agente local (AL) informa sobre los cambios en los patrones de los semáforos al agente coordinador (AC). Este agente se dedica a monitorear posibles cuellos de botella y congestiones de tránsito en las intersecciones, así como cualquier evento que dificulte el flujo vehicular. Por último, el agente de información(AI) no participa en el ambiente pero recibe información de los otros dos agentes y apoya con información acerca de todo el ambiente.

Figura 35. Sistema multiagente jerárquico propuesto por France y Ghorbani (2003)



a Agente de información de tránsito. Solamente se tendrá un agente de información que estará en constante comunicación con los demás agentes. Su principal función es guardar información de las calles. Almacenará la dirección de las calles, la interconexión entre intersecciones, densidad de tránsito por intersección y el tiempo que falta para que el semáforo cambie de luz. La razón de tener un agente aislado es para centralizar toda esta información sin que haya inconsistencias y proveer al agente coordinador con información para cada intersección. Los sensores están conectados al agente para poder recopilar información presente e informar a los agentes locales de su respectiva intersección.

b Agente coordinador de tránsito. Dado que es difícil que todos los agentes se comuniquen entre ellos para determinar los mejores patrónes de semáforos, se necesita de un agente que sea el mensajero para gestionar los diferentes agentes. Por lo que si hay intersecciones con alta congestión, el coordinador se encarga de que esta intersección reciba menor carga adicional y que no colapse las calles. Establece un vínculo con cada agente de cada intersección y necesita saber que intersecciones se interconectan, que puede obtener del agente de información

c Agente local de tránsito. Los agentes locales son los cerebros del sistema y se encargan de estar calculando un patrón de semáforos óptimo y disminuir la cantidad de carros parados en su respectiva intersección. Este agente es especializado en recibir información de la intersección y poder proponer acciones que puedan mejorar el flujo vehicular. Cada vez que se quiera cambiar de luces, primero informa al coordinador para que tenga autorización en efecto pueda cambiar de vía para no perjudicar otros agentes. Son estos los que se equipan con un algoritmo por refuerzo para que vayan aprendiendo en tiempo real la mejor combinación de patrones para optimizar el flujo de vehículos.

d Intercomunicación de agentes. Los agentes locales necesitan estar en constante comunicación con el agente coordinador para recibir autorización e información adicional sobre sus vecinos y así evitar que todo el sistema se vea afectado. El agente local envía un mensaje al coordinador conteniendo el tiempo calculado para su semáforo. Luego el coordinador verifica con los tiempos de sus vecinos para determinar cual será el ótptimo global para este AL. El coordinador también consulta con el agente de

información para saber cuáles son los vecinos de cada intersección. Con esta información el coordinador determina si autoriza los patrónes de luces del agente local o si penaliza para no perjudicar el óptimo global. La ventaja de esta metodología es que en caso de algún accidente o congestión, solamente se informa a los agentes involucrados y no a todos en el sistema.

2 Aprendizaje de los agentes locales. Los agentes que estarán en cada intersección deberán aprender los diferentes tiempos y patrones de semáforo basado en la información presente de cantidad y velocidad promedio de carros en los carriles. El periodo de aprendizaje. Para este sistema, se utilizará el algoritmo de aprendizaje por refuerzo SARSA, que permite buscar el mejor comportamiento y reduzca la cantidad de carros en espera. Las razones por la que se eligió este algoritmo son las siguientes:

- Al modelar el comportamiento del tránsito como un modelo de Markov, no se tiene las probabilidades de cambios de estados dado que estamos trabajando en un entorno estocástico. Al ser SARSA un algoritmo independiente de modelos, puede implementarse en el sistema.
- No se necesita calcular los resultados esperados para la transición de un estado al siguiente. En SARSA solo se necesita consultar el valor Q de los dos estados involucrados en la función de transición. También solo se necesita guardar información de los estados explorados y hasta que se exploren nuevos estados se crean espacios en la memoria para guardar. Esto beneficia para estados que necesitan más características para ser representados y la cantidad de memoria para almacenar aumenta exponencialmente. Por lo que la actualización también es mucho más rápido y el peso computacional y la cantidad de memoria utilizada es mucho menor.
- Utilizando una política ε-greedy permite al agente explorar el entorno sin necesidad de conocer previamente los diferentes estados. También se busca que una solución que permita tanto explorar como encontrar la mejor política a cumplir. SARSA es un algoritmo ideal que puede encontrar una política óptima y segura que evite grandes penalizaciones (Sutton y Barto , 1998)

a Estados del entorno. El entorno en la ciudad cuenta con infinitas posibilidades, desde la cantidad de vehículos que llegan a la intersección hasta otros factores impredecibles como accidentes o eventos que ocasionan mayor congestión. Para que el agente logre aprender debe generalizar diferentes situaciones que compartan características similares a partir de experiencias anteriores. Se describen los estados como un vector de características. En base a los trabajos de Pham *et. al.* (2013) y de Balaji *et. al.* (2010), se especificaron las siguientes características que definirán el conjunto de diferentes estados:

- Tiempo (en segundos) del último cambio de luces del semáforo. Permite que el agente pueda aprender las consecuencias usando tiempos fijos.

- Tiempo (en segundos) del penúltimo cambio de luces del semáforo. El agente puede aprender patrónes de cantidad de vehículos que fluyen en cada calle y determinar en qué calle transitan más vehículos.
- Proporción de la velocidad promedio entre la calle que tiene luz verde y la calle que está detenida. Esta proporción permite al agente determinar los cambios en la cantidad de carros por calle y se adapte para poder distribuir la carga vehicular entre todas las calles. En la ecuación VII..14 se detalla cómo se calcula esta proporción. Se busca que la velocidad promedio de cada calle sea igual. El logaritmo se utiliza para mantener el valor de la proporción alrededor del cero y facilita la generalización para el entorno. Se puede dar el caso que el flujo vehicular para la calle con luz verde sea inferior a la de la calle con luz roja, por lo que se propone la ecuación VII..15, donde el resultado será el negativo de la proporción.

$$ratio_v = \log(\frac{v_{verde}}{v_{rojo}}) \tag{VII..14}$$

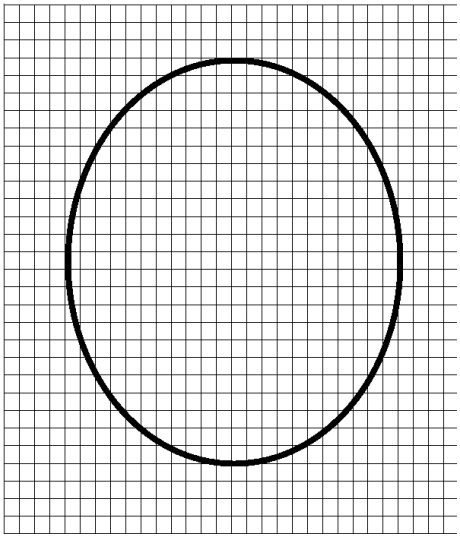
$$ratio_v = -\log(\frac{v_{rojo}}{v_{verde}}) \tag{VII..15}$$

b Acciones y penalizaciones para el agente local. El agente local tiene dos tipos de acciones: decidir continuar con la configuración de luces o cambiarla. En cada instante que el agente decida realizar una acción, el entorno se comportará de cierta manera y es cuando el agente necesita medir si la acción realizada ayudó a que estuviera más cerca de su objetivo. Para el sistema que se propone, el agente no buscará obtener la mayor recompensa sino reducir al máximo las penalizaciones que recibe por la cantidad de carros que están esperando a que cambie de vía. Estas penalizaciones son valores negativos que se calculan sumando el tiempo que ha pasado desde el último cambio más la cantidad de carros que están esperando, y negando ese resultado. Maximizar la recompensa equivale reducir los tiempos de espera y mejorar la velocidad promedio de la calle. También es importante notar que la recompensa que se obtiene es independiente de las decisiones del CTA.

c Discretización de estados. El tránsito de la ciudad es un entorno continuo por lo que hay una inmensa cantidad de estados posibles. La necesidad de memoria y poder computacional para operar sobre estas cantidad sería una desventaja para el sistema y el tiempo de respuesta sería muy largo. Para afrontar este problema se utiliza la técnica de discretización.

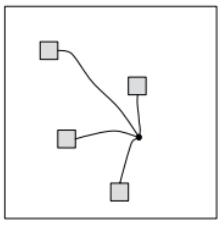
Dado que se trabajará con tres variables, se necesita discretizar en tres dimensiones. En la Figura 36 se muestra gráficamente el concepto de dividir el espacio de estados en una cuadrícula estados. Todos los estados deben corresponder a una celda. En este proyecto, las variables abarcarán 30 diferentes celdas. Como son tres variables, se estará trabajando con un total de $30 \times 30 \times 30$ estados discretos. Cada agente reservará esta cantidad de estados en memoria. Esta cantidad de estados puede reducirse dado que no se

Figura 36. El espacio de estados (rodeado por la elipse) es discretizada por medio de celdas. Para el escenario del tránsito urbano, se trabajará con 30 celdas para cada variable.



utilizarán todos. En la Figura 36 la elipse dibujada representa los estados que seguramente los agentes visitarán varias veces. Los estados que están afuera son poco probables por lo que no se debería reservar espacio de memoria para estos estados. Se utilizó *hashing* para asociar las tres variables a un estado discreto. Solamente se guardarán los estados que han sido explorados al menos una vez. Por lo tanto, se ahorrará espacio de memoria y evita que se pierda rendimiento.

Figura 37. *Hashing* permite asociar diferentes valores de cada estado continuo hacia un estado discreto en común.



(Sutton y Barto , 1998)

H RESULTADOS

El diseño propuesto fue implementado en el lenguaje de programación Python. Cada agente fue implementado como una clase y el algoritmo de aprendizaje SARSA también. El orquestador de este sistema es el agente de información ITA. Los demás agentes guardan la referencia de este para poder comunicarse y obtener la información que se necesita.

ITA se encarga de obtener la información del simulador a través de los métodos que provee la clase

`tca_service`, resultado del *Módulo de Integración de Algoritmos con Traffic Cellular Automaton, Evaluación y Análisis de Resultados de Inteligencia Artificial*. La clase LTA es el agente que estará calculando los cambios de luces en las intersecciones. Hay un agente LTA por cada intersección que haya en el mapa del simulador. El agente local utiliza la clase SARSA como su algoritmo de aprendizaje. Al instanciar el algoritmo, de una vez se asigna los valores α, γ y ε. Por último, está el agente CTA. El agente se encarga de recibir peticiones de cambio de luces de cada agente LTA. El CTA a través de la función , controla que las calles de salida de la intersección no estén saturadas e impedir que continúen entrando carros. La densidad de tráfico en las calles se calculan de la siguiente forma:

$$d = \frac{n_{vehicles}}{avg_{speed} + 0{,}1} \tag{VII..16}$$

La cantidad de vehículos no debe exceder a la capacidad que tiene la calle, y aunque haya un alto número de vehículos, debe compararse con la velocidad promedio de la calle y así determinar que el tráfico está fluyendo. Por lo tanto, la densidad es la tasa de cantidad de vehículos y la velocidad promedio. Se suma 0,1 como penalización en caso la velocidad promedio sea 0, es decir, está completamente varado.

1 Agente de información ITA. La clase ITA se encarga de obtener información del simulador de TCA y guarda las variables que utilizarán los agentes de las intersecciones. A través de la función `dynamic_time_update`, el agente se encarga de construir un diccionario con la información de cada calle. El siguiente código ejemplifica el resultado:

```
dynamic={
  id:{'cars_number': 9, 'avg_speed': 2.5, 'green_light': 0},
  id:{'cars_number': 11, 'avg_speed': 3.1, 'green_light': 1},
  id:{'cars_number': 9, 'avg_speed': 2.3, 'green_light': None}
}
```

El valor de `green_light` determina si la calle actualmente tiene luz verde. Si no hay un semáforo en al final de la calle el valor será `None`. También construye un diccionario de intersecciones donde la información que guarda es distinta y desde la perspectiva de un agente local:

```
intersections={
  {'neighbors': [1,3], 'traffic_light': 0, 'in_streets': [0, 1],
      'out_streets': [2, 3], 'green_light_id':0,lights:[0,1]},
  {'neighbors': [0,2], 'traffic_light': 1, 'in_streets': [3, 4],
      'out_streets': [5, 6],'green_light_id':2,lights:[2,3]},
  {'neighbors': [3,1], 'traffic_light': 2, 'in_streets': [6, 2],
      'out_streets': [0, 4],'green_light_id':4,lights:[4,5]}
```

```
}
```

Los diccionarios de cada intersección muestran los `id` de las intersecciones vecinas, el semáforo en la intersección y que luces maneja y las calles que entran y salen de la intersección. El valor de `green_light_id` es el identificador de la luz que está actualmente en verde y es necesario guardarlo para el momento que se desea realizar cambio de luz se tenga referencia qué luz hay que cambiar a rojo.

Por último, la clase ITA tiene la función `updateSchedule` donde realiza los cambios de luces de todas las intersecciones que decidieron realizar esta acción. Para llevar registro de todas las luces que hay que cambiar se tiene el diccionario `newSchedule`:

```
newSchedule=[
  {'id': 0, 'schedule': {0: 0}},
  {'id': 1, 'schedule': {0: 2}},
  {'id': 2, 'schedule': {0: 4}}
]
```

El diccionario `schedule` indica que al inicio del nuevo ciclo debe poner en verde la luz con el identificador de luz especificado. El `id` se refiere al número de intersección.

2 Algoritmo de aprendizaje SARSA. Dado que no se cuenta con datos históricos, las pruebas de la metodología propuesta serán realizadas sobre un simulador de Traffic Cellular Automata y evaluar el desempeño del algoritmo. El agente recibe información del simulador y con base en estas entradas toma una decisión y realiza una acción. El algoritmo SARSA guarda una tabla de valores-Q para cada estado que ha visitado.

```
qmatrix={
  'change': {
    '3,2,10': -6,
    '2,1,0': -5,
  },
  'stay': {
    '1,0,0': -10,
    '2,2,20': -16,
    '1,1,20': -10,
    '1,2,20': -12
  }
}
```

El diccionario mapea una acción hacia una recompensa de haber realizado dicha acción en ese estado. Dado que los estados son continuos, el proceso de discretización se hace a través de la función `activeTile`. Los parámetros que se utilizaron para discretizar los estados son los siguientes:

```
specs=[
  {"negative":False,"start":0,"width":5.0,"size":30},
  {"negative":False,"start":0,"width":10.0,"size":30},
  {"negative":True,"start":-5,"width":0.5,"size":30}
]
```

El valor `negative` indica si la característica de este estado puede tomar un valor negativo. Las primeras dos variables son los tiempos acumulados desde las últimas acciones del agente. Por lo tanto, se tiene como `False` la propiedad de ser negativo. La proporción de velocidad promedio entre las dos calles de entrada, por otra parte, sí puede ser negativo. `start` indica el valor inicial de la primera celda, `width` es el tamaño de la celda, `size` es la cantidad de celdas.

3 Experimentación del sistema multiagente en el simulador. Se realizaron pruebas en el simulador para visualizar los resultados del algoritmo al llevarse a la práctica. La experimentación consiste en utilizar dos mapas distintos de intersecciones y ejecutar el algoritmo utilizando estos mapas. Cada intersección del mapa será un agente local del algoritmo. El plan para determinar el desempeño fue el siguiente:

- Los agentes estuvieron consultando información y realizando una acción cada cierto tiempo. Cada actualización del simulador representará una unidad de tiempo. La actualización del simulador consiste en calcular y mostrar las nuevas posiciones de los vehículos dependiendo de las reglas del TCA y la velocidad que llevaban.
- Se tomaron muestras del algoritmo ejecutandose para tres casos: cada 5 unidades de tiempo, cada 10 unidades de tiempo y cada 20 unidades de tiempo. Para cada caso se ejecutó utilizando los dos mapas previamente descritos. Cuando los agentes consultan el estado de las intersecciones al agente de información, este guarda las unidades de tiempo transcurridas desde el inicio de la simulación y la velocidad promedio del mapa completo en ese instante de tiempo.
- La muestras de velocidad promedio obtenidas son de las primeras 1000 iteraciones, es decir, se observaron los resultados por 1000 unidades de tiempo.
- Se realizaron 30 simulaciones para cada caso. Se calculó la mediana de las velocidades promedio para cada intervalo de tiempo establecido. La mediana fue la opción ideal, comparado con la media, debido a que no se comprobó que los resultados presentan una distribución normal.

Debido a que los resultados son mejores para el caso donde los agentes realizan una acción cada 5 unidades de tiempo, los siguientes resultados fueron obtenidos con la misma configuración, asimismo utilizando el mapa 8×8. Se realizó una comparación entre la velocidad promedio del mapa y la cantidad de carros. Además, se realizó una simulación sin el agente coordinador para compara resultados del desempeño del algoritmo.

I DISCUSIÓN

En aprendizaje por refuerzo, generalmente los agentes empiezan sin tener información previa del entorno. Al inicio estarán en constante exploración para conocer las reacciones del entorno ante las acciones que el agente realice. Para la implementación del algoritmo se determinó que utilizar información histórica del tránsito urbano puede llevar a mejores resultados al inicio del aprendizaje. Se puede ver en la Gráfica 41 que conforme van entrando los vehículos al mapa la velocidad va disminuyendo para las simulaciones en el mapa 8×8. En las gráfica 40a, se observa que la velocidad promedio se mantiene estable por el lapso de tiempo que se estuvo simulando. Para el mapa 4×4 las congestiones son más manejables debido a que solo hay cuatro intersecciones y las calles son más largas y de un carril. La frecuencia que los agentes realizaban una acción no dio resultados contrastantes. Las unidades de tiempo que transcurren para realizar una acción afecta a la calle con luz roja y termina congestionandose. La penalización aumenta con el tiempo que permanece en rojo el semáforo. El agente decide si cambiar de luces o continuar con la misma configuración para determinar si es beneficiado con el descongestionamiento de la otra calle.

Comparándolo con la Gráfica 40b, la cantidad de intersecciones es mucho mayor y las calles se congestionan con mayor rapidez. Debido al rápido incremento de vehículos en el mapa, mientras más se tarda el agente en reaccionar, más rápido se congestionará el mapa y la velocidad promedio se reducirá. En la gráfica se puede ver que para el caso de actualizaciones del agente cada 20 unidades de tiempo termina con cuellos de botella mucho más rápido y la velocidad promedio llega a 0. Para el caso de actualizaciones del agente cada 10 unidades de tiempo logra mantener una velocidad promedio superior por más tiempo que la frecuencia de actualización de 20 unidades de tiempo. Sin embargo, después de varias actualizaciones, el mapa igual termina con un cuello de botella y la velocidad promedio baja a 0. Por último está el caso de actualizaciones del agente cada 5 unidades de tiempo. Los resultados fueron superiores a los dos anteriores, e incluso después de 2000 unidades de tiempo logra mantener una velocidad promedio superior a 1. Entre las muestras que se obtuvieron, la mediana resultó ser un valor cercano a 1.5. Por lo tanto, en distintas simulaciones la velocidad promedio podría ser mejor o peor tras 2000 unidades de tiempo de simulación. A pesar del mejor rendimiento, se puede ver la tendencia que la velocidad promedio continua cayendo debido a posibles cuellos de botella y el incremento de vehículos en el mapa.

La razón por la que se dan cuellos de botella se debe al comportamiento del simulador. Los vehículos

tienen cierta probabilidad de cambiar de calle (tienen una ruta establecida). En caso los carros de ambos carriles al frente de la calle deciden cambiar de calle en lugar de continuar en la misma dirección, tienen que validar que no hayan otros carros para no colisionar. Mientras no se libere la calle, los carros quedarán parados impidiendo que los carros traseros puedan continuar. Aunque el semáforo de luz verde, mientras estos carros no avanzen las calles se atascaran y formarán largas colas que incluso lleguen a obstruir otras intersecciones, impidiendo el flujo vehicular de esa intersección. Al haber cuello de botella, en pocas unidades de tiempo todo tránsito vehicular se paraliza y los agentes terminan realizando acciones esperando aprender pero sus acciones no afectarán el estado del entorno.

El propósito de implementar una jerarquía de agentes, donde hay un coordinador para dirigir a los agentes en las intersecciones, es para optimizar la velocidad promedio global. Se realizó una simulación sin el agente coordinador funcionando para comparar si hay alguna mejora significativa. En la Figura 42 se muestra la gráfica comparando la velocidad promedio, en una mapa 8×8, del sistema con un agente coordinador en funcionamiento contra un sistema con agentes locales independientes, es decir, sin un agente coordinador. Ambas simulaciones fueron ejecutadas con los agentes actualizando cada 5 unidades de tiempo. Se puede ver que en último caso, el cuello de botella se genera mucho más rápido que si hubiera un coordinador. El sistema con coordinador logra mantener una velocidad promedio entre 2.5 y 3 a partir de la unidad de tiempo 400. El sistema sin coordinador terminar decreciendo su desempeño y la velocidad promedio se reduce hasta llegar a 1000 unidades de tiempo donde llega a un cuello de botella y la velocidad llega a 0. Para el sistema con coordinador, todavía logra mantener el flujo vehicular hasta 1400 unidades de tiempo donde se reduce considerablemtente. Sin embargo, todavía hay vehículos que transitan y logran aumentar la velocidad promedio y que esté en valores cercanos a 0.5. La diferencia de tiempo que tarda un sistema con coordinador con uno sin coordinación es notable. Por lo que el sistema jerárquico implementado logra mantener la mayor cantidad de vehículos transitando por el tiempo que pueda.

J CONCLUSIONES

El sistema multiagente desarrollado muestra potencial para administrar mejor los semáforos en las intersecciones (France y Ghorbani , 2003). Sin embargo, las limitaciones del simulador también limitó obtener mayores resultados para realizar un mejor pronóstico de la eficiencia del algoritmo. El diseño jerárquico del sistema mejora notablemente el flujo vehicular, impidiendo que la velocidad promedio global llegue a 0 y se estanquen las calles el mayor tiempo posible.

Por otra parte, el algoritmo de aprendizaje por refuerzo implementado en los agentes locales también muestra tener efectos positivos en las intersecciones. Con los parámetros establecidos, se encontró que mientras más frecuente sea el análisis de las entradas del entorno, más rápido aprenderá y reaccionará de forma adecuada ante las congestiones de su intersección. Mientras menor sea el tiempo entre las diferentes

acciones ejecutadas, mayor será el flujo vehicular. La velocidad promedio se verá afectada principalmente por la cantidad de vehículos que estén transitando.

K RECOMENDACIONES

El agente coordinador debe comportarse de forma inteligente. Un agente que aprende debe poder reaccionar ante el entorno que está cambiando. Si el agente coordinado aprender por su lado ante el comportamiento de las intersecciones, puede que se logre una mejor coordinación de agentes en las intersecciones y mejorar la velocidad promedio de la zona urbana.

El diseño de los estados que fue propuesto está basado en otra literatura. Por lo tanto se recomienda buscar alternativas para representar los diferentes estados que puedan contribuir a la optimización del flujo vehicular en las intersecciones. Por ejemplo, el tiempo acumulado de cada vehículo. También se sugiere que se implementen nuevas técnicas de discretización de estados para determinar cómo la representación discreta del entorno afecta en los resultados.

Por último, un nuevo enfoque de sistema multiagente podría mostrar resultados interesantes. En este trabajo se definió el sistema multiagente como diferentes agentes compitiendo para optimizar su intersección e ignorar el de sus vecinos. Un sistema multiagente orientado a la división de tareas, aprendizaje en paralelo en lugar de independiente y cooperativo. La comparación de resultado podría traer mayor discución y nuevas propuestas para mejorar continuamente una solución viable para que pueda ser implementado en la realidad.

Figura 38. Diagrama UML de las clases implementadas

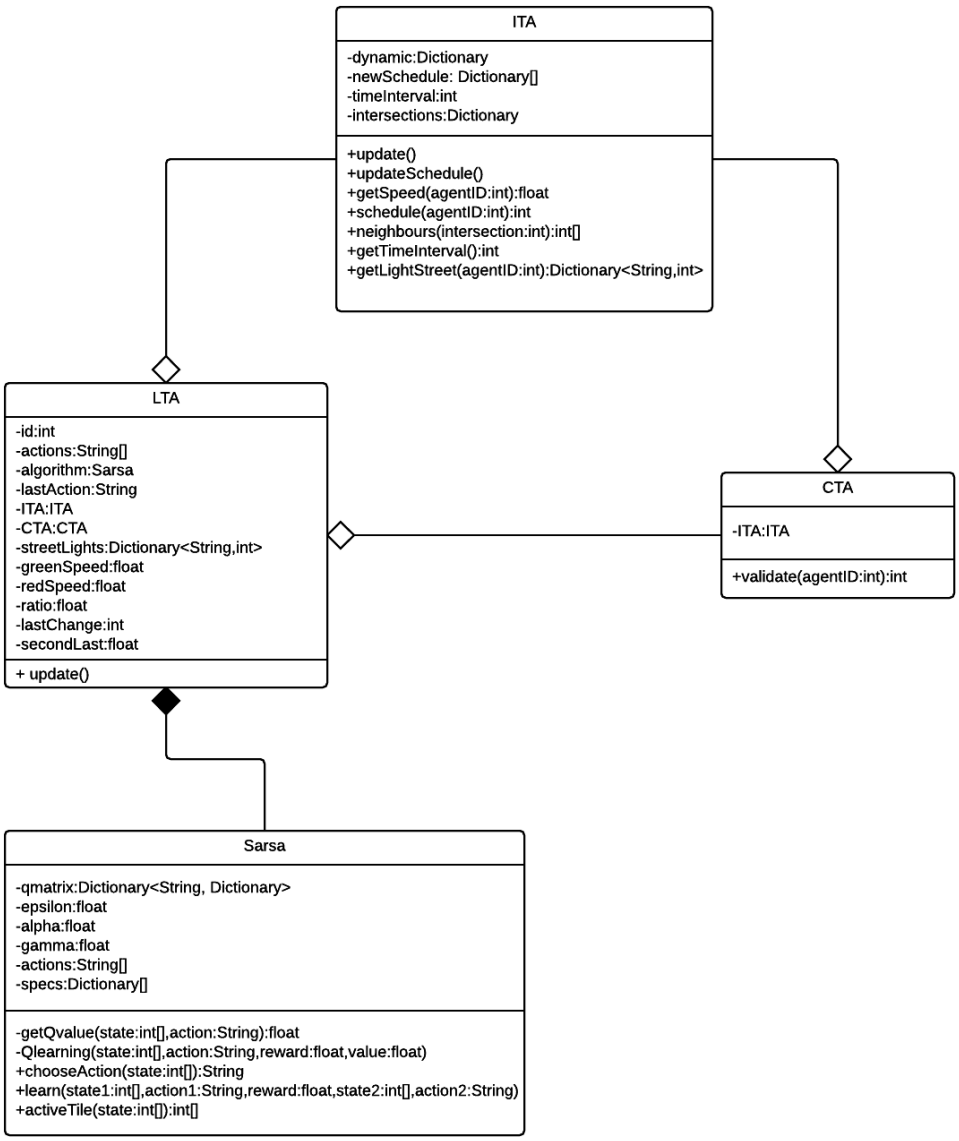
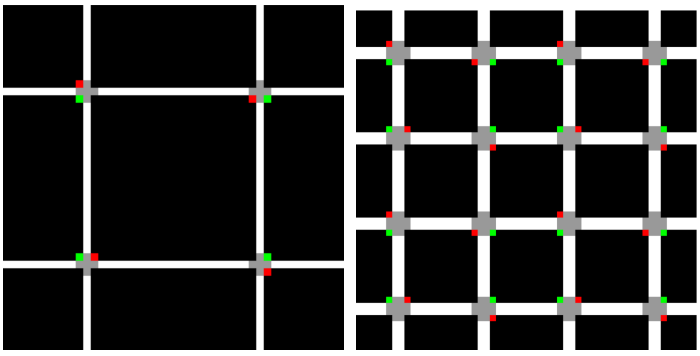
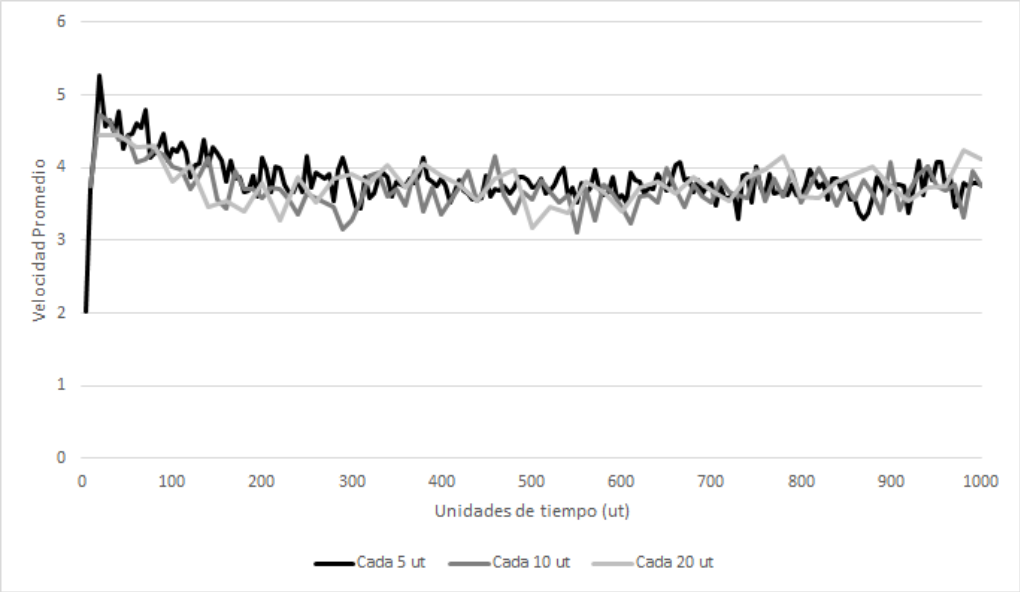


Figura 39. Mapas de intersecciones utilizados para la experimentación del sistema multiagente.

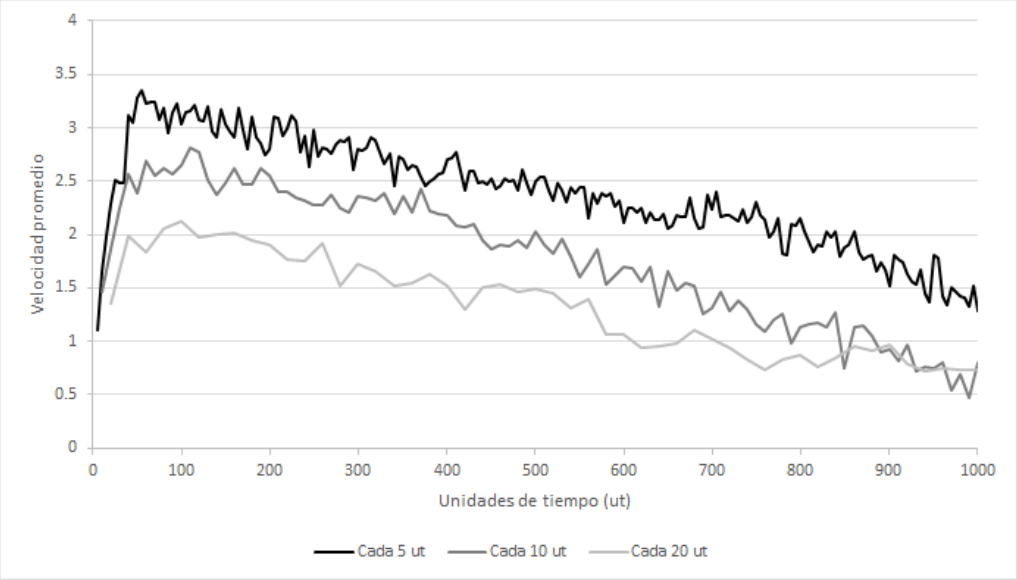


(a) Mapa 4×4 (b) Mapa 8×8

Figura 40. Velocidad promedio a través del tiempo de ejecución del sistema multiagente para cada mapa utilizado



(a) Mapa 4×4



(b) Mapa 8×8

Figura 41. Comparación entre velocidad promedio y cantidad de vehículos en el mapa durante una simulación

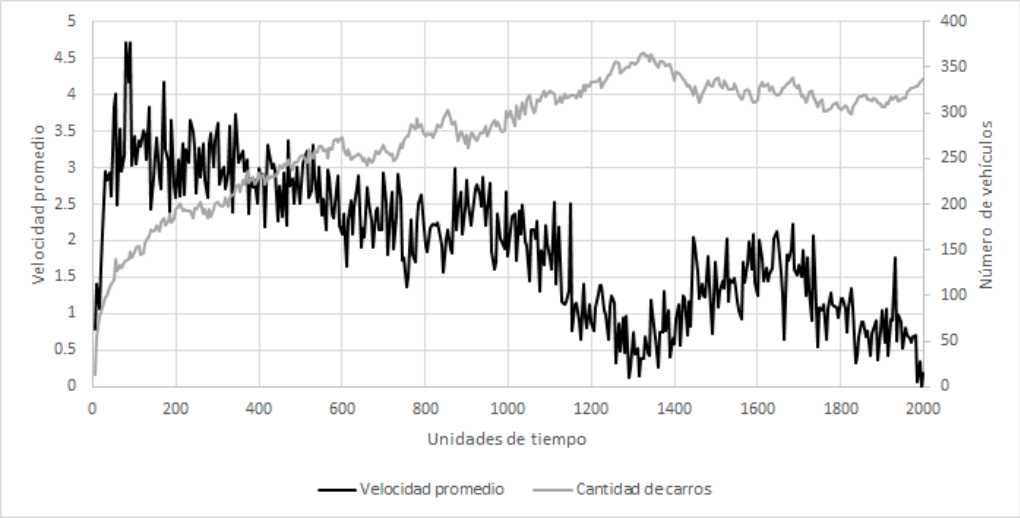
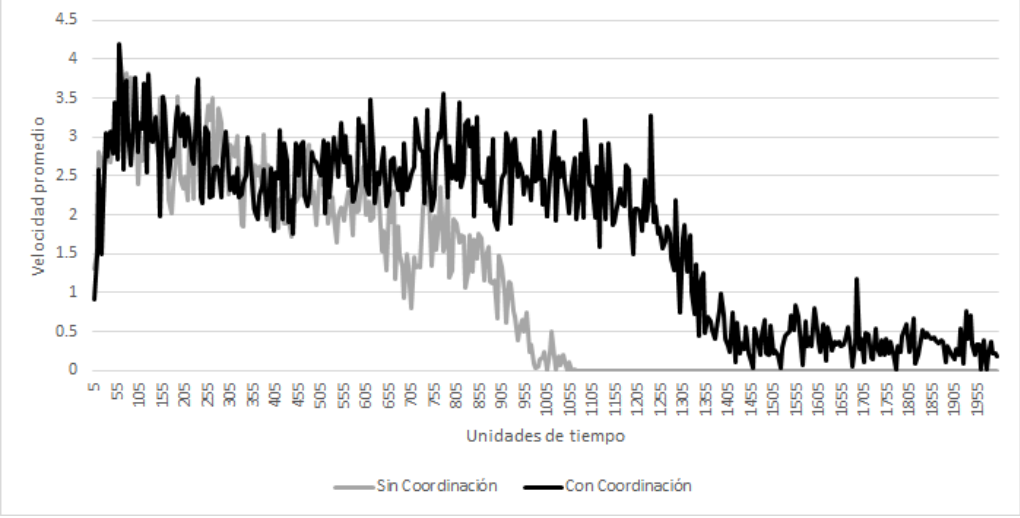


Figura 42. Comparación de velocidad promedio del mapa entre la presencia de un agente coordinador y sin coordinación



VIII. INTEGRACIÓN DE ALGORITMOS CON TRAFFIC CELLULAR AUTOMATON, EVALUACIÓN Y ANÁLISIS DE RESULTADOS DE INTELIGENCIA ARTIFICIAL

A INTRODUCCIÓN

El tránsito es un problema global, particularmente caótico en ciudades como Guatemala, en la que la infraestructura es insuficiente para el parque vehícular. Como un paliativo, la Municipalidad de Guatemala ha optado por conformar y fortalecer la Policía Municipal de Tránsito (PMT), incluyendo entre sus tareas el dirigir el tránsito en horas de particular congestionamiento. Mientras cumplen esta tarea, se apagan los semáforos y el flujo de vehículos queda en manos de los agentes, quienes deben hacer proezas para determinar cuándo dar vía a una calle u a otra, además de intentar coordinarse con agentes en otras intersecciones cercanas por radio. De ahí surge la motivación de este megaproyecto: brindar herramientas para empoderar a los agentes y ayudarles a tomar decisiones sobre el tránsito.

En este megaproyecto se busca proponer bases en dos frentes: hardware y software. En el primero, se presenta un prototipo de la infraestructura que tendría que adoptar la ciudad para contar con sensores de tránsito, dispositivos móviles para agentes de PMT y comunicación de estos con un sistema centralizado. En el segundo, se explora el diseño, implementación y comparación de dos algoritmos de inteligencia artificial para optimizar el tránsito; además, se propone un simulador basado en *Traffic Cellular Automata*, que será la herramienta utilizada para brindar información a los algoritmos de inteligencia artificial.

En el presente módulo se busca integrar los algoritmos de inteligencia artificial implementados al simulador de TCA para después poder realizar una comparación entre algoritmos y evaluar su rendimiento en el manejo del tránsito. También se busca emplear dos estrategias distintas de algoritmos aleatorios para poder contrastar los resultados obtenidos de los algoritmos de inteligencia artificial con un grupo control.

Para la integración se propone la utilización de un estilo de integración basado en invocación de procedimientos remotos permitiendo mayor flexibilidad y sin demasiada complejidad de implementación que termine acoplando las diferentes secciones sin posibilidad de realizar cambios sobre ellas. Para este ca-

so se desarrollaron diferentes métodos que permiten que los algoritmos de inteligencia artificial utilicen el simulador y métodos que brindan datos estadísiticos que calculan con información obtenida dentro del simulador.

Después de realizar diversas simulaciones para obtener datos del desempeño de los algoritmos se concluyó que el mejor algoritmo es de estrategia aleatoria con un enfoque de tiempos fijos. Sin embargo, no se recomendó la implementación del mismo debido al factor de aleatoridad, por lo que se sugiere utilizar el algoritmo genético cuyos resultados reflejan que este algoritmo obtuvo un desempeño cercano al mejor y cuyos resultados podrían mejorarse notablemente si su entrenamiento se realiza de forma más rigurosa en un equipo con mayor poder computacional.

B OBJETIVOS DEL MÓDULO

1 Objetivo general del módulo.

Integrar algoritmos con *traffic cellular automaton*, evaluar y analizar resultados de inteligencia artificial.

2 Objetivos específicos del módulo.

- Establecer un marco de trabajo con las reglas a seguir por cada módulo para lograr una integración rápida y sin errores.
- Realizar una planificación de integración de módulos.
- Implementar un algoritmo con estrategia de tiempos fijos para utilizar como grupo control.
- Implementar un algoritmo con estrategia de tiempos dinámicos para utilizar como grupo control.
- Establecer métricas para determinar la eficiencia de algoritmos en conjunto con los alcances de los algoritmos de inteligencia artificial.
- Diseñar los escenarios con el modelo de *traffic cellular automata* sobre los cuales se realizarán las simulaciones de los algoritmos de inteligencia artificial.
- Analizar los datos obtenidos en la simulación de cada algoritmo para determinar el rendimiento de los mismos.

C JUSTIFICACIÓN DEL MÓDULO

Las estrategias de desarrollo de software nos indican que cuando el software se desarrolla en diferentes módulos, es necesario contar con una estrategia para integrarlos. Por ello, el módulo del megaproyecto

plantea establecer un marco de trabajo conteniendo las reglas que se deben cumplir para una correcta integración. Respecto a la comparación de algoritmos, es necesario contar con un grupo de control para contar con mediciones que se obtienen sin aplicar inteligencia artificial a las decisiones, por lo que el módulo plantea implementar un tercer algoritmo basado en la estrategia de tiempos fijos y un cuarto algoritmo basado en la estrategia de tiempos dinámicos. Adicionalmente, al contar con cuatro diferentes algoritmos resulta necesario realizar diferentes análisis con variados enfoques para recopilar datos y en base a estos resultados determinar la eficiencia de cada uno de los algoritmos.

D ANTECEDENTES

El presente módulo busca integrar algoritmos genéticos con un simulador de *Traffic Cellular Automaton*, evaluar y analizar los resultados de dichos algoritmos en la optimización de ciclos de semáforo. A continuación se presentan algunos antecedentes sobre simuladores de tránsito y de el paradigma sobre el que se basa el simulador utilizado en este módulo: *Cellular Automata*. Por último se hace un recorrido por la evolución en la integración de software y diferentes publicacioes que se han enfocado en realizar mediciones sobre simuladores de tránsito.

1 Integración de software.

La integración de software se ha vuelto más compleja conforme pasan los años. Smith (2009) atribuye este comportamiento a que ahora existen más fuentes de donde el software obtiene información, existen múltiples arquitecturas y una mayor variedad en sistemas operativos. Debido a esto es que surgen diferentes estrategias que pueden adoptarse para realizar la integración de una forma más eficiente.

La integración también depende de las herramientas que se utilizan para desarrollo, incluyendo el lenguaje de programación. Es por ello que Sanner (1999) realizaron una investigación en donde lograron determinar que el uso de lenguajes interpretados, como *Python* es un componente clave para la integración exitosa. Esto debido a la habilidad de crear ambientes dinámicos con componentes no acoplados a bajo nivel. Es de suma importancia utilizar las herramientas existentes, patrones y estilos de integración para no caer en el *Integration Hell*, como lo denomina Duvall *et al.* (2007) ya que los desarrolladores pueden llegar a estancarse incluso semanas y perder segmentos de código.

2 Mediciones en simuladores de tránsito.

Desde el inicio de el uso de algoritmos cuyo objetivo es la optimización del tránsito en los simuladores, estos han ido de la mano de mediciones cuyo objetivo es calificar el rendimiento de los algoritmos. Existen varias publicaciones en donde se mencionan las estrategias que han sido utilizadas. Por ejemplo Brockfeld *et al.* (2008) utiliza tres diferentes enfoques con los que busca mejorar las condiciones generales del tránsito en el modelo planteado, las estrategias que utilizan son las siguientes: *Synchronized Traffic Lights*, estrategia basada en la sincronización de los semáforos en el modelo. *Green Wave Strategy*, esta estrategia está basada en flujos de cambios de semáforos

tomando como referencia un período establecido. Por último la estrategia *Random Offset Strategy* basada en la aleatorización. El simulador utilizado en este caso se basó en el modelo de *Nagel–Schreckenberg* y contó con 16 intersecciones.

Es posible observar otro ejemplo en la investigación realizada por Esser (1997) basada también en un modelo de *Cellular Automata* utilizando las reglas propuestas por *Nagel–Schreckenberg*. En este caso se consideró necesario incorporar tres instrumentos de medición al simulador. El primer instrumento cuenta con diferentes puntos en donde se obtiene el número de vehículos y la velocidad promedio de los mismos que pasan sobre estos puntos, separados por el tipo de vehículo. Cada vehículo también fue considerado un instrumento de medición teniendo la capacidad de brindar el tiempo de viaje individual o información específica de un punto a otro. Como último punto se consideró la red completa como instrumento obteniendo el número de vehículos total, velocidad promedio y uso de bordes.

Sánchez-Medina *et al.* (2004) también han realizado un estudio basdado en *Traffic Cellular Automaton* publicado en una serie de artículos científicos inlcuyendo una propuesta de arquitectura que utiliza algoritmos genéticos. Sánchez-Medina *et al.* (2005) contiene una demostración entre la estrecha correlación entre el uso de ambientes estocásticos y ambientes determinísticos. En la última publicación Sánchez-Medina *et al.* (2015) contiene un estudio de escalabilidad realizan pruebas a diferentes escenarios: *small, big, huge, immense*. También realizan un estudio de tiempo utilizando los enfoques de *FixSim* en donde la secuencia de cambios de luces del semáforo es fijada con anterioridad, *RanSim* que se basa en generar una secuencia aleatoria de cambios de luces y *LISim* que es un agente inteligente básico que cambia a verde la calle que tenga el mayor número de vehículos.

Algunas publicaciones también han buscado encontrar relaciones entre el simulador y elementos de la vida real como Maerivoet *et al.* (2005) y Gershenson (2009) que utilizan diferentes estrategias para determinar el ancho en metros de una celda basados en el largo promedio de un vehículo. Adicionalmente buscan encontrar la duración de una iteración dentro del simulador para poder realizar una relación con la cantidad de iteraciones que se deben realizar para representar una cantidad determinada de segundos en un semáforo.

E DELIMITACIÓN

El módulo de Integración de algoritmos con Traffic Cellular Automaton, evaluación y análisis de resultados de inteligencia artificial establecerá un marco de trabajo el cual deberán seguir los otros módulos para lograr una integración rápida y sin errores entre el modelo de Traffic Cellular Automaton realizado en otro módulo y los algoritmos de inteligencia artificial implementados en los otros dos módulos: un algoritmo basado en algoritmos genéticos y uno basado en sistemas multiagentes. Se implementará un algoritmo basado en tiempos fijos y uno basado en tiempos dinámicos que serán utilizados como grupo control al rea-

lizar las simulaciones con los algoritmos de inteligencia artificial. Por último se analizarán los resultados obtenidos y se realizará un análisis una comparación entre algoritmos evaluando el rendimiento de cada uno en diferentes situaciones.

F MARCO TEÓRICO

1 Patrones de integración.

Los patrones de integración surgen debido a la dificultad de integrar de forma sencilla diferentes aplicaciones. Como indica Hohpe *et al.* (2011), cualquiera que indique que la integración es sencilla debe ser increíblemente inteligente, increíblemente ignorante o tiene algún interés financiero en hacernos creer que la integración es sencilla. Sin embargo, existen personas que cuentan con la experiencia de resolver integraciones en numerosas ocasiones por lo que pueden comparar los nuevos problemas de integración con otros que ya han resuelto anteriormente asociando los *patrones* de problemas a las soluciones.

Cuando una aplicación se encuentra en desarrollo, el o los encargados de esta tarea se enfocan únicamente en que la aplicación cumpla con las funciones específicas para las que fue diseñada. Por este motivo, cuando se cuenta con diferentes aplicaciones desarrolladas por diferentes personas se crea naturalmente una separación. Es aquí donde tiene un papel importante la integración, debe ser capaz de proveer de forma eficiente, confiable y segura el intercambio de datos entre múltiples aplicaciones. (Berczuk , 2002)

En la etapa de desarrollo también es importante contar con un control sobre los cambios, especialmente cuando se cuenta con un equipo de personas trabajando juntas. Cada uno de los miembros debe ser capaz de realizar cambios sin interferir con el trabajo de los otros, esto también se facilita mediante el uso de patrones de integración. Por ejemplo, un miembro del equipo puede realizar un cambio en su sección de código y este cambio puede causar que toda la aplicación deje de funcionar pero si la integración se realizó por medio de un patrón, solamente habrá que adaptar el fragmento de código que realiza la conexión y las secciones de los otros miembros seguirán funcionando sin realizarles cambios. (Berczuk , 2002)

a Estilos de integración.

Debido a que las aplicaciones pueden tener diferente orígen (desarrollo propio o desarrollo externo) y pueden ser de diferente tipo (ejecución en diferente plataforma, ejecución en la nube, entre otros) existen diferentes opciones de integración. Los diferentes enfoques pueden resumirse, según Hohpe *et al.* (2011), en cuatro diferentes estilos:

Transferencia de archivos La transferencia de archivos es la forma más simple para transmisión de datos entre aplicaciones. Una de las decisiones más importantes de este estilo es el formato a utilizar para escritura y lectura de la información. Como podemos observar en la Figura 43 una de las aplicaciones realiza la tarea de generar un archivo mientras que la otra aplicación se encarga de analizar su contenido para identificar los datos de interés y utilizarlos. Uno de los formatos más utilizados en la

actualidad es el XML.

Base de datos compartida El estilo de base de datos compartida, como lo indica su nombre, integra diferentes aplicaciones haciendo que cada una de ellas almacene sus datos en una única base de datos. Esta práctica nos garantiza que la información siempre será consistente entre aplicaciones, puesto que los datos se obtienen exactamente de la misma fuente. Este estilo de integración obliga a implementar controles de administración de transacciones para actuar en casos que el mismo segmento de datos quiera ser modificado por dos aplicaciones diferentes.

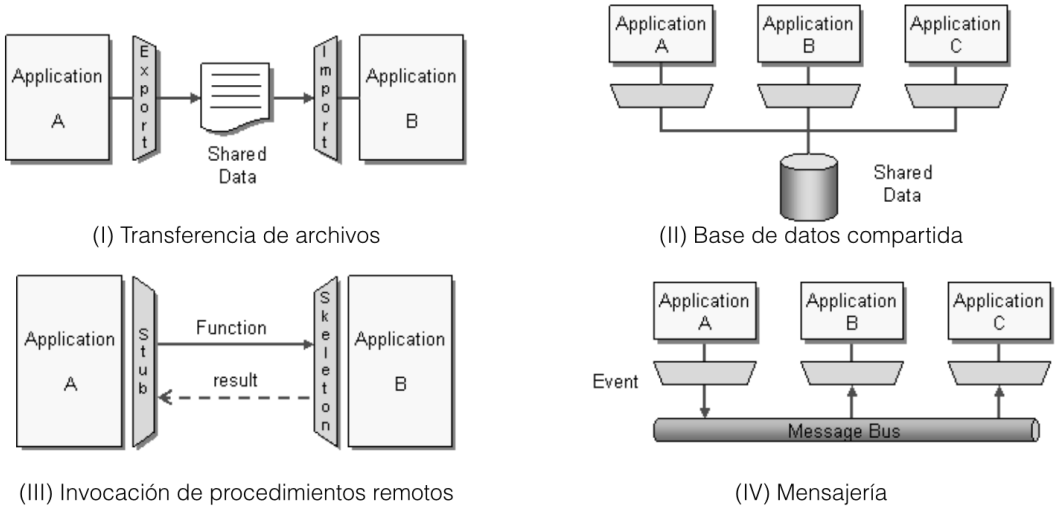
Invocación de procedimientos remotos Cuando se cuenta con dos aplicaciones como objetos independientes encapsulados es necesario proveer una interfaz que permita a una o más aplicaciones el intercambio de datos con una aplicación que se encuentra ejecutándose. Esta interfaz que se provee cuenta con métodos para obtención y modificación de datos que se encargan de realizar las llamadas correspondientes en la aplicación en ejecución y obtener resultados en caso sea necesario para entregarlos a la aplicación que realiza la solicitud. Los datos propios de cada aplicación son modificados únicamente por las mismas aplicaciones, la interfaz solamente es un mecanismo de interacción entre ambas.

Mensajería El estilo de mensajería es el más elaborado para el intercambio de datos, pero también el más robusto. Como se observa en la Figura 43 el envío de paquetes de datos se realiza por medio de un bus de mensajes. Este bus de mensajes permite el envío de datos de forma frecuente, inmediata, confiable y utilizando cualquier formato establecido. Uno de los puntos más importantes de este estilo es que el envío de mensajes no requiere que ambas aplicaciones se encuentren en ejecución por lo que las aplicaciones pueden comunicarse de forma remota. Sin embargo, el costo de el envío de mensajes puede ser elevado en cuanto a recursos por lo que solamente debe existir intercambio de mensajes en situaciones en dodne sea necesario.

b Patrón traductor. Muchas veces los objetos se encuentran organizados en estructuras de datos complejas de modo que puedan representar relaciones entre ellos mismos. En estos casos es fácil encontrar un ploblema al extraer alguno o varios objetos de su contexto original ya, como indica Kühne (1997), una estructura de árbol puede representar desde un compilador hasta una impresora. En estos casos aplica el uso del patrón traductor el cual se encarga de generar una correspondencia de uno a uno entre los elementos de la estructura original con los elementos de una nueva estructura.

El patrón traductor también puede utilizarse cuando se necesitan diferentes traducciones, o estructuras resultantes, sobre un mismo concepto dependiendo del contexto en el que se encuentra el objeto original. En la Figura 44 se puede observar que las traducciones A y B son formas concretas del concepto que serán entregadas al cliente dependiendo del contexto en el que se encuentre. Según Keller (1997) el patrón traductor tiene las siguientes ventajas:

Figura 43. Representación de los diferentes estilos de integración.



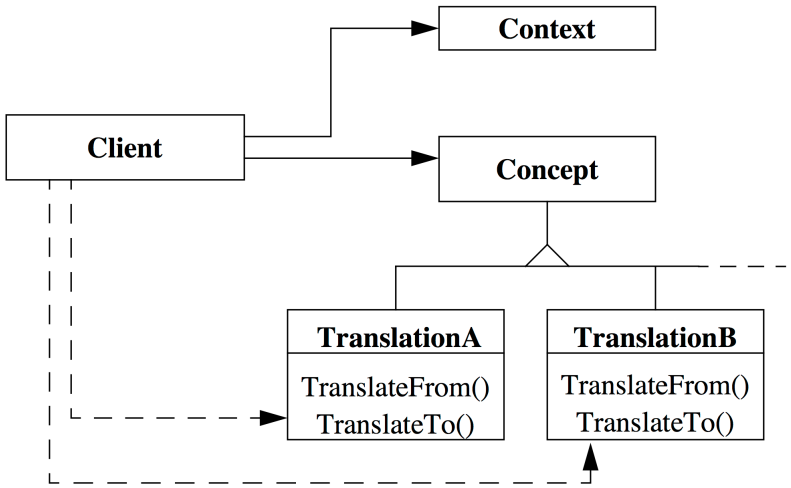
(Hohpe *et al.* , 2011)

- Nuevas traducciones pueden ser agregadas en varios puntos del ciclo de desarrollo.
- Las traducciones tienen un alto nivel de similitud por lo que se puede emplear un mecanimos semi-automático para generarlas.
- El concepto puede solamente se preocupa de sus propias operaciones ya que las traducciones son las que se encargan de los detalles específicos de las diferentes formas concretas.

2 Análisis comparativo de algoritmos. Un análisis comparativo de algoritmos indica que dos o más algoritmos van a ser comparados en base a criterios de evaluación establecidos. El objetivo de realizar una comparación es evaluar el rendimiento de cada uno de los algoritmos. Breese *et al.* (1998) realiza una comparación entre diferentes algoritmos de predicción, para ello define una serie de métricas iniciales dependiendo el tipo de filtro que se aplica y la interfaz que se provee. Adicionalmente, cuenta con los conjuntos de datos o escenarios que se van a utilizar para la evaluación. En este caso: web, televisión y películas. En esta comparación también se planifican previamente los experimentos que se van a realizar basados en los datos disponibles, un experimento utilizó votos reales de usuarios mientras que otro generó datos aleatorios para realizar predicciones.

Los conjuntos de datos utilizados para realizar comparaciones deben ser seleccionados cuidadosamente. En algunos casos es necesario realizar un pre-procesamiento de los datos para que estos puedan ser utilizados de forma rápida, estas transformaciones pueden ir desde una simple escala de datos hasta transformaciones de Fourier dependiendo del caso. Esta etapa no debe ser tomada a la ligera ya que puede representar

Figura 44. Diagrama de clases que representa el patrón traductor.



(Keller , 1997)

hasta un 80 % del esfuerzo y los resultados dependerán de que se realice correctamente. (Ghanbari *et al.* , 2010)

a Pruebas de software. Las pruebas de software son la estrategia principal para encontrar defectos de desarrollo. Se sugiere que tome entre el 40 % y el 60 % del tiempo de desarrollo, sin embargo muchas de las prubas se pueden automatizar utilizando diferentes enfoques para ahorrar tiempo. Li *et al.* (2010) indica que existen dos estrategias principales de prueba que se utilizan dependiendo del tipo de resultados que se busca y las condiciones del ambiente en el que se realiza la prueba con las que se cuenten. Estas dos estrategias se definen a continuación:

Pruebas de caja blanca Esta prueba considera la estructura interna y el comportamiento del software. Evalúa paso a paso las transformaciones que sufren los datos de prueba para verificar que los procesos sean los correctos.

Pruebas de caja negra Este tipo de prueba solamente se centra en el resultado que genera el sotware. Evalúa la entrada y salida de datos sin tomar en cuenta el comportamiento del programa bajo prueba.

b Comparación de algoritmos de TCA. Las comparaciones de algoritmos que involucran tránsito y el uso de algoritmos de inteligencia artificial deben tener una o más métricas que se buscarán optimizar. Por ejemplo Hong *et al.* (1999) propone utilizar dos métricas, reducir el tiempo de espera de los vehículos mientras se aumenta la velocidad promedio de los mismos. Para obtener estas métricas del simulador se utilizan las técnicas descritas en la sección de *Evaluaciones macroscópicas de TCA*. Por otro lado, Sánchez-Medina *et al.* (2015) indica que se debe contar con una estrategia de comparación

utilizando otros enfoques en las mismas pruebas para poder contar con un marco de comparación.

Sánchez-Medina *et al.* (2015) también propone un estudio de escalabilidad que consiste en variar la escala del mapa utilizado para evaluar el comportamiento de los diferentes enfoques utilizados al aumentar el número de calles, vehículos y semáforos. Es importante utilizar las mismas condiciones de entrenamiento en los algoritmos y que las pruebas tengan la misma cantidad de iteraciones para que las mismas tengan validez. Sin embargo, Sánchez-Medina *et al.* (2005) realiza una demostración en la que indica que es indiferente el uso de un ambiente estocástico comparado con un ambiente determinístico debido a que existe una correlación fuerte entre los resultados que se obtienen al usar uno o otro ambiente.

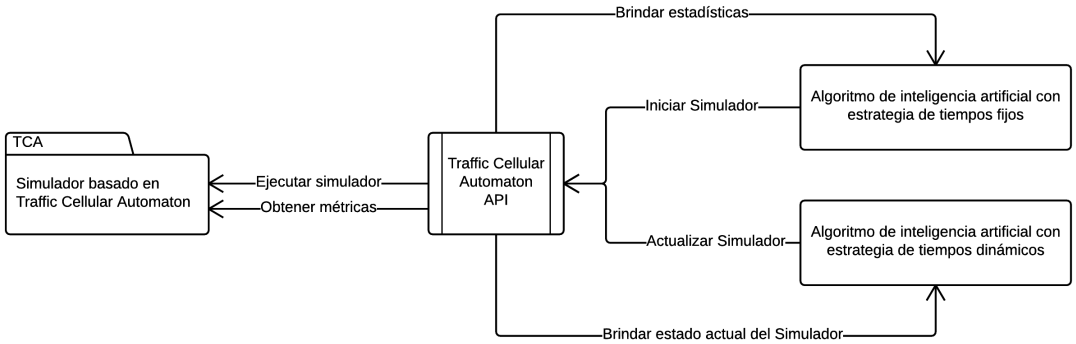
G METODOLOGÍA

Para el presente módulo se diseñará e implementará una *Application Program Interface* para integrar un simulador basado en *traffic cellular automaton* con algoritmos de inteligencia artificial que buscan optimizar los tiempos de semáforo. Adicionalmente se tomarán mediciones en el simulador de *traffic cellular automaton* para realizar un análisis de los datos obtenidos y poder comparar ambos algoritmos de inteligencia artificial. A continuación se describen los métodos que se implementaron para proveer un servicio de *traffic cellular automaton* y las métricas que se consideraron para obtener datos del simulador y realizar las comparaciones.

1 Diseño de *Application Program Interface (API)*. Para integrar al simulador basado en *traffic cellular automaton* con algoritmos de inteligencia artificial es necesario decidr por cual de los estilos de integración se va a optar. Al revisar los estilos que propone Hohpe *et al.* (2011) se consideró utilizar la integración por medio de invocación de procedimientos remotos ya que es la que se adapta de mejor forma al entorno. Se descartó la integración por medio de transferencia de archivos debido a los recursos de procesamiento innecesarios utilizados para escribir y leer archivos, al igual que introducir una base de datos para realizar la integración cuando ni el simulador ni los algoritmos de inteligencia artificial almacenan datos en una base de datos. En cuanto al estilo de integración por medio de mensajería se descartó debido a la complejidad que requría implementar un bus de mensajes cuando la principal ventaja que brindaba es el envío de mensajes cuando una de las aplicaciones no está en ejecución, pero en este caso ambas aplicaciones en comunicacion estarán en ejecución en todo momento.

a Esquema general de integración. La escalabilidad es uno de los puntos más importantes a considerar durante una integración, los diferentes componentes de una aplicación deben integrarse de tal forma que al realizar algún cambio en uno de los componentes no provoque cambios en otras secciones más que en el fragmento de código cuya función es realizar la conexión de componentes. (Berczuk , 2002) Con esto en mente, se buscó crear un esquema de integración que sea funcional para cualquier algoritmo de inteligencia artificial que se desee integrar al simulador. Como podemos observar en la Figura

Figura 45. Esquema de integración ente simulador y algoritmos de inteligencia artificial utilizando estrategias de tiempos fijos y tiempos dinámicos.



45 es necesario separar los algoritmos de inteligencia artificial en dos grupos debido a que hay dos posibles estrategias, tiempos de semáforo fijos y tiempos de semáforo dinámicos, que estos pueden emplear. El comportamiento de estas dos estrategias cambia por completo la forma de interactuar con el simulador siendo esta la principal causa por la que es necesario crear dos conjuntos independientes de métodos dentro de la *API*.

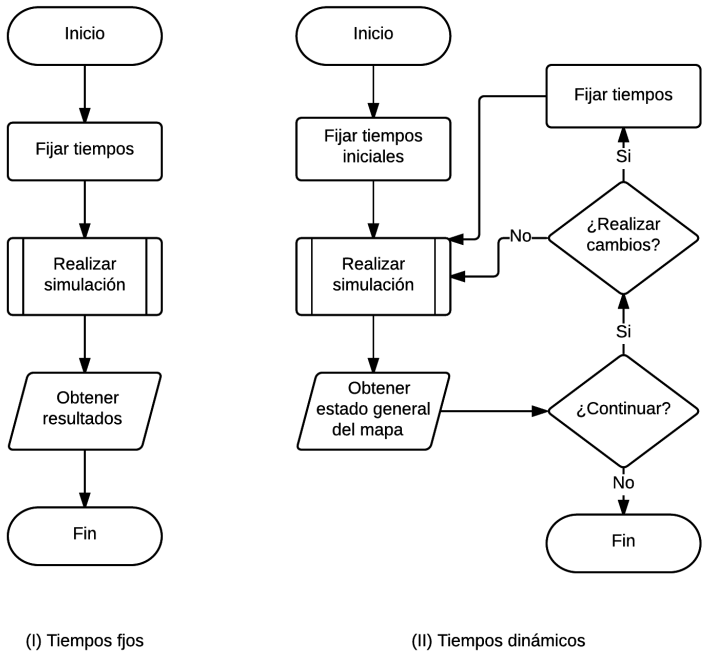
A continuación se presenta una breve descripción de la interacción entre los algoritmos de inteligencia artificial empleando estrategias de tiempos fijos y tiempos dinámicos con el simulador de *traffic cellular automaton*:

Estrategia de tiempos fijos Los algoritmos que utilizan una estrategia de tiempos fijos son aquellos que realizan un entrenamiento previo de los agentes inteligentes utilizando el simulador con lo que generan una planificación completa para el cambio de tiempos de los semáforos. Russell (2010) Para los algoritmos de inteligencia artificial que utilizan esta estrategia la *API* provee métodos con los que se puede fijar el tiempo de los semáforos previo a relizar simulaciones. Adicionalmente, ofrece métodos con los que se pueden obtener diferentes métricas dependiendo del objetivo que busquen optimizar. En la figura 46 se puede observar que un algoritmo de inteligencia que utiliza estrategia de tiempos fijos se puede separar en tres etapas: fijar tiempos, simular y obtener resultados.

Estrategia de tiempos dinámicos Los algoritmos que utilizan una estrategia de tiempos dinámicos son los que analizan la información brindada por diferentes sensores y en base a esta tomarán decisiones. Russell (2010) La interacción constante de los algoritmos de inteligencia artificial utilizando esta estrategia con el simulador requiere que la *API* provea métodos con los que se pueda cambiar el tiempo

de los semáforos en cualquier momento. También debe proveer métodos para obtener el estado del mapa completo y diferentes métricas, esto se retorna al finalizar el número de iteraciones solicitado como se puede observar en la Figura 46.

Figura 46. Diagrama de flujo representando la interacción entre el simulador de *traffic cellular automaton* y algoritmos de inteligencia artificial utilizando (I) estrategia de tiempos fijos y (II) estrategia de tiempos dinámicos.



2 Métodos de la *Application Program Interface*. Como se mencionó en la sección anterior, es necesario que la *application program interface* por medio de la cual diferentes algoritmos de inteligencia artificial pueden conectarse al simulador de *traffic cellular automaton* provea métodos con los que se obtienen o establecen datos de la simulación. Los métodos con los que se cuenta son los siguientes.

a Inicialización de la *API*. La *application program interface* que provee los servicios de TCA es un objeto creado en Python que debe instanciarse para ser utilizado. El método constructor se llama `TCAService` y recibe como parámetros el mapa que se desea utilizar y una tasa de creación de vehículos que determina la cantidad de los mismos que circulará por el mapa. Cada uno de los mapas del simulador cuenta con calles, intersecciones, semáforos y luces de semáforos definidas, cada uno posee un identificador único generado al momento de la construcción del mapa.

b Uso de simulador basado en TCA. El simulador basado en *traffic cellular automaton* que se ha utilizado es el que se desarrolló en el *Módulo de Simulación de Tránsito Vehicular a Través de un Modelo de Cellular Automaton*. Para el uso del simulador fue necesario proveer los siguientes

métodos:

`fixed_time_start` Este método está diseñado para realizar el número total de iteraciones de la simulación de forma contínua. Debido a este motivo por cada una de estas iteraciones extrae datos del simulador y los almacena en memoria. El último paso que realiza este método es procesar los datos que obtuvo de la simulación para tener las macroestadísticas listas cuando sean solicitadas.

`dynamic_time_update` Este método avanza un número determinado de iteraciones en la simulación cada vez que es llamado. Este método no almacena datos del mapa en cada iteración, puesto que solamente interesa conocer el estado en el que han quedado las calles en el mapa dentro del simulador. El estado de las calles se procesa al finalizar el avance total de iteraciones y se retorna en forma de listado. Este listado contiene diccionarios que representan cada una de las calles con los datos de id, número de carros, velocidad promedio y si la luz verde está encendida *1*, apagada *0* o *None* en caso esa calle no tenga una intersección al final. El listado tiene el siguiente formato:

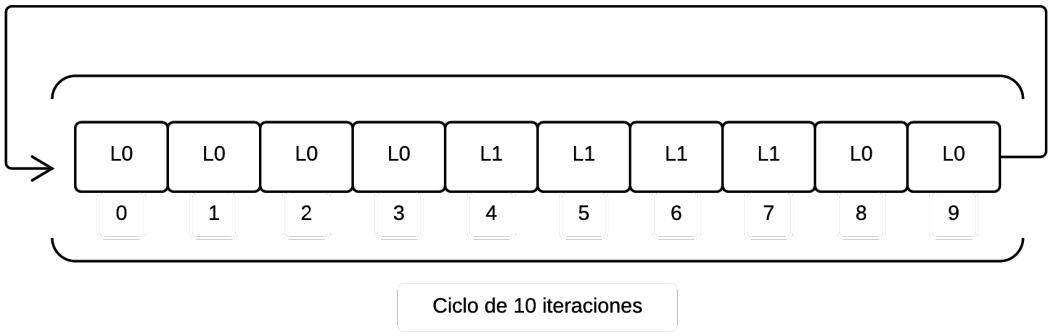
```
[{
  'id': 0,
  'cars_number': 9,
  'average_speed': 2.5,
  'green_light': 1
 }]
```

c Semáforos y luces de semáforos. El objetivo que buscan los algoritmos de inteligencia artificial es modificar los tiempos de las luces de los semáforos de tal forma que mejoren el tránsito en el simulador. Por esta razón es necesario que la *API* permita conocer la configuración de las luces en los semáforos del mapa y modificar los tiempos según sea necesario.

El simulador basado en TCA realiza los cambios de las luces de los semáforos basado en un ciclo, en el cual se asigna cada uno de las iteraciones en el ciclo a una luz en específico. En la Figura 47 se puede observar que la luz 1 (L1) estará en verde para la iteración 4, 5, 6 y 7 mientras que la luz 0 estará en verde para el resto de iteraciones en el ciclo. Este comportamiento se repetirá sucesivamente cada 10 iteraciones hasta que se modifica la configuración.

`get_traffic_lights` Este método es utilizado para obtener un listado de semáforos en donde se especifican las luces que se encuentran en cada semáforo y la configuración de las mismas durante un ciclo. El listado de retorno contiene el id del semáforo, las luces que pertenecen y la configuración de la distribución de tiempo de las luces. El listado de retorno tiene el siguiente formato:

Figura 47. Diagrama de asignación de luces de semáforos para un ciclo de 10 iteraciones en el simulador de *traffic cellular automaton*.




```
[{
  'id': 0,
  'lights': [0, 1],
  'schedule': {0: 0, 5: 1}
 }]
```

La configuración de tiempos, *schedule*, utiliza el formato `{start: id}` donde *start* es la iteración dentro del ciclo en la que la luz con identificador *id* va a cambiar a verde.

`set_traffic_lights` Este método es el que se utiliza para fijar una nueva configuración de la distribución de tiempo de las luces para los diferentes semáforos dentro del mapa. Para utilizarlo solamente se debe enviar como parámetro un listado de diccionarios que contiene las configuraciones que se desean fijar, el método retornará *True* en el caso que el cambio se relizó correctamente, *False* en caso existe algún error con el listado enviado o bien una excepción. La excepciones se detallarán más adelante en la sección *Manejo de excepciones*. El listado que se debe enviar como parámetro debe tener diccionarios indicando el identificador del semáforo que se modificará y la nueva configuración con el siguiente formato:

```
[{
  'id': 0,
  'schedule': {0: 0, 5: 1}
}]
```

`get_cycle_size` Este método se encarga de retornar el tamaño de ciclo actual que posee el simulador de TCA.

`set_cycle_size` Este método se encarga de fijar el tamaño de ciclo en el simulador de TCA en caso que se necesite modificar.

d Estructura del mapa. Algunos algoritmos de inteligencia artificial solamente consideran el mapa como un todo, es indiferente la estructura que este posea. Sin embargo, otros algoritmos, se basan en la estructura que el mapa posee para tomar sus decisiones de asignación de tiempos a las luces de los semáforos. La *API* provee un método con el cual es posible obtener la estructura del mapa sobre el cual se están realizando las simulaciones, este método se datalla a continuación:

`get_intersections` Este método representa cada una de las interesecciones indicando su identificador, el identificador del semáforo que está asignado a esa intersección en caso cuente con uno, las calles que están conectadas a esa intersección con vía entrante y las que están conectadas con vía saliente, por último se indican las intersecciones vecinas, aquellas a las que se pueda llegar por medio de una calle que conecte a ambas. El listado se retorna con el siguiente formato:

```
[{
  'id': 0,
  'traffic_light': 0,
  'in_streets': [0, 1],
  'out_streets': [2, 3],
  'neighbors': [3, 4]
}]
```

e Mediciones estadísticas dentro del mapa. Una parte esencial para que los algoritmos de inteligencia artificial puedan entrenarse o tomar decisiones es contar con datos históricos y de un preciso instante en el simulador. Algunas de las mediciones estadísticas se mencionaron en el método de `dynamic_time_update` ya que el retorno de este método incluye un conteo de vehículos y velocidad promedio de los mismos en cada una de las calles del mapa. Adcionalmente la *API* cuenta con otros métodos basados en las propuestas de y para obtener datos estadísticos que se describen a continuación:

`get_average_speed` Este método retorna la velocidad promedio de todo el mapa dentro del simulador desde que se inició la simulación hasta el momento en el que se solicita. Este valor es calculado en base a la siguiente fórmula:

$$\frac{\left(\frac{\sum_{i=1}^{n} \text{carro } i}{\text{número de carros}}\right)}{\text{número de iteraciones}}$$

`get_average_cars_number` Este método es utilizado para obtener la cantidad promedio de vehículos que ha existido por iteración durante la simulación. Por razones semánticas se retorna el entero más cercano de tal forma que no existan fracciones de vehículos, la fórmula utilizada para realizar este cálculo es la siguiente:

$$\frac{\sum_{i=1}^{n} \text{número de carros en iteración } i}{\text{número de iteraciones}}$$

`get_stopped_time` Este método retorna la cantidad total de veces en las que un vehículo ha estado detenido desde el inicio de la simulación hasta el momento en que se realiza la llamada al método. Se considera que un vehículo está detenido cuando su velocidad es cero. El cálculo realizado se basa en la siguiente fórmula.

$$\sum_{i=1}^{n} \text{número de carros con velocidad 0 en iteración } i$$

`get_average_stopped_time` Este método es similar a `get_stopped_time` con la diferencia que el tiempo que retorna es el promedio de vehículos que ha estado detenido por cada una de las iteraciones y no el tiempo total. Se calcula en base a la siguiente fórmula:

$$\frac{\sum_{i=1}^{n} \text{número de carros con velocidad 0 en iteración } i}{\text{número de iteraciones}}$$

`reset_statistics` Este método se encarga de retornar todas las estadísticas a su estado inicial, así como regresar el contador de iteraciones a cero. Es utilizado cuando se desea iniciar con un nuevo entrenamiento en el que es necesario contar con nuevas mediciones en el simulador.

f Manejo de excepciones. El manejo de excepciones es un punto importante dentro del software para que este no presente errores inesperados al momento de su ejecución, según indica Deb (1999). La *API*, por ser un punto de conexión entre otros dos sistemas es bastante suceptible a contar con situaciones inesperadas que podrían detener la ejecucuón de la simulación por completo. Tomando en cuenta esto se identificó los puntos en donde era necesario contar con manejo de excepciones, estos se detallan a cotinuación:

Ejecución de la simulación Siendo el simulador un sistema aislado es posible que presente algún error mientras se realizan las llamadas de actualizaciones de simulación. Por este motivo se tomó la decisión de capturar cualquier excepción que el simulador presente en indicarle al usuario de la *API* que sucedió un problema junto con el mensaje de error que se obtuvo del simulador. Esto sin interrumpir la ejecución de los algoritmos que se encuentran utilizando el simulador.

Cambio en el tamaño de ciclo El tamaño de ciclo es una propiedad de uno de los objetos principales del simulador de TCA por lo que para cambiarlo solamente se sustituye el valor anterior por el nuevo.

Sin embargo, al ser un componente importante para el funcionamiento del simulador se optó incluirse en el manejo de excepciones de forma similar a la ejecución de la simulación.

Cambio en configuración de semáforos Realizar cambios en la configuración de semáforos involucra que el parámetro recibido debe estar bien formado y contener información existente dentro del del simulador por lo tanto hay más de un punto en el cual se pueden generar errores inesperados. Dentro del diccionario se envían identificadores de semáforos y de luces, por lo que se verifica como primer paso que los identificadores existen en el simulador y de lo contrario se lanza la excepción `InvalidTrafficLightId` o la excepción `InvalidTrafficLightId` creadas para este efecto. Adicionalmente se verifica la estructura del listado recibido como parámetro y en caso de no encontrarse bien formado se lanza una excepción junto con el mensaje del error.

3 Comparación de algoritmos de inteligencia artificial. La segunda parte de este módulo busca realizar una comparación entre dos algoritmos de inteligencia artificial desarrollados en otro módulo de Megaproyecto. El *Módulo de diseño e implementación de un algoritmo genético para la optimización de ciclos de semáforo en intersecciones de una cuadrícula* desarrolló, como lo indica su nombre, un algoritmo genético. El *Módulo de diseño e implementación de un sistema multiagente con aprendizaje por refuerzo para optimizar el tráfico* desarrolló un algoritmo de inteligencia artificial con aprendizaje por refuerzo utilizando una estrategia multiagente. Estos algoritmos se pondrán a prueba junto a dos enfoques de algoritmos aleatorios similar a como Sánchez-Medina *et al.* (2015) lo realizan en su estudio. El simulador que se utilizará es el desarrollado por el *Módulo de simulación de tránsito vehicular a través de un modelo de cellular automaton*, el mismo para el cual se realizó la *API* mencionada en las secciones anteriores.

a Mapas. Los mapas utilizados para realizar la comparación de los diferentes algoritmos se pueden observar en la Figura 48. Se utilizaron los mapas disponibles en el simulador de TCA aumentando su tamaño de forma incremental. El mapa *pequeño* solamente cuenta con un carril en sus calles mientras que el mapa *mediano* y *grande* cuentan con dos carriles en sus calles y permiten que los vehículos cambien de carril. Todos los mapas cuentan con las vías alternadas, es decir si una calle va hacia la derecha, la siguiente irá hacia la izquierda. Lo mismo sucede con las calles con dirección arriba y abajo.

b Densidad de vehículos. Para realizar la comparación de algoritmos es necesario, además de variar los mapas, utilizar diferentes densidades de vehículos para poder evaluar el comportamiento de los algoritmos de acuerdo a la variación de las condiciones vehiculares. El simulador de TCA permite variar la tasa de generación de vehículos con lo que se logran diferentes densidades de vehículos. Las densidades utilizadas para la comparación son las siguientes: 20 %, 40 %, 60 %, 80 %, 100 %.

c Número de iteraciones. Para asegurar que el comportamiento de un algoritmo en específico se mantiene constante a lo largo del tiempo se realizó una simulación utilizando 200 iteraciones

Figura 48. Mapas utilizados en la simulación para la comparación de algoritmos de inteligencia artificial. *Pequeño:* 4 intersecciones. *Mediano:* 8 intersecciones. *Grande:* 16 intersecciones.

(I) Pequeño (II) Mediano (III) Grande

y una segunda utilizando 1,000 iteraciones para poder evaluar si el algoritmo mantiene una tendencia en las condiciones del mapa.

H RESULTADOS

En la evaluación de rendimiento de diferentes algoritmos es necesario contar con varios enfoques que brinden datos suficientes para poder realizar una comparación. En la sección de *Comparación de algoritmos de inteligencia artificial* se plantearon los diferentes enfoques que se tomaron para realizar la evaluación. Adicional a los resultados obtenidos con el algoritmo genético *(GA)* y el algoritmo multiagente con aprendizaje por refuerzo *(MA)* se agregaron resultados de dos diferentes enfoques con estrategias aleatorias. Uno con estrategia de tiempos fijos *(FR)* y uno con estrategia de tiempos dinámicos *(DR)*. La velocidad utilizada en todas las simulaciones es de 3 *celdas/iteración* debido a que el simulador de TCA no permite fijar otro valor. A continuación se presentan los resultados encontrados en las diferentes simulaciones.

1 Análisis de escalabilidad. Los algoritmos comparados en el presente módulo deben ser capaces de optimizar más de un mapa, especialmente si lo que se busca es poder implementar un sistema inteligente en una ciudad donde existen diferentes configuraciones de calles e intersecciones. Tomando esto en cosideración se realizó un análisis de escalabilidad aumentando el tamaño del mapa como lo realiza Sánchez-Medina *et al.* (2015) en su estudio. En este caso se utilizó una densidad vehicular de 80 % en los tres mapas sobre los cuales se realizó la simulación. En el Cuadro 1 podemos observar los resultados obtenidos con cada uno de los algoritmos en los diferentes mapas. Cabe mencionar que en la mitad de los casos el rendimiento fue decreciendo conforme se aumentó el tamaño del mapa, pero el algoritmo genético y el aleatorio con estrategia de tiempos dinámicos lograron obtener un mejor resultado en el mapa grande

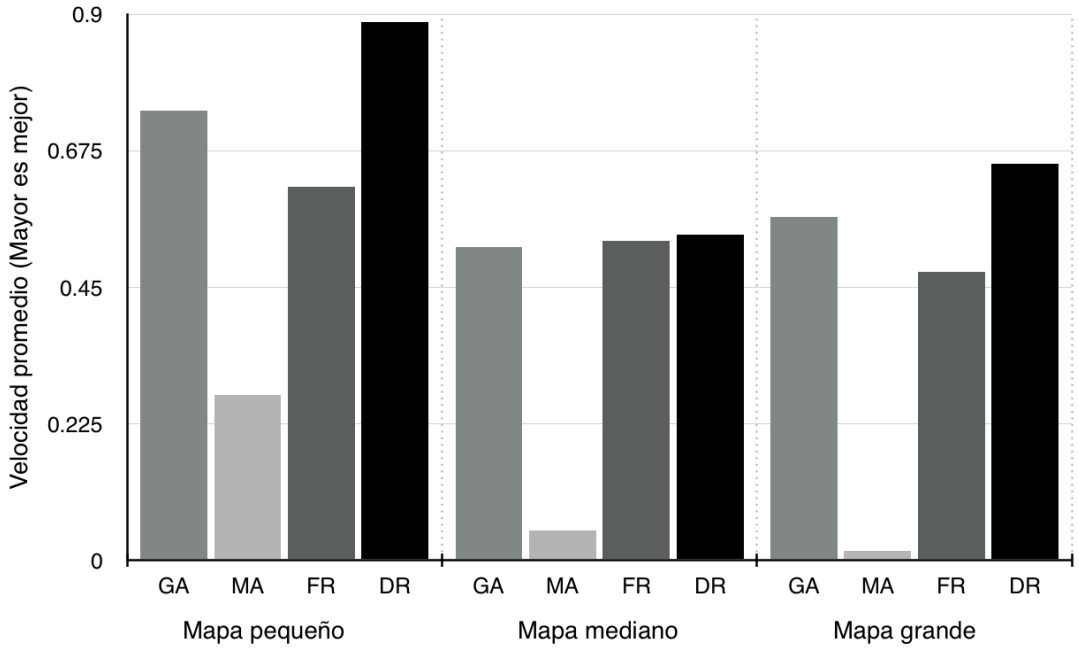
comparado al obtenido en el mapa mediano.

Cuadro 1. Velocidad promedio en *celdas/iteración* utilizando algoritmos DR, FR, MA, GA en mapas pequeño, mediano y grande con 80 % de densidad vehicular.

Mapa	DR	FR	MA	GA
Pequeño	0.88704	0.61529	0.27252	0.74044
Median	0.53631	0.52607	0.04946	0.51608
Grande	0.65275	0.47489	0.01546	0.56543

El algoritmo que presentó mejores resultados durante la totalidad de las simulaciones fue el algoritmo aleatorio con estrategia de tiempos dinámicos *DR* con valores de velocidad promedio de 0,88704 para el mapa pequeño, 0,53631 para el mapa mediano y 0,65275 para el mapa grande. Estos resultados se pueden observar gráficamente en la Figura 49 en donde la barra correspondiente a este algoritmo se encuentra por encima de las de los otros algoritmos.

Figura 49. Gráfica mostrando la velocidad promedio en *celdas/iteración* obtenida utilizando los 4 diferentes algoritmos (DR, FR, MA, GA) en 3 mapas diferentes (pequeño, mediano, grande) utilizando 80 % de densidad vehicular.



Un aspecto importante a considerar es el desempeño del algoritmo multiagente con aprendizaje por refuerzo *MA* debido a que es el que presenta el peor rendimiento de todos los algoritmos evaluados y su rendimiento decrece considerablemente al aumentar el tamaño del mapa llegando a una velocidad promedio de 0,01546*celdas/iteración* al realizar la simulación en un mapa grande.

El algoritmo *DR* presentó un rendimiento 15.44 % mejor que el algoritmo *GA*, 37.42 % mejor que el

algoritmo *FR* y 4,122.19 % mejor que el algoritmo *MA* en la prueba realizada en el mapa grande.

2 Análisis de densidad de vehículos. Cuando hablamos de tránsito en una ciudad es difícil imaginar una ciudad que mantiene una cantidad de vehículos constante, por este motivo se realizó una análisis de densidad de vehículos. Este análisis consiste en realizar diferentes simulaciones aumentando gradualmente la densidad de vehículos dentro del mapa para obserbar el cambio de comportamiento de los algoritmos al cambiar las condiciones dentro del mismo mapa. El cambio de densidad se realiza modificando la tasa de ingreso de vehículos al mapa, esto quiere decir que con una densidad de 10 % se generará 1 vehículo cada 10 iteraciones en los puntos de generación dentro del mapa. La densidad vehicular inicial fue de 20 % y se aumentó 20 % en cada simulación hasta que se llegó a una densidad vehicular final de de 100 % en la que se generan vecículos nuevos en cada una de las iteraciones de la simulación.

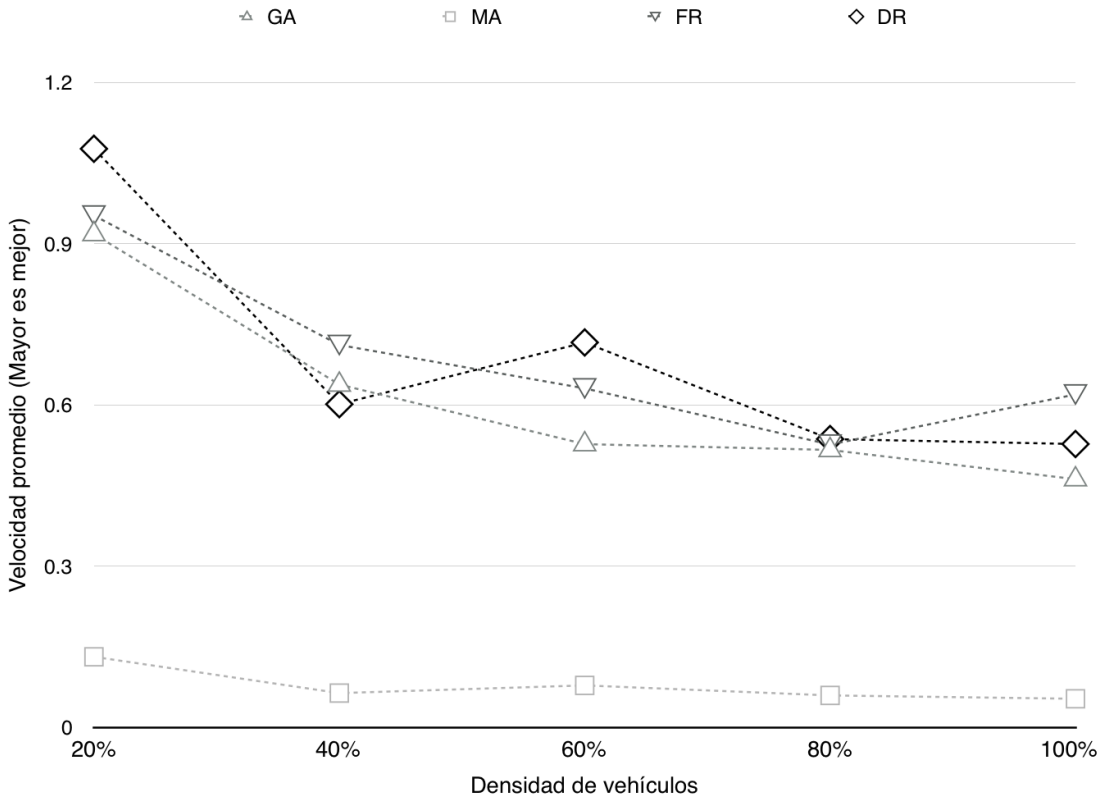
En la Figura 50 podemos observar el comportamiento de la velocidad promedio de los diferentes algoritmos al ir aumentando la densidad vehicular. Se observa una tendencia a disminuir la velocidad promedio a medida que se aumenta la densidad, como se esperaría ya que mientras más vehículos hay dentro del mapa existe mayor dificultad para desplazarse. Sin embargo, llama la atención que para el algoritmo *DR* hay un notable aumento de velocidad promedio cuando la densidad es de 60 %, pero tomando en cuenta que es un algoritmo aleatorizado, es problable que esto se deba a que las condiciones favorecieron al algoritmo en la simulación utilizando esa densidad.

Una vez más podemos notar que el algoritmo multiagente con aprendizaje por refuerzo se encuentra muy por debajo de los otros algoritmos evaluados. Otro punto interesante a observar es que a partir de una densidad de 40 % los resultados son prácticamente los mismos para todas las simulaciones lo que indicaría que sus mejores resultados se obtienen en un mapa con poca carga vehicular. Sin embargo, los resultados para el 20 % son 719.77 % inferiores a los de su competidor aleatorio con estategia de tiempos dinámicos, muy por debajo de lo esperado.

Observando la gráfica en la Figura 50 es posible observar que el algoritmo *DR* es superior o se encuentra muy cercano a los otros algoritmos en las diferentes simulaciones realizadas. Los algoritmos *FR* y *GA* presentan resultados muy similares siendo el primero ligeramente mejor en las primeras simulaciones y con una diferencia mayor cuando la densidad es de 100 %, es importante mencionar que ambos algoritmos utilizan una estrategia de tiempos fijos con lo que se podría explicar la similitud en comportamiento. Los tiempos obtenidos por cada uno de los algoritmos se pueden observar con mayor detalle en el Cuadro 2.

Haciendo relación a los tipos de algoritmos observamos que en este caso los dos algoritmos que utilizan aleatorización son los que obtienen un mejor resultado evaluando las cinco simulaciones. El promedio en la velocidad al considerar las cinco simulaciones es 0,691384*celdas/iteración* para el algoritmo *DR*, 0,68789*celdas/iteración* para el algoritmo *FR*, 0,61205*celdas/iteración* para el algoritmo *GA* y

Figura 50. Gráfica que muestra el cambio en velocidad promedio *celdas/iteración* al aumentar la densidad vehicular. Análisis realizado en los algoritmos DR, FR, MA y GA utilizando densidades de 20 %, 40 %, 60 %, 80 % y 100 % en un mapa mediano.



0,077176*celdas/iteración* para el algoritmo *MA*. Con esto se confirma lo mencionado anteriormente que los algoritmos con técnicas aleatorias tanto con estrategia de tiempos fijos como dinámicos son los que obtuvieron un mejor resultado en este análisis.

Cuadro 2. Cambio de velocidad promedio en *celdas/iteración* de los algoritmos DR, FR, MA, GA al aumentar la densidad vehicular utilizando un mapa mediano.

Algoritmo	20 %	40 %	60 %	80 %	100 %
DR	1.07627	0.60123	0.71601	0.53631	0.52710
FR	0.95215	0.71133	0.63062	0.52607	0.61928
MA	0.13129	0.06380	0.07810	0.05946	0.05323
GA	0.91775	0.63777	0.52691	0.51608	0.46172

3 Análisis de tiempo de simulación. Al observar los resultados obtenidos por el algoritmo multiagente con aprendizaje por refuerzo fue neceario realizar un análisis incluyendo simulaciones con un mayor número de iteraciones. Para este análisis se realizaron simulaciones utilizando diferentes densidades vehiculares y diferentes mapas con la diferencia que las simulaciones contaron con 1,000 iteraciones.

En el Cuadro 3 se puede observar el comportamiento de los diferentes algoritmos durante un mayor tiempo de simulación. Es importente observar que el algoritmo *MA* antes de llegar a la iteración número 200 ya se encuentra en una situación que no se observa en ningún otro algoritmo, parece ser un punto de no retorno en el que todos los vehículos se han detenido y es por ello que la velocidad promedio es 0,0*celdas/iteración* y el tiempo de espera es de 86*iteraciones* que pareciera ser debido a que ingresaron 86 vehículos al mapa y no se pueden mover.

Cuadro 3. Cambio de velocidad promedio (VP) en *celdas/iteración* y tiempo de espera (TE) en *iteraciones* de vehículos durante 1000 iteraciones en intervalos de 100 utilizando los algoritmos DR, FR, MA, GA en un mapa pequeño con densidad vehicular de 80 %.

Iteracion	DR		FR		MA		GA	
	VP	TE	VP	TE	VP	TE	VP	TE
100	0.57353	45	0.52542	41	0.25758	59	0.44286	54
200	0.58904	48	0.53030	48	0.00000	86	0.15116	78
300	0.75000	41	0.28986	55	0.00000	86	0.19101	75
400	0.51807	56	0.39189	58	0.00000	86	0.20212	79
500	0.64634	49	0.28986	54	0.00000	86	0.23913	75
600	0.55422	56	0.50000	47	0.00000	86	0.19588	83
700	0.80488	43	0.40000	54	0.00000	86	0.36082	72
800	0.69737	41	0.56757	48	0.00000	86	0.31683	77
900	0.85185	39	0.44304	53	0.00000	86	0.36190	77
1000	0.45652	62	0.57895	47	0.00000	86	0.45455	75

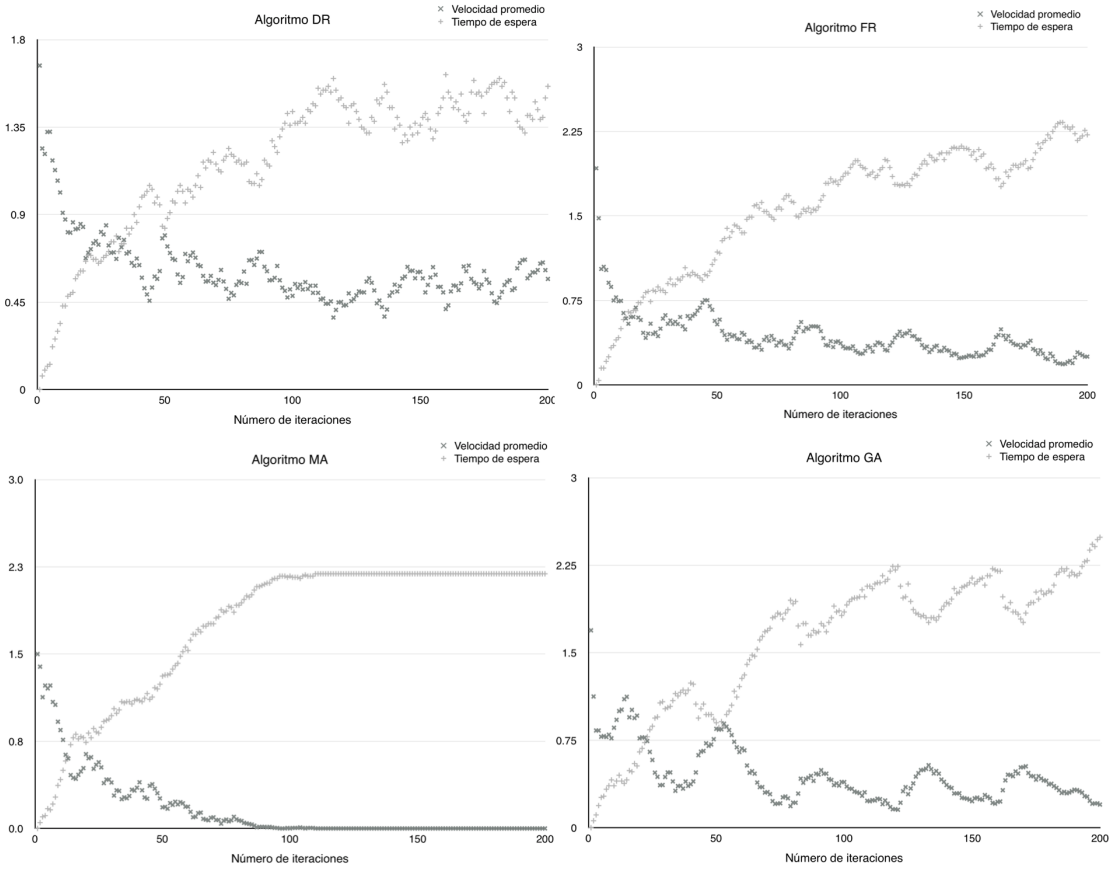
Una característica que es importante resaltar es que el algoritmo *FR* parece estar oscilando cada 100 iteraciones sus resultados. Por ejemplo se puede observar un patrón 54, 47, 54, 48, 53, 47 al observar la columna de tiempo de espera de vehículos. Este algoritmo al contar con una estrategia de tiempos fijos resulta normal que tenga un comportamiento como el descrito ya que cada cierto período se repite la secuencia de luces de semáforo llevando al mapa a condiciones similares. Lo interesante de observar es que el otro algoritmo que utiliza esta estrategia no cuenta con patrones similares, más bien parece tener resultados constantes mientras se avanza en iteraciones.

4 Análisis de cantidad y tiempo de espera de vehículos. Este análisis busca identificar alguna relación entre la cantidad o el tiempo de espera de vehículos con la velocidad promedio del mapa. Para ello se colocó en una misma gráfica el comportamiento de la velocidad promedio y el comportamiento de los vehículos.

En la Figura 51 se puede observar la realación entre la velocidad promedio y el tiempo de espera de vehículos en cuatro gráficas, una por cada uno de los algoritmos. De forma similar en la Figura 52 se observa la relación entre la velocidad y la cantidad de vehículos dentro del mapa.

En las dos gráficas se observa que para el algoritmo *MA* la velocidad desciende hasta el punto de llegar a 0,0*celdas/iteración* pero hay un aspecto importante a observar y es que en la Figura 52 la cantidad de

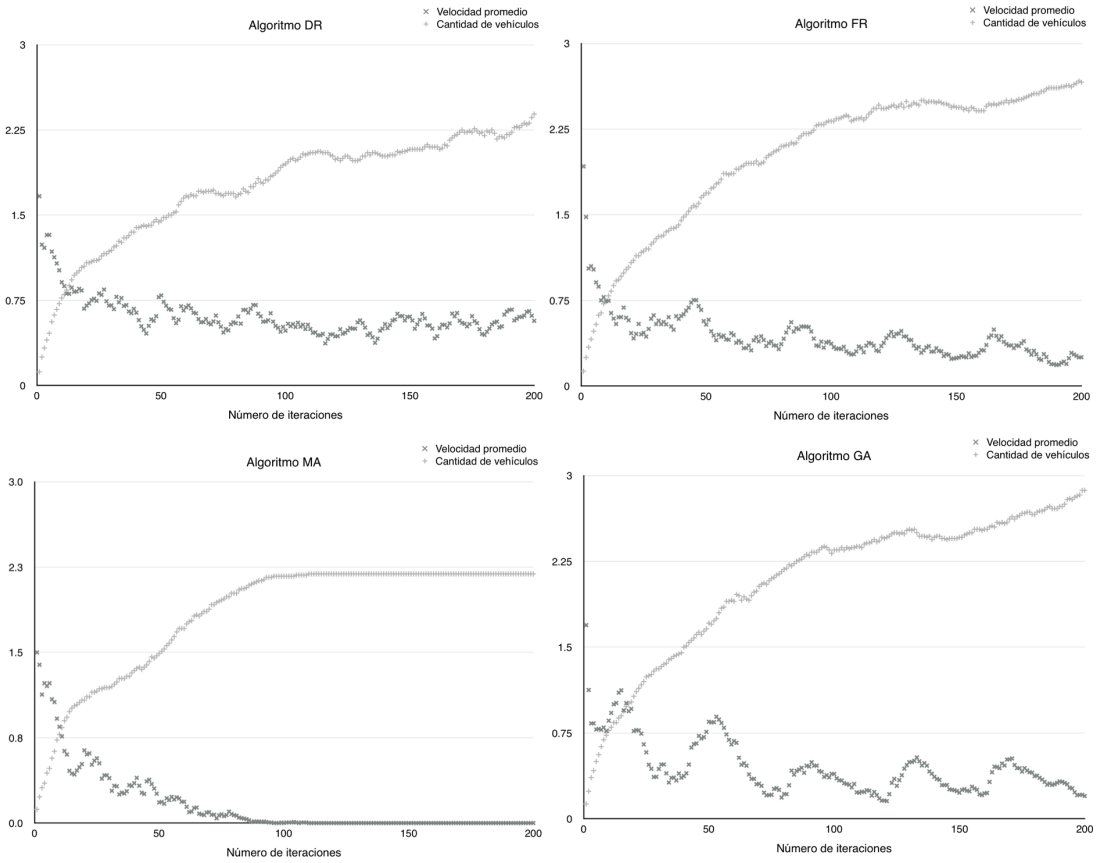
Figura 51. Gráficas que muestran la relación entre velocidad promedio en *celdas/iteración* y tiempo de espera de vehículos en *iteraciones*. Análisis realizado en los algoritmos DR, FR, MA y GA utilizando 80 % de densidad vehicular en un mapa grande.



vehículos dentro del mapa llega a un punto máximo a diferencia de las otras gráficas que mantiene una tendencia de crecimiento. Este comportamiento confirma la suposición que hay un punto en el que el mapa se satura impidiendo que los vehículos puedan moverse y por lo tanto tampoco es posible que incresen más vehículos al mapa estando saturados los puntos de generación de vehículos. Debido al funcionamiento del simulador una vez se alcanza este estado no es posible salir de él.

En la Figura 51 la relación entre velocidad promedio y tiempo de espera de vehículos parece ser muy clara, las gráficas presentan la misma forma pero invertidas. Esto es lo esperado debido a que al aumentar la velocidad el tiempo de espera de los vehículos disminuye, las gráficas mostradas confirman este comportamiento. Por otro lado, el comportamiento de las gráficas en la Figura 52 parece ser diferente según la estrategia que se utiliza. Al observar las gráficas de los dos algoritmos con estrategia de tiempos fijos se observa una relación bastante cercana en la forma del incremento de vehículos y en el comportamiento periódico de la velocidad promedio siendo más marcado el del algoritmo *GA* debido a que no cuenta con factores aleatorios.

Figura 52. Gráficas que muestran la relación entre velocidad promedio *celdas/iteración* y la cantidad de vehículos dentro del mapa. Análisis realizado en los algoritmos DR, FR, MA y GA utilizando 80 % de densidad vehicular en un mapa grande.



Lo que más llama la atención es el comportamiento de la cantidad de vehículos en la gráfica del algoritmo *DR* debido a que parece no tener un crecimiento tan acelerado en el inicio similar al crecimiento inicial del algoritmo *MA*. Considerando que todos los mapas tienen las mismas condiciones de densidad vehicular, resulta interesante mencionar que la cantidad de vehículos al utilizar el algoritmo *DR* en ningún momento llega a ser tan alta como en los otros mapas (descartando el de *MA* que deja de aumentar por otras causas). Al comparar este comportamiento con el de su velocidad promedio que incluso tiene un ligero incremento al final pareciera que este algoritmo permite una mayor circulación de vehículos con lo que logra liberar de cierto modo el mapa teniendo como resultado que no llegue a niveles tan altos la cantidad de vehículos.

I DISCUSIÓN

Los resultados encontrados en los diferentes análisis realizados parecen indicar a primera vista que es mejor utilizar una estrategia aleatoria para configuración de tiempos en el semáforo por inusual que parezca. Sin embargo, es necesario entender el porqué de los resultados de los algoritmos de inteligencia artificial y

no descartarlos sin realizar una análisis sobre ellos.

Como primer punto Deb (1999) indica que los algoritmos genéticos multiobjetivo, como el utilizado para el análisis, pueden tener problemas para superar algunas pruebas debido a la orientación a un único objetivo de estas. Este puede ser el caso en muchos de los análisis realizados donde se le dio prioridad a la velocidad promedio ignorando si el tiempo de espera de los vehículos estaba disminuyendo. Este no es el caso para los resultados mostrados en la Figura 51 ya que allí si es tomado en cuenta el tiempo de espera y parece no tener un impacto alto ya que ambos objetivos se comportan de forma contraria a lo esperado, descartando los problemas que indica Deb (1999) como causante de resultados con bajo rendmiento en el análisis.

Por otro lado existe la posibilidad que el problema no sea el algoritmo genético como tal sino un entrenamiento pobre. Esto se debe principalmente a la poca disponibilidad de recursos en el *Módulo de diseño e implementación de un algoritmo genético para la optimización de ciclos de semáforo en intersecciones de una cuadrícula* por lo que el entrenamiento del algoritmo desarrollado en ese módulo no fue sometido a un entrenamiento tan riguroso. Sería necesario realizar más pruebas para descartar esta como una causa del bajo rendimiento.

En cuanto al algoritmo multiagente con aprendizaje por refuerzo sus resultados demuestran que no es capaz de tomar las decisiones adecuadas con la información que obtiene del mapa, empeorando esta situación notablemente em mapas de mayor tamaño. Contrastando con el algoritmo aleatorio que utiliza la misma estrategia de tiempos dinámicos, siendo este el que mejor rendimiento presentó en general no es posible afirmar que el problema se encuentra en la estrategia utilizada. Tomando esto en consideración se debe revisar la sección de toma de decisiones del algoritmo *MA* en busca de posibles errores al momento de la implementación. Adicionalmente, debido a que se encontró que la causa de la disminución de la velocidad promedio es que se llega a un estado en el que todos los vehículos se detienen en el simulador podría contemplarse la idea de realizar pruebas de este algoritmo utilizando otro simulador de tránsito para descartar que este sea el causante del bajo rendimiento del algoritmo.

Los resultados tan variados, especialmente como se observa en el Cuadro 1 que algoritmos con diferentes estrategias y diferentes son los que presentan mejores resultados no es posible descartar alguno de los algoritmos utilizados. A diferencia de los resultados de Sánchez-Medina *et al.* (2015) en los que uno de los algoritmos claramente era mejor que los otros utilizados, en este caso resulta difícil sugerir que alguno es mejor que otro. El algoritmo aleatorio con una estrategia de tiempos dinámicos fue el de mayor rendimiento en las comparaciones realizadas en general pero estos resultados no pueden ser concluyentes debido a que la naturaleza aleatoria de este algoritmo no permite afirmar que su comportamiento se mantendrá con la misma tendencia.

J CONCLUSIONES

Respecto a las estrategias utilizadas por los algoritmos en las pruebas realizadas, no es posible determinar si alguna presenta mejor rendimiento debido a que los dos mejores algoritmos encontrados utilizan estrategias diferentes. Sin embargo la estategia de tiempos dinámicos presentó mejores resultados en la liberación de vehículos en el mapa.

En cuanto al bajo rendimiento de los algoritmos, con la información encontrada no se puede determinar una causa específica. Sin embargo se logró identificar en que puntos presentan deficiencias cada uno de los algoritmos de inteligencia artificial para poder contar con una revisión de su implementación.

Por otro lado, no se recomienda utilizar el algoritmo que presentó mejores resultados debido a su factor de aleatoriedad. El candidato a implementarse en caso es el algoritmo genético implementado en el *Módulo de diseño e implementación de un algoritmo genético para la optimización de ciclos de semáforo en intersecciones de una cuadrícula*.

K RECOMENDACIONES

El simulador de TCA utilizado para las pruebas demostró cumplir con su objetivo, sin embargo algunos puntos pueden ser mejorados para contar con la disponibilidad de realizar otro tipo de pruebas. Uno de estos puntos es la velocidad máxima permitida que no puede ser modificada durante la ejecución y contar con uan velocidad máxima baja podría resultar perjudicial para los algoritmos de inteligencia artificial. El segundo punto es la capacidad de modificar la tasa con la que se generan vehículos, para realizar la prueba de densidad vehicular fue necesario ralizar simulaciones independientes eliminando la posibilidad de evaluar el comportamiento al modificar las condiciones del mapa durante la simulación.

En cuanto a los algoritmos de inteligencia artificial se recomienda realizar una revisión para mejorar los puntos débiles identificados. Para el algoritmo genético se debe realizar un entrenamiento más riguroso utilizando los parámetros correspondientes para cada escenario y en un equipo con mejor poder de procesamiento para que el entramiento sea viable. Con el algoritmo multiagente con aprendizaje por refuerzo, se recomienda revisar la sección de toma de decisiones y verificar que las mismas tengan efecto dentro del simulador para evitar que llegue a un punto con todos los vehículos detenidos.

IX. MÓDULO DE COMUNICACIÓN Y ALMACENAMIENTO LOCAL

A. INTRODUCCIÓN

El módulo de comunicación y almacenamiento local es la interfaz entre sensor y base de datos, y entre este último y dispositivo portátil del usuario. Permite que la información obtenida por el módulo de detección de tránsito vehicular sea almacenada en una base de datos de fácil acceso y manipulación, y que las recomendaciones brindadas por el sistema de computación sean recibidas correctamente por el dispositivo portátil, siendo posible para este último transmitir notificaciones de eventos que afectan al tránsito vehicular y que esta información sea almacenada localmente.

Básicamente, este módulo se enfoca en dos asuntos primordiales. El primero es la selección del medio de comunicación y el desarrollo del protocolo para su utilización, y el segundo es la selección de la plataforma de control, en la cual también se realiza el almacenamiento local. En ambos casos, en la fase de pruebas se empieza a trabajar con los dispositivos disponibles en el departamento de Electrónica de la Universidad, y dependiendo de sus limitantes, se evalúa la necesidad de conseguir herramientas con más funciones integradas o mayor alcance.

En cuanto al medio de comunicación, se consideran los requerimientos de distancia, costo y facilidad para su uso y manipulación de datos. Por otro lado, la selección de la base de datos a utilizar depende del módulo de control seleccionado.

La metodología empleada para el desarrollo del módulo fue asignar tareas semanales para entregar paulatinamente, presentando conclusiones y resultados importantes aproximadamente cada dos semanas. Las categorías en que se puede resumir cada fase corresponden a una etapa de investigación, una etapa de pruebas de campo para verificar la selección de tecnologías, y el desarrollo del programa para la comunicación.

Con esto, el módulo es capaz de recibir y almacenar la información de los sensores conectados a la red al mismo tiempo que establece conexión con los dispositivos portátiles para el intercambio de mensajes. Se tiene un grado de seguridad en la conexión con este último al implementar un protocolo de inicio de sesión, y se tiene verificación de entrega del paquete. Para ambos casos se asegura la integridad de los datos por medio de verificación por redundancia cíclica, y se codifican los datos cuando es posible para aprovechar mejor los recursos. Esto se logra empleando los módulos XBee serie 2 con antena de cable para la comunicación inalámbrica y el sistema de control es una Raspberry Pi, capaz de almacenar en tablas de una base de datos y conectarse a Internet para descargar las instrucciones a transmitir.

B. OBJETIVOS

1. Objetivo general

Permitir una comunicación efectiva entre el módulo de detección de tránsito vehicular y el sistema de almacenamiento de datos, en el cual se archiva adecuadamente toda la información recibida, así como la recepción y transmisión correcta de datos al dispositivo portátil.

2. Objetivos específicos

- Determinar el medio y tecnología que mejor se adecúe a las necesidades/requerimientos del proyecto en cuanto a la forma de comunicación.

- Recibir de forma inalámbrica la información enviada por el módulo de detección de tránsito vehicular.

- Almacenar en una base de datos local la información proveniente del módulo de detección de tránsito vehicular y que se encuentre disponible para cualquier sistema que desee operar dichos datos.

- Implementar un protocolo de comunicación que permita la transmisión y recepción de datos de un dispositivo portátil hacia el módulo de almacenamiento.

C. JUSTIFICACIÓN

El módulo de comunicación y almacenamiento local es el encargado del control de flujo de información entre las diferentes partes que componen el proyecto. La información transmitida por el sensor debe ser decodificada y almacenada correctamente para su posterior procesamiento, y debe existir un protocolo de comunicación con el dispositivo portátil que permita a este último recibir las recomendaciones así como enviar notificaciones.

Al tener los datos medidos por los sensores disponibles para su uso inmediato, se tiene acceso a información prácticamente en tiempo real, lo que permite al policía municipal de tránsito evitar congestionamientos o accidentes de tránsito. Además, el módulo permite tener un récord del tránsito para su posterior análisis y determinar en qué horas y fechas específicas hay mayor o menor flujo vehicular. A partir de ello se pueden sacar conclusiones provechas para beneficio del tránsito vehicular.

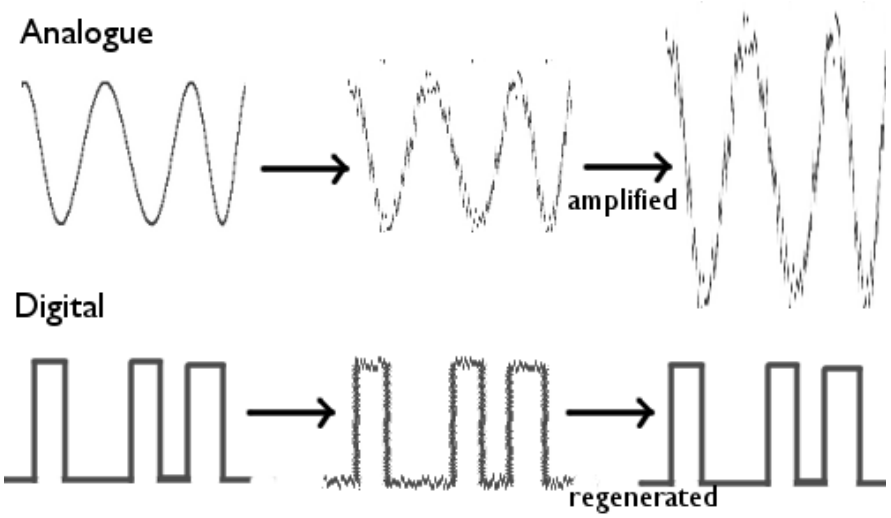
Por otro lado, el tener un protocolo de comunicación específico con el dispositivo portátil, permite que cualquier sistema de cómputo que se use para análisis solo debe tener en cuenta la información y formato en que debe transmitir sus resultados, haciendo más fácil la tarea de interconexión de elementos. La lista de instrucciones y comandos que espera recibir el dispositivo portátil, así como las notificaciones que esta herramienta puede enviar, ya está predeterminada por el módulo de comunicación, por lo que el operario del sistema de cómputo no debe preocuparse por más que realizar bien su análisis y codificar sus datos como es solicitado por el módulo en cuestión.

D. MARCO TEÓRICO

1. Sistema de comunicación. Un sistema de comunicación es un sistema que transmite información de su fuente a algún destino que se encuentra a cierta distancia de la fuente. Un mensaje es la manifestación física de la información según es producido por la fuente. Cualquiera que sea la forma del mensaje, el objetivo del sistema de comunicación es reproducir en el receptor una réplica aceptable del mensaje que emitió la fuente (Carlson, Crilly, & Rutledge, 2002).

Los mensajes se pueden clasificar en analógicos o digitales. Un mensaje analógico es una cantidad física que varía con el tiempo, usualmente de una forma suave y continua. Un sistema de comunicación analógico debe entregar la señal con un grado de fidelidad determinado. Un mensaje digital es una secuencia ordenada de símbolos seleccionados de un conjunto finito de elementos discretos. Un sistema de comunicación digital debe entregar estos símbolos con un nivel de precisión en un tiempo determinado (Carlson, Crilly, & Rutledge, 2002).

Figura 53.Ejemplos de mensajes analógicos y digitales.

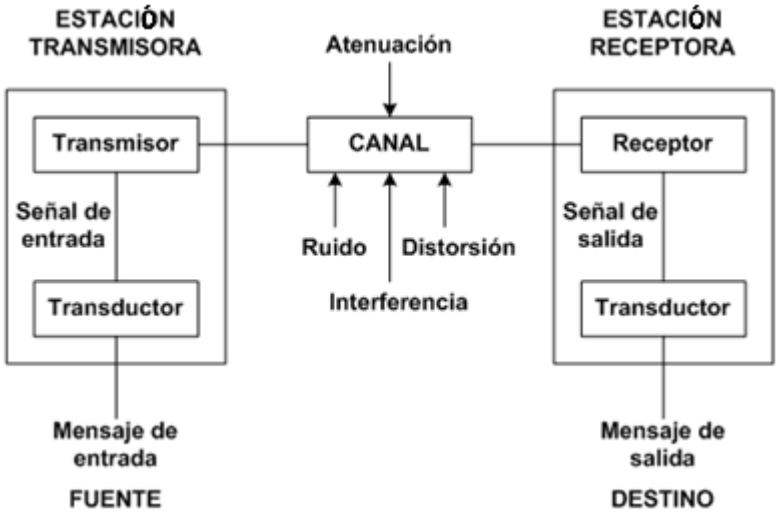


(Science aid, s.f.)

La mayoría de sistemas de comunicación poseen transductores para recibir o entregar un mensaje. El transductor en la fuente convierte el mensaje en una señal eléctrica (voltaje o corriente), mientras que el transductor en el receptor convierte la señal recibida a una forma (cantidad física) deseada (Carlson, Crilly, & Rutledge, 2002).

Los tres elementos esenciales en un sistema de comunicación son el transmisor, el canal y el receptor. El primero procesa la señal de entrada y produce una señal adecuada para las características del canal. El procesamiento generalmente involucra modulación y codificación. El segundo elemento es el medio que une la fuente con el destino, siendo un par de cables o una onda de radio. Cada canal introduce cierta cantidad de distorsión, ruido e interferencia provocando pérdidas o atenuación. El receptor opera en la señal de salida del canal preparándola para entregarla al transductor en el destino. Estas operaciones incluyen amplificación, demodulación, decodificación y filtrado (Carlson, Crilly, & Rutledge, 2002).

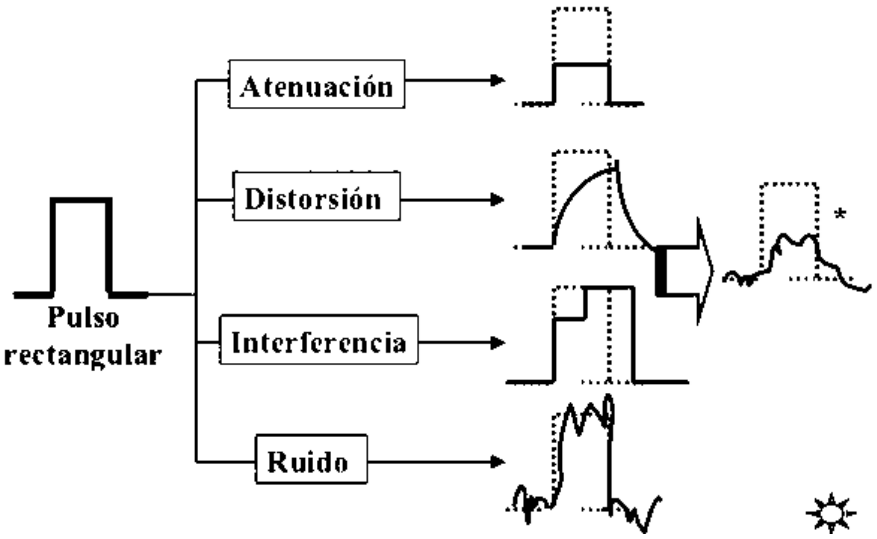
Figura 54. Elementos de un sistema de comunicación.



(Torres, s.f.)

La distorsión es la perturbación de la señal debida a la respuesta imperfecta del sistema a la señal misma. A diferencia del ruido y la interferencia, la distorsión desaparece cuando la señal se apaga. Este fenómeno se puede corregir (o reducir) con ecualizadores cuando el canal tiene una respuesta lineal. La interferencia es contaminación debida a señales extrañas de fuentes humanas. Ocurre principalmente en los sistemas de radio, en los cuales las antenas receptoras interceptan múltiples señales al mismo tiempo. El filtrado adecuado elimina este fenómeno. En cambio, el ruido se refiere a señales eléctricas aleatorias e impredecibles producidas por procesos naturales dentro y fuera del sistema. Esto puede corromper el mensaje parcial o totalmente. El filtrado reduce el ruido, pero no puede eliminarlo. Entonces, el ruido es una limitación fundamental en los sistemas de comunicación (Carlson, Crilly, & Rutledge, 2002).

Figura 55. Efectos del canal sobre una señal.



(Acurio, s.f.)

Otras limitaciones en el diseño de sistemas de comunicación incluyen los problemas tecnológicos y las limitaciones físicas. En el primer caso, se incluye las consideraciones de hardware, factores económicos, regulaciones federales, entre otros. Por el otro lado, el segundo caso se refiere a las leyes de la naturaleza, las cuales limitan el ancho de banda, que se traduce a velocidad de transmisión. Cuando una señal cambia rápidamente en el tiempo, su contenido frecuencial (también conocido como espectro), se extiende en un amplio rango. Asimismo, la capacidad del sistema de seguir los cambios de la señal es reflejada en su ancho de banda. Todos los sistemas eléctricos poseen elementos que almacenan energía, y esta energía no puede cambiar instantáneamente. Entonces, todo sistema de comunicación posee un ancho de banda finito que limita la tasa de variación de la señal (Carlson, Crilly, & Rutledge, 2002).

Según Digi International, en la comunicación inalámbrica, los componentes principales del sistema incluyen el elemento transmisor, el dispositivo receptor, el entorno en el que ocurre la comunicación y las antenas. El transmisor se encarga de entregar una señal a la antena para su transmisión. Un transmisor de radio, por ejemplo, codifica los datos en ondas de radio con cierto nivel de potencia. El receptor recibe y decodifica los datos que provienen de la antena receptora, y ejecuta la tarea de aceptar y decodificar las señales deseadas mientras rechaza las demás. El espacio entre los dos componentes anteriores es el entorno del sistema. Obstrucciones físicas y ruido pueden entrar en el entorno y limitan la capacidad de transmitir información. Algunos elementos simples (como paredes o personas) son capaces de limitar el rango de la transmisión.

2. Medios de transmisión. Los medios de transmisión son las autopistas y arterias que proveen una ruta de conexión para los dispositivos de telecomunicaciones. Estos también presentan una barrera a la señal de comunicación. Todo medio requiere de una infraestructura de soporte.

a. Cable de cobre. Las propiedades eléctricas del cable de cobre generan resistencia (introduce un retardo en la señal) e interferencia. Estas características limitan la velocidad de transmisión y distancia. Sin embargo, el costo, facilidad de manufactura, capacidad de reducirse en hebras finas, entre otros, lo hacen una buena selección. Se utiliza principalmente en señales de comunicación DC. Cualquier otra señal de corriente cerca del cable introduce interferencia y ruido (U.S. Department of transportation, Federal Highway Administration, s.f.).

1) Par trenzado. Está compuesto por dos cables de cobre aislados y trenzados. Esto último es realizado para evitar que corrientes eléctricas opuestas viajando a través de los cables individuales interfieran la una con la otra. Esta es la base de la mayoría de las tecnologías y servicios de telecomunicaciones. Para algunas aplicaciones, el par trenzado se encuentra dentro de un blindaje que funciona como tierra. Este es conocido como par trenzado apantallado o blindado (STP). El cable ordinario hacia las casas es no blindado (UTP). Existe un estándar (EIA/TIA) para el código de colores de los cables, los pares de hilos y los grupos de cables. El par trenzado es categorizado por el número de vueltas por metro (Cuadro). Un número mayor de vueltas provee mayor protección contra la interferencia, resultando en una mejor calidad de la transmisión (U.S. Department of transportation, Federal Highway Administration, s.f.).

Cuadro 4. Categorías del par trenzado

Categoría	Máxima velocidad de transmisión	Aplicación usual
CAT 1	Menos de 1 Mbps	Voz analógica, cableado de timbres
CAT 2	4 Mbps	Redes Token Ring
CAT 3	16 Mbps	Voz y datos, Ethernet 10-BaseT, servicios básicos de telefonía
CAT 4	20 Mbps	Redes Token Ring de 16 Mbps
CAT 5	100 Mbps hasta 1 Gbps	10Base-T, 100Base-T, GigE, FDDI, 155 Mbps ATM
CAT 5E	100 Mbps	FDDI, ATM
CAT 6	Más de 100 Mbps	Aplicaciones de banda ancha
CAT 7	-	GigE plus

(U.S. Department of transportation, Federal Highway Administration, s.f.)

2) Coaxial. Está formado por un núcleo de cobre (que transporta la señal) rodeado de una capa de aislante, seguida de una lámina de metal trenzado con una cubierta exterior. Esta última funciona como blindaje o aterrizaje. Varios de estos cables pueden colocarse en un conducto y, usando repetidoras, transportar información largas distancias. También existe el cable triaxial, el cual posee un conductor central y dos blindajes. Esta configuración permite alcanzar una mayor distancia con menos pérdidas debido a la interferencia de señales eléctricas externas. Inicialmente, el cable coaxial era utilizado por departamentos de tránsito para proveer comunicación entre controladores en el campo y el controlador central en sistemas de tránsito automatizado (U.S. Department of transportation, Federal Highway Administration, s.f.).

b. Fibra óptica. Se refiere al medio y a la tecnología asociada con la transmisión de información por medio de impulsos de luz a lo largo de una hebra de vidrio. Un hilo de fibra óptica transmite mucha más información que el cable de cobre convencional y es menos propenso a la interferencia electromagnética. La transmisión por fibra óptica requiere repetidoras en intervalos mayores que los sistemas basados en cobre. El núcleo de vidrio es cubierto con un material refractivo (conocido como "cladding") que resulta el paso de luz controlado a través del núcleo. La siguiente capa es una protección contra el daño externo y evita que la luz se escape. Esta última posee un código de color para propósitos de identificación (U.S. Department of transportation, Federal Highway Administration, s.f.).

1) Multimodo. Diseñado para transportar múltiples rayos de luz o modos concurrentemente, cada uno a un ángulo de reflexión ligeramente distinto dentro del núcleo. Es utilizado en distancias relativamente cortas (menos de 15 mil pies). Generalmente es iluminado con LED.

2) Monomodo. Diseñado transmitir un solo rayo o modo de luz. Es producido en diversas variaciones, diseñadas para facilitar distancias muy largas, y la transmisión de múltiples frecuencias de luz en un solo rayo. El más utilizado de todo propósito es el SMF-28. El control de señalización de tráfico es un ejemplo de sistemas que lo utilizan.

Cuadro 5. Comparación entre fibra monomodo y multimodo

Característica	Fibra monomodo	Fibra multimodo
Ancho de banda	Virtualmente ilimitado	Menos que virtualmente ilimitado
Calidad de señal	Excelente a través de largas distancias	Excelente a través de cortas distancias
Atenuación primaria	Dispersión cromática	Dispersión modal
Tipos de fibras	Índice de paso y dispersión desplazada	Paso e índice gradual
Aplicación típica	Casi cualquier cosa (incluyendo Ethernet)	Video analógico, Ethernet, Comunicaciones de corto alcance

(U.S. Department of transportation, Federal Highway Administration, s.f.)

c. Medio inalámbrico

1) Sistemas de celulares. Los sistemas analógicos utilizan dos estándares: AMPS (Sistema de telefonía móvil avanzada) y GSM (sistema global para comunicación móvil). No son compatibles entre sí (U.S. Department of transportation, Federal Highway Administration, s.f.).

2) Sistemas de radio punto-a-punto. Son sistemas de radio que se comunican entre dos localidades fijas. Pueden ser establecidas con cualquier frecuencia. Sin embargo, la mayoría de sistemas son desarrollados usando frecuencias en el espectro de la microonda, entre 800 MHz y 30 GHz. La Comisión Federal de Comunicaciones ha designado grupos de frecuencias en el espectro de radio para servicios fijos (U.S. Department of transportation, Federal Highway Administration, s.f.).

a) Microonda. Provee conectividad entre los principales nodos de comunicación. Las frecuencias para este servicio van de 6 GHz a 11 GHz. Los sistemas pueden ser diseñados para operar a distancias de 20 millas entre dos puntos. Los dispositivos operando a 900 MHz, 2 GHz y 23 GHz no requieren licencia.

b) Radio de espectro disperso. Se refiere a una técnica de modulación RF que dispersa la señal transmitida a través de un amplio espectro. El transmisor típico integra la señal actual con una secuencia de bits que codifican y dispersan la señal a través del ancho de banda usualmente de 20 MHz a 30 MHz. La señal transmitida es diluida en un amplio ancho de banda, lo que minimiza la potencia a cualquier frecuencia. El resultado es una señal que se encuentra debajo del límite de ruido de los receptores de banda angosta, pero dentro de los márgenes de recepción de un receptor de espectro disperso. Cada transmisor y receptor es programado con una secuencia de dispersión que son utilizados para demodular la señal, eliminando el ruido. Muchos sistemas de gestión de tránsito utilizan este sistema (U.S. Department of transportation, Federal Highway Administration, s.f.).

3) Radio de doble vía. Las frecuencias más empleadas son las de 30, 150, 450-512 y 800 MHz. La cobertura suele expresarse en términos de millas de aire. Los sistemas en la banda de 150 MHz pueden cubrir desde 15 hasta 30 millas de aire en el radio de un transmisor (U.S. Department of transportation, Federal Highway Administration, s.f.).

4) Wi-Fi (Wireless fidelity). Se refiere a un sistema de comunicación punto-a-multipunto. Utiliza el espectro de frecuencias de 900 MHz, 2 GHz y 5 GHz (U.S. Department of transportation, Federal Highway Administration, s.f.).

5) WLAN. El estándar 802.11B ha hecho de WLAN mucho más rápido y fácil. Una WLAN puede tener un alance de 300 m en el exterior. WLAN requiere un punto de acceso cableado al que se conectan todos los dispositivos inalámbricos. El estándar 802.11A transfiere datos a velocidades de 54 Mbps en la banda de 5 GHz; el 802.11B alcanza velocidades de 11 Mbps en la banda de 2.4 GHz; el 802.11G transfiere a 54 Mbps en la banda de 2.4 GHz, y es compatible con el anterior. Una desventaja es la falta de seguridad y la sobrepoblación en la banda (U.S. Department of transportation, Federal Highway Administration, s.f.).

d. Óptica de espacio libre. Se refiere a sistemas inalámbricos que utilizan transmisión por medio de LASER a través del aire entre dos puntos. Estos sistemas están limitados a un rango de 3 millas de aire (U.S. Department of transportation, Federal Highway Administration, s.f.).

Cuadro 6. Comparación entre medios de transmisión alámbricos

Configuración	Velocidad máxima de transmisión (Mbps)	Estándar	Distancia máxima	Medio de transmisión	Precio aproximado
10BASE-T	10	IEEE 802.3	100 m	Dos pares trenzados CAT 3 o mejor	$0.09/ft
100BASE-TX	100	IEEE 802.3u	100 m	Dos pares trenzados UTP o mejor	
1000BASE-T	1000	IEEE 802.3ab	100 m	Cuatro pares trenzados CAT 5E o mejor	$0.32/ft
10BASE5	10		500 m/segmento	Coaxial grueso (RG-8/U)	
10BASE2	10		180 m/segmento	Coaxial delgado (RG-58/U)	$0.32/ft
1000BASE-SX	1000	IEEE 802.3z	550 m / 275 m	Fibra óptica 50-micron multimodo / 62.5-micron multimodo	
1000BASE-LX	1000	IEEE 802.3z	550 m	Fibra óptica 50-micron multimodo / 62.5-micron multimodo	
100BASE-FX	100		400 m	Fibra óptica 62.5-micron multimodo / 125-micron multimodo	
1000BASE-LX	1000		10 km	Fibra óptica 9-micron monomodo	

(Computer Networking, s.f.) (Cisco, s.f.)

Cuadro 7. Pérdidas en el espacio libre

Distancia (km)	Atenuación (dB)		
	900 MHz	2.4 GHz	5.8 GHz
2.5	99	108	116
5	106	114	122
8	110	118	126
10	112	120	128
16	116	124	132

(Poulsen, s.f.)

Cuadro 8. Comparación de módulos RF

Módulo RF	Distancia máxima en línea de vista	Rango de frecuencias	Potencia de transmisión	Velocidad de transmisión (Kbps)	Topologías de red	Precio aproxi-mado ($)
Xtend	64 km con antena de alta ganancia; 22 km con antena dipolo de 2.1 dB	902 – 928 MHz	1 mW – 1 W	10 o 125	Punto-a-punto, punto-a-multipunto, repetidor, mesh	194.95
Xstream	16 km con antena de alta ganancia; 5 km con antena dipolo de 2.1 dB	2.4 – 2.4385 GHz	50 mW	10 o 20	Peer-to-peer, punto-a-punto, punto-a-multipunto, multidrop	
XBee-PRO XSC	45 km con antena de alta ganancia; 14 km con antena dipolo de 2.1 dB	902 – 928 MHz	100 mW o 250 mW	10 o 20		66.95
XBee-PRO 900 HP	15.5 km con antena dipolo de 2.1 dB	902 – 928 MHz	250 mW	10 o 200	DigiMesh, repeditor, punto-a-punto, punto-a-multipunto, peer-to-peer	54.95
XBee ZB / ZBee- PRO ZB	120 m / 3.2 km o 1.5 km	2.4 GHz	1.25 mW o 2 mW / 63 mW o 10 mW	250	Mesh	299
XBee / XBee-PRO 802.15.4	100 m / 1.6 km	2.4 GHz	1 mW / 60 mW	250	Multipunto	129
XBee / XBee-PRO ZB SMT	1.2 km / 3.2 km	2.4 GHz	3.1 mW / 63 mW	250	Mesh	299
XBee / XBee-PRO DigiMesh 2.4	90 m / 1.6 km	2.4 GHz	1 mW / 63 mW	250	DigiMesh, mesh	269
nRF24L01	1.8 km con antena de alta ganancia	2.4 GHz		250 a 2000	Punto-a-multipunto	18.5

(Sparkfun, s.f.)

3. Topologías de red. El arreglo de los dispositivos en una red es denominado topología física.

a. Punto-a-punto. Es una conexión directa entre dos nodos, resultando un intercambio de datos únicamente entre estos dos elementos (Rankin, s.f.).

b. Punto-a-multipunto. Está compuesta por un nodo central que se conecta a varios nodos. De esta se derivan algunas topologías como la estrella (IT Law Wiki, s.f.).

c. Topología en estrella. Posee un punto de conexión central al que se conectan todos los dispositivos por cable o se encarga de las transmisiones inalámbricas a todos los periféricos y estaciones de trabajo. La ventaja de esta topología es que cualquier enlace puede fallar sin afectar al resto de la red. Su principal desventaja es que requiere mucho cable para conectar todos los dispositivos, lo que no ocurre en las redes inalámbricas (Bird & Hardwood, 2003).

d. Topología mesh. Conecta cada dispositivo de una red con los otros dispositivos de la misma red. Los datos que viajan en esta red pueden seguir un de varios caminos posibles de la fuente al destino. Estas rutas redundantes hacen de la red mesh robusta. Aunque varios enlaces fallen, los datos pueden seguirse transmitiendo por otros enlaces. En la actualidad, esta topología se utiliza en redes inalámbricas. En estas redes, los datos son transportados a nodos situados lejos del punto de acceso central simplemente saltando de un nodo a otro (Bird & Hardwood, 2003).

Una verdadera topología mesh utiliza una conexión punto-a-punto entre todos los dispositivos. La ventaja de esto es que existen múltiples enlaces que proveen tolerancia a fallas y redundancia. La desventaja es la dificultad para implementarlo. Por otro lado, una topología mesh híbrida utiliza conexión punto-a-punto entre algunos dispositivos. La ventaja de esto es que la red puede expandirse con poca alteración, pero requiere de hardware y cable especializado, lo que lo hace más costoso.

1) Módulos Zigbee. Zigbee es un estándar para dispositivos de baja potencia y corto alcance basados en el estándar IEEE 802 para red de área personal (PAN). Los módulos Zigbee trabajan en la banda ISM (Industrial, Científico y Médico). Una red Zigbee consiste de tres tipos de dispositivos Zigbee. Todos los dispositivos en una de estas redes tienen un identificador PAN asignado. Estos dispositivos pueden ser un coordinador (raíz de la red y puede unir a otras redes, almacena información de la red), un router (pasa los datos de otros dispositivos, puede operar cuando se conecta a un coordinador), o un dispositivo terminal (solo puede hablar con el nodo padre, debe unirse a una PAN antes de transmitir datos). Cada dispositivo consta de una dirección de red de 16 bits y una dirección de dispositivo de 64 bits. En los módulos XBee ZB, esta última dirección es un número único y permanente asignado por el fabricante. Cada dato transmitido por un dispositivo Zigbee es enviado especificando ambas direcciones (Seeed Studio, s.f.).

2) Digimesh. El firmware DigiMesh permite la auto sanación de la red mesh con poca configuración del usuario. En una red DigiMesh, todos los nodos pueden hablar con todos los otros nodos de la misma red y que se encuentren en su alcance. No hay una ruta definida por el usuario para que fluyan los datos. La ventaja es que los nodos eligen el mejor camino entre ellos y el destinatario, y si algún nodo abandona la red, los datos son automáticamente redireccionados. Otra ventaja es que se sincronizan para desactivarse y activarse según sea necesario (Byford, s.f.).

4. Tecnologías inalámbricas. Una red inalámbrica transporta datos de un dispositivo a otro sin el uso de cables o alambres. Redes de cualquier tamaño pueden usar tecnologías inalámbricas, como señales de radio, microondas y luz infrarroja. La mayoría de las redes inalámbricas transporta datos con señales de radio frecuencia (RF). Estas son enviadas y recibidas por un transceptor (combinación de transmisor y receptor) que está equipado con una antena. Las microondas y luz infrarroja trabajan mejor para transmisiones en línea de vista.

La principal ventaja de las redes inalámbricas es su movilidad. No existen cables y es menos probable que ocurran picos de potencia en las estaciones de trabajo. Las principales desventajas son la velocidad, rango, necesidad de licencia y seguridad.

a. Ehternet. Es la tecnología líder en las redes de área local (LAN). Las redes construidas con Ethernet son fáciles de entender, implementar, manejar y mantener. Como una tecnología no propietaria, su equipo es fabricado por diferentes empresas, por lo mantiene los precios bajos. Los estándares actuales permiten gran flexibilidad en topologías de red, cubriendo las necesidades de pequeñas y grandes instalaciones. Ethernet es compatible con redes Wi-Fi, por lo que permite combinar dispositivos alámbricos con dispositivos inalámbricos en una sola red. Ethernet transmite paquetes de datos a todos los dispositivos en la red; el paquete solo es aceptado por el dispositivo al que está dirigido (Parsons & Oja, 2011). Además, ofrece diferentes velocidades de transmisión (Cuadro 9).

Cuadro 9. Estándares de Ethernet

Estándar	Designación IEEE	Velocidad
10BASE-T Ethernet	IEEE 802.3	10 Mbps
Fast Ethernet	IEEE 802.3u	100 Mbps
Gigabit Ethernet	IEEE 802.3z	1 Gbps
10 Gigabit Ethernet	IEEE 802.3ae	10 Gbps
40/100 Gigabit Ethernet	IEEE 802.3ba	40 o 100 Gbps

(Parsons & Oja, 2011)

El equipo de Ethernet consta de dos o más computadoras con puerto Ethernet, un router para Ethernet, un UPS (o cualquier dispositivo de protección), cables para conectar las computadoras y un dispositivo de acceso a internet (Parsons & Oja, 2011).

1) 802.11B: Wireless. Este estándar define el uso de Ethernet inalámbrico para redes LAN. El más común tiene un punto de acceso que permite a diversos dispositivos inalámbricos comunicarse unos con otros. Estos pueden estar conectados a redes alámbricas, creando porciones inalámbricas. Los puntos de acceso suelen cubrir hasta cientos de pies, pero su alcance actual depende de las condiciones locales y del receptor. La velocidad máxima de transmisión es de 11 Mbps, y los dispositivos están diseñados para ser compatibles con estándares anteriores, proveyendo velocidades de 1, 2 y 5.5 Mbps. Utiliza la banda de 2.4 GHz (Bird & Hardwood, 2003).

b. GSM. El sistema global para comunicación móvil es la tecnología celular más utilizada. Fue diseñada como una segunda generación de tecnología celular. Utiliza TDMA digital (time división multiple access). Al adoptar esta técnica, más usuarios son acomodados en el ancho de banda disponible. Se adoptó el cifrado de voz codificada digitalmente para mantener la privacidad. Una gran variedad de servicios de datos son soportados con velocidades de transmisión de 9.6 Kbps. Uno de los servicios que más auge ha tenido es el servicio de mensajes cortos (Poole, GSM: global system for mobile communications tutorial, s.f.).

Esta tecnología utiliza canales de radio de 200 kHz. Estos son multiplexados en el tiempo para permitir hasta ocho usuarios acceder a cada canal. Utiliza la banda 890 – 915 MHz para subida de datos, y la banda 933 – 960 MHz para descarga de datos. Se basa en la modulación GMSK (Poole, GSM: global system for mobile communications tutorial, s.f.).

Las estaciones transceptores base son organizadas en pequeños grupos, controlados por una estación base controladora. Este conjunto se denomina subsistema estación base. En el núcleo de la red se encuentra el área de switcheo principal, conocido como el centro de switcheo móvil. Asociado a él se tienen los registros de ubicación que rastrean la localización de los móviles y permiten que llamadas sean ruteadas a ellos. Adicionalmente, existe un centro de autenticación y un registro de identificación de equipo, que se utilizan en la autenticación del móvil antes de que acceda a la red. Por último, se tiene el móvil. Este es el elemento con el que el usuario tiene contacto. Con GSM se desarrolló la SIM (módulo de identidad del suscriptor) (Poole, GSM: global system for mobile communications tutorial, s.f.).

Las ventajas son menor deterioro de la señal dentro de edificios, la capacidad de usar repetidoras, tiempo de conversación mayor, la disponibilidad de SIM permite a usuarios cambiarse de red, y la cobertura global. Las desventajas son que interfiere con algunos electrónicos, como amplificadores de audio; y la propiedad intelectual se concentra en pocas industrias creando barreras a nuevos entrantes y limitando la competencia entre fabricantes de celulares (Poole, GSM: global system for mobile communications tutorial, s.f.).

c. GPRS. El servicio general de paquetes de radio es un servicio de telefonía móvil, que permite a los usuarios conectarse a internet a través de su teléfono. Esta tecnología utiliza las porciones del ancho de banda de GSM que se encuentran sin uso. No hay nivel de calidad garantizado; opera en las mejores

condiciones que puede. Los canales son de 200 kHz, utiliza modulación GMSK y la máxima velocidad de transmisión es de 172 Kbps. La red estándar GSM tuvo que ser modificada para poder transmitir paquetes de datos (Poole, GPRS general packet radio service tutorial, s.f.).

Las ventajas son la velocidad, la operación en base a paquetes, la conectividad permanente, flexibilidad y eficiencia de recursos, y permite futuras extensiones. La desventaja es que como utiliza la red GSM, no se pueden realizar varias funciones de red simultáneamente (Poole, GPRS general packet radio service tutorial, s.f.).

5. Libelium. Libelium es un proveedor de una plataforma para el internet de las cosas (IoT por sus siglas en inglés). Se trata de una industria enfocada en ciudades inteligentes y soluciones M2M, ofreciendo una plataforma de sensores de código abierto para el IoT. Sus sensores de denominan Waspmote, y el router lo llaman Meshlium. Ambos son dispositivos poderosos que se utilizan actualmente en algunas ciudades europeas para desarrollar redes inalámbricas de sensores (Libelium, s.f.).

Meshlium contiene seis interfaces para comunicación inalámbrica: WiFi 2.4 GHz, WiFi 5 GHz, 3G/GPRS, Bluetooth, XBee y LoRa. También puede integrar un módulo GPS para aplicaciones móviles. Posee un disco duro de 8 GB. Cada modelo está diseñado para aplicaciones específicas. Por otro lado, Wapsmote es un sensor orientado a desarrolladores. Trabaja con múltiples protocolos (ZigBee, Blutooth, GPRS) y frecuencias (2.4 GHz, 868 MHz, 900 MHz) siendo capaz de tener enlaces de hasta 22 km. El precio de Waspmote se encuentra alrededor de 150 euros, mientras que Meshlium se encuentra alrededor de 900 euros (Libelium, s.f.).

6. Comunicación RF. National Instruments define la comunicación por radio frecuencia (RF) como un tipo de comunicación inalámbrica que abarca la región del espectro de frecuencias entre unos cuantos kilohertz hasta aproximadamente un gigahertz.

Digi International explica que la comunicación por radio frecuencia se basa en las leyes de la física que describen el comportamiento de la energía en ondas electromagnéticas. Se requiere una fuente y un receptor capaz de percibir estas ondas, las cuales se propagan a través del aire a una velocidad cercana a la de la luz. La longitud de onda de estas señales es inversamente proporcional a la frecuencia. La frecuencia se mide en Hertz (ciclos por segundo) y las radio frecuencias se miden en kilohertz (miles de ciclos por segundo), megahertz (millones de ciclos por segundo) y gigahertz (miles de millones de ciclos por segundo).

En general, las señales con una mayor longitud de onda viajan mayores distancias, y penetran o rodean los objetos mejor que aquellas con menor longitud de onda (Digi International, s.f.).

Se puede resumir la transmisión de señales por radio frecuencia como describe Digi International: se tiene un transmisor RF que hace vibrar a un electrón en un punto, causando un efecto “dominó” en la cual los

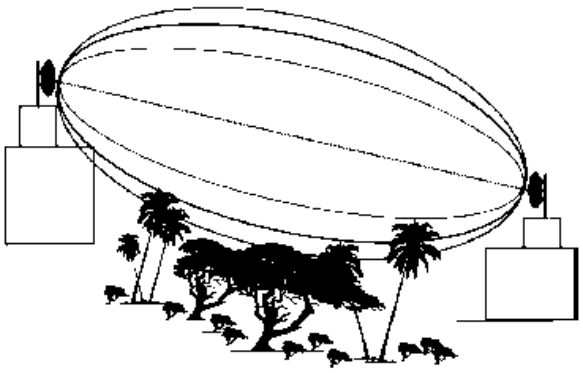
electrones a su alrededor comienzan a vibrar, lo que resulta en el medio de propagación de la onda electromagnética. El receptor detecta estas vibraciones, las cuales se pueden realizar por medio de patrones; estos son utilizados por los sistemas de comunicación RF para representar información.

En la mayoría de los sistemas inalámbricos, según Digi International, se tienen dos limitantes: se debe operar a cierta distancia (rango) y se debe transferir una cantidad de información en un tiempo específico (data rate). Además, se deben considerar las regulaciones y licencias de cada país. Sírvase de ejemplo la Ley General de Telecomunicaciones, Decreto 94 – 96 del Congreso de la República de Guatemala, Capítulos III al V, en la cual se restringen las bandas de frecuencias según la clasificación a partir de su uso (Artículo 51) (Superintendencia de Telecomunicaciones).

El rango depende de dos parámetros: potencia de transmisión y la sensibilidad del receptor. El primero se refiere a la cantidad de potencia irradiada por la antena, mientras que el segundo se refiere al nivel mínimo de señal que el dispositivo puede demodular. La atenuación total que se puede tener entre el transmisor y el receptor para que haya comunicación se denomina "link budget" (Digi International, s.f.).

Para obtener una comunicación inalámbrica de larga distancia, se debe lograr una línea de vista directa (LOS por sus siglas en inglés) entre la antena emisora y la antena receptora. Esta puede ser de dos tipos. La primera se refiere a la línea visual, lo que implica un paso en línea recta entre los dos puntos. La segunda es la línea RF, que requiere tanto la línea visual como un paso con forma de balón de fútbol americano libre de obstáculos. Esto último es lo que se conoce como la zona de Fresnel. La altura a la que se deben colocar las antenas se determina por el diámetro de la zona de Fresnel, el cual depende de la frecuencia y la distancia entre los dispositivos (Digi International, s.f.).

Figura 56. Ejemplo de la zona de Fresnel.

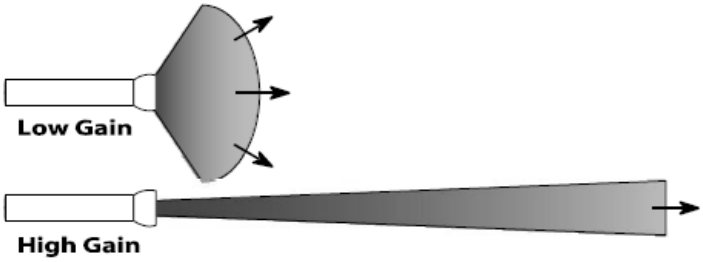


(Digi International, s.f.)

Según Digi International, las antenas son dispositivos que concentran energía en una dirección particular. Pueden proveer diferentes patrones de radiación dependiendo de su diseño y aplicación. La ganancia

determina la cantidad de energía que es capaz de concentrar en determinada dirección. Se obtiene mayor distancia cuando el patrón es angosto.

Figura 57. Dispersión de energía en antenas de alta y baja ganancia.



(Digi International, s.f.)

Las antenas transmisoras y receptoras se utilizan para concentrar y direccionar ondas de radio en direcciones específicas. Es un componente ajustable que impacta directamente en la distancia que puede viajar la información en un sistema de comunicación inalámbrico. La ganancia describe la cantidad de concentración que la antena es capaz de aplicar en el sistema, y se ajusta para incrementar el rango. Las antenas omni-direccionales concentran la energía equitativamente en una especie de dona alrededor de la antena. En cambio, las antenas direccionales concentran la energía en una sola dirección, y son las antenas de alta ganancia (Digi International, s.f.).

Algunas características importantes de la comunicación RF son la baja potencia de transmisión en comparación con otros sistemas, rango de operación grande (de 3 a 30 metros para dispositivos pequeños), soporta velocidades de hasta 1 – 2 Mbps, atraviesa muros, y no requiere una vía directa de transmisor hacia receptor (Ton, s.f.).

7. Telemetría/GPRS. Telemetría es un sistema de medición de magnitudes físicas que permite transmitir los datos a un lugar lejano. Es una técnica automatizada de las comunicaciones que utiliza comúnmente transmisión inalámbrica. Consiste en un transductor como un dispositivo de entrada, un medio de transmisión, dispositivos de procesamiento de señales o dispositivos de grabación o visualización. La cantidad física medida es convertida a una señal eléctrica por medio del transductor, y los datos son enviados a cierta distancia para su registro (Innova Technologies, s.f.).

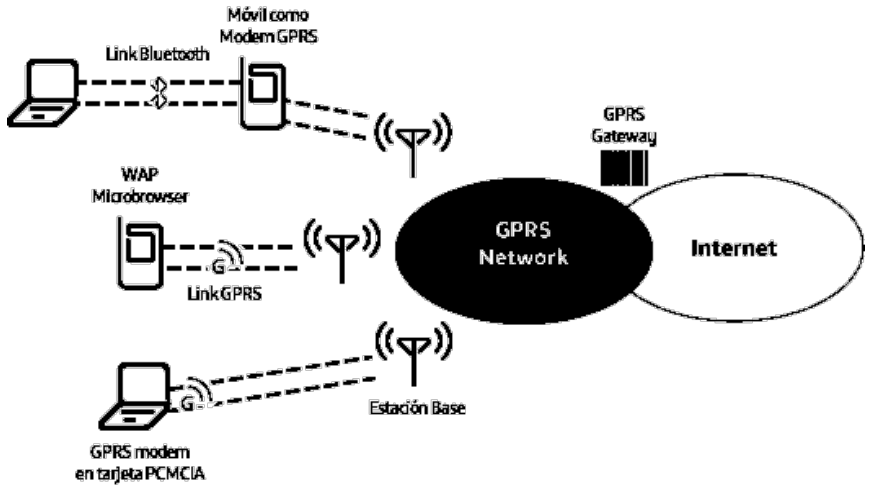
El sistema GSM es el sistema de comunicación de móviles digital de segunda generación basado en células de radio. Fue diseñado para la transmisión de voz y se basa en la conmutación de circuitos. La red GSM se compone de la estación móvil (compuesta por el equipo móvil y la SIM – Subscriber Identity Module), estación base (encargados de transmisión y recepción, conecta las estaciones móviles con los NSS), el subsistema de conmutación y red (NSS – Network and Switching Subsystem, administra las comunicaciones)

y subsistemas de soporte y operación (controlar y monitorear la red). Ofrece una velocidad de transferencia de 9.6 kbps (Linares, s.f.).

El estándar GPRS (General Packet Radio Service) es un estándar de telefonía de segunda generación que permitió la transición hacia la tercera generación. Permite la transferencia de datos con una tasa teórica alrededor de 171.2 kbps, obteniendo 114 kbps en la práctica. Transfiere los datos en paquetes, por lo que solo usa la red cuando lo necesita. El usuario paga por volumen de datos y no por duración de la conexión. El estándar brinda servicio de punto-a-punto (modo cliente-servidor), servicio de punto-a-multipunto (enviar paquetes a varios destinatarios) y servicio de mensajes cortos (SMS) (CCM Benchmark Group, s.f.).

Según CCM Benchmark Group, la integración de GPRS a la arquitectura GSM requiere de nuevos nodos denominados GSN (nodos de soporte GPRS) los cuales son: el router SGSN (gestiona las direcciones de las terminales y proporciona la transferencia de la interfaz de paquetes con la pasarela GGSN) y la pasarela GGSN (permite la conexión con otras redes de datos, proporcionando una dirección IP a las terminales).

Figura 58. Red de comunicación con GPRS.



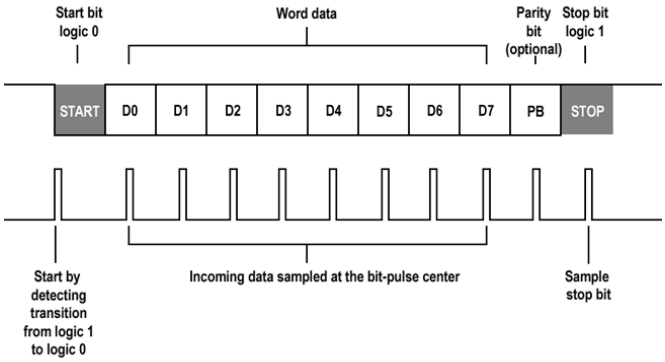
(Telefónica, s.f.)

8. USART. Texas Instruments define el Receptor Transmisor Síncrono/Asíncrono Universal (USART por sus siglas en inglés) como un módulo de comunicación serial que realiza la conversión de serial a paralelo cuando recibe de un dispositivo periférico y la conversión de paralelo a serial cuando recibe del CPU.

En la comunicación serial se envía un bit a la vez. Una secuencia de bits representa la información que se desea transmitir, codificada en niveles de voltaje específicos. Cada secuencia de bits debe llevar un bit de inicio y un bit de parada, para identificar el inicio y fin de la secuencia de datos (Traylor, s.f.).

La comunicación serial puede ser síncrona o asíncrona. En la primera, se requiere de un dispositivo con su propio reloj; cuando este desea comunicarse con otra unidad, envía un pulso de reloj junto con el bit de información, resultando en un intercambio de datos en cada vez que cambia el reloj. Esto se utiliza, por ejemplo, en comunicación con circuitos integrados simples (registros de memoria conectados con los recursos mínimos necesarios al circuito que controlan) que solo ejecutan una tarea, como el de leer un sensor, y que no poseen su propio reloj (Tisch School of the Arts). Cuando la comunicación es asíncrona, no existe reloj que sincronice los dispositivos, por lo que es necesario conocer la velocidad de transmisión (conocida como baud rate) y el bit de inicio. Con estos parámetros, el receptor es capaz de reconocer donde debe iniciar la toma de datos. La detección se puede realizar por medio de un bit adicional, denominado bit de paridad (Traylor, s.f.).

Figura 59. Trama de datos de UART.



(Electric Imp, s.f.)

9. CRC. La información se puede codificar basándose en la presencia o ausencia de una señal eléctrica, la cual está sujeta a distorsiones, especialmente cuando las distancias son grandes. Por esta razón, es necesario un método de verificación de autenticidad. Existen algunos mecanismos que garantizan un nivel de integridad de los datos, por ejemplo, el destinatario obtiene una confirmación de que los datos recibidos son idénticos a los que fueron transmitidos. Algunas formas de proteger la transferencia de datos incluyen instalar un medio de transmisión más seguro e implementar mecanismos lógicos para detectar y corregir errores (CCM Benchmark Group, s.f.).

«La mayoría de sistemas de control lógico de errores se basan en la suma de la información» (CCM Benchmark Group, s.f.) (lo que se denomina redundancia); esta información adicional se denomina suma de comprobación. Los sistemas de detección de errores se han perfeccionado por medio de los códigos de autocorrección y los códigos de autoverificación (CCM Benchmark Group, s.f.).

Según CCM Benchmark Group, algunos mecanismos de verificación son: verificación de paridad (agrega un bit adicional para indicar si el número de unos lógicos en la palabra es impar o par), verificación de

redundancia longitudinal (verifica la integridad del bit de paridad de un grupo de caracteres) y verificación de redundancia cíclica (CRC). Este último es el método de detección de errores utilizado en telecomunicaciones.

CCM Benchmark Group menciona que «la verificación de redundancia cíclica consiste en la protección de los datos en bloques» (tramas), al cual se le asigna un segmento de datos denominado código de control. El código CRC contiene datos redundantes con la trama, de manera que se pueda tanto detectar como solucionar los errores. La idea es observar cada trama como un gran número binario, el cual se divide entre cierto valor y el residuo es lo que se llama CRC. El algoritmo trata a las secuencias binarias como polinomios; la secuencia binaria corresponde a los coeficientes. Todas las expresiones se manipulan utilizando un módulo 2 (Bies, s.f.).

En este proceso de detección de errores, tanto en transmisor como el receptor conocer el polinomio generador. Ambos ejecutan un algoritmo, y si coinciden los valores, se concluye que los datos son válidos (CCM Benchmark Group, s.f.).

Existen diversos algoritmos CRC de 16 bits, entre los que se encuentra el CCITT, el cual utiliza el polinomio $x^{16} + x^{12} + x^6 + 1$ (x1021). Las variantes de CCITT se diferencian por el valor inicial que emplean; el llamado XMODEM tiene un valor inicial de cero, no realiza reflexión en la entrada ni en la salida, y tampoco ejecuta la operación XOR en la salida (Cook, s.f.).

10. Módulos de control

Cuadro 10. Características de procesador y RAM de dispositivos posibles para módulo de control.

Dispositivo	Procesador	Frecuencia CPU	RAM
BioDigitalPC I-Card	Intel Dual Core x86 / AMD Dual Core x86 /AMD Quad Core x86	1.6 GHz - 1.86 GHz / 1.0 GHz / 1.2 GHz - 2.0 GHz	2 GB / 4 GB / 8 GB DDR3
Parallella Board	ARM A9 dual core / Epiphany multi-core	600 MHz	1 GB DDR3
BeagleBone Black	AM335x ARM Cortex-A8	1 GHz	512 MB DDR3
Arduino Mega 2560	ATmega2560	16 MHz	8 KB SRAM
Arduino Mega ADK	ATmega2560	16 MHz	8 KB SRAM
Arduino Due	AT91SAM3X8E	84 MHz	96 KB SRAM
Raspberry Pi 2 Model B	ARM Cortex A-7	900 MHz	1 GB
Seeeduino Stalker	ATmega328	8 MHz	

(Arnouse Digital Devices Corp., s.f.) (Parallella, s.f.) (Beagleboard, s.f.) (Arduino, s.f.) (Raspberry Pi Foundation, s.f.) (Seeed Studio, s.f.)

Cuadro 11. Características de salidas y memoria de dispositivos posibles para módulo de control.

Dispositivos	Alimentación	Salidas	Almacenamiento	Precio
BioDigitalPC I-Card	12VDC	5 USB ports, Display port, 2 PCIex1, HD Audio	32 GB / 64 GB / 128 GB SSD Storage	
Parallella Board	5VDC	48 GPIO signal, HDMI port, A three-pin header for 3.3V UART output, USB 2.0	GB Ethernet, microSD storage	$99
BeagleBone Black	5VDC	USB, HDMI, 2x46 pin headers	4 GB 8-bit EMMC on-board flash storage, Ethernet	$55
Arduino Mega 2560	7-12VDC	54 Digital IO, regular USB, 4 UART	4 KB EEPROM, 256 KB Flash	$35
Arduino Mega ADK	7-12VDC	54 Digital IO, regular USB, 4 UART	4 KB EEPROM, 256 KB Flash	$49
Arduino Due	7-12VDC	54 Digital IO, 2 micro USB, 4 UART	512 KB Flash	$40
Raspberry Pi 2 Model B	5VDC	4 UBS ports, 40 GPIO pins, Ethernet port	Micro SD storage	$40
Seeeduino Stalker	Solar panel, Li-Po Bat	Bee Socket	Micro SD storage	$39

(Arnouse Digital Devices Corp., s.f.) (Parallella, s.f.) (Beagleboard, s.f.) (Arduino, s.f.) (Raspberry Pi Foundation, s.f.) (Seeed Studio, s.f.)

11. Raspberry Pi. Una Raspberry Pi es una computadora, posee un procesador, memoria, entradas y salidas; es capaz de recibir y procesar datos y entregar resultados. Su tamaño físico es como el de una tarjeta de crédito, se puede conectar a un monitor por medio de HDMI y utiliza el teclado y ratón estándar. También tiene acceso a internet por medio de un puerto Ethernet o un adaptador Wi-Fi. Es programable por medio de lenguajes como Python (Raspberry Pi Foundation, s.f.).

Figura 60. Vista superior de una Raspberry Pi modelo B.

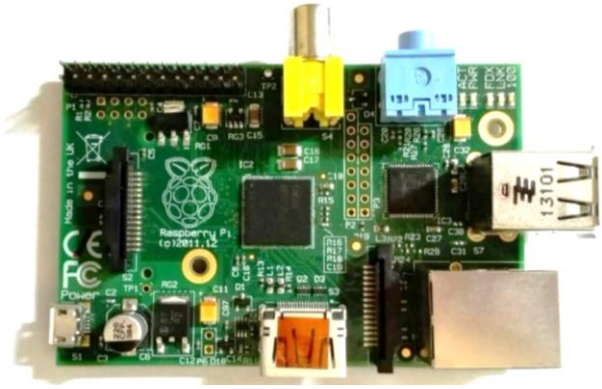
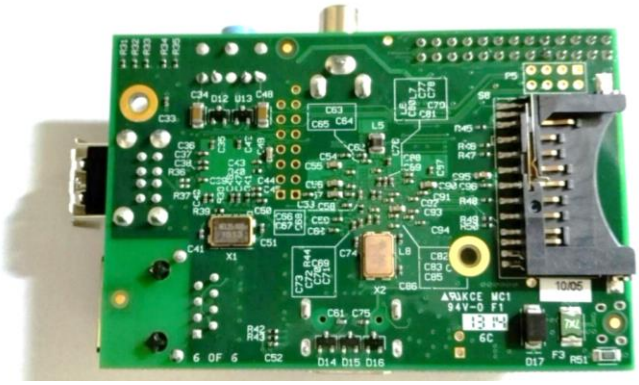


Figura 61. Vista inferior de una Raspberry Pi modelo B.

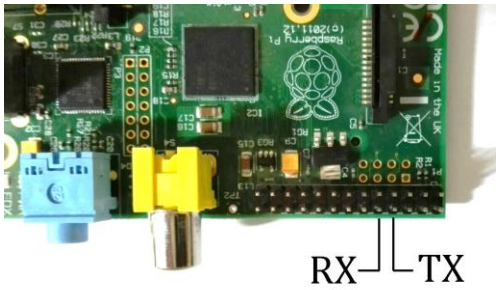


Para poder utilizar la Raspberry Pi, se debe conectar una tarjeta SD con una imagen del sistema operativo a usar. Los sistemas operativos que utiliza este dispositivo se basan en Linux, como por ejemplo, Raspbian. Este es un sistema operativo basado en Debian (sistema operativo y distribuidor de software libre) optimizado para el hardware de Raspberry Pi. Además de los programas y utilidades necesarios para que el dispositivo funcione, provee 35 mil paquetes y software precompilado para facilitar su instalación y mejorar el desempeño (Debian Project).

Figura 62. Tarjeta SD con sistema operativo para Raspberry Pi.

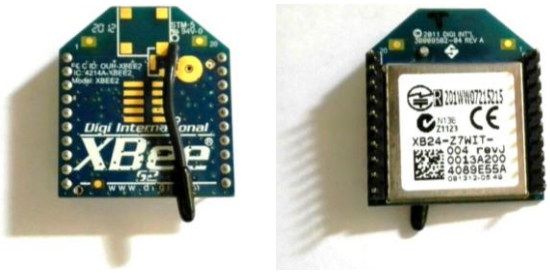
La Raspberry Pi posee un conjunto de pines para entradas y salidas digitales, 17 de ellos son de propósito general y 9 son de alimentación/tierra, siendo la interfaz física entre el módulo y el mundo exterior. En un par de ellos se encuentra lo necesario para la comunicación serial (TX – transmisión, RX – recepción). Con un programa en Python, puede leerse y escribirse dicho puerto. Otros pines incluyen funciones de comunicación serial síncrona (para protocolo I2C), salida de PWM, entre otros. (Raspberry Pi Foundation)

Figura 63. Pines GPIO de la Raspberry Pi empleados para comunicación serial.



12. XBee. Un XBee es un dispositivo de comunicación inalámbrica. Posee dos pines de interés que funcionan con comunicación serial asíncrona UART. Usando al menos dos de estas unidades, uno es capaz comunicarse de forma inalámbrica, pues poseen una antena que les sirve para la transmisión y recepción de ondas electromagnéticas, con la cual se transporta la información. Los módulos XBee soportan diversos protocolos inalámbricos y frecuencias RF (MCI Electronics, s.f.).

Figura 64. Dispositivo XBee en su vista superior e inferior.



La mayoría de los módulos operan a 2.4 GHz, pero también los hay que operan a 900 MHz. Estos últimos pueden llegar a tener un alcance de 24 km con una antena de alta ganancia. Debido a que en algunos países no es permitido operar a 900 MHz, existen versiones de 868 MHz. Estas versiones no pueden mezclarse en una misma red (Digi International, s.f.).

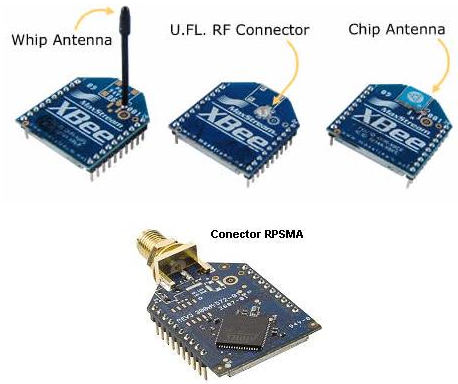
Los dispositivos XBee pueden operar de dos formas: en modo transparente o en modo API. Como indica Digi International, en el modo transparente, todos los datos recibidos por la entrada serial son inmediatamente transmitidos al medio, y cuando el módulo captura señales electromagnéticas, envía estos datos por la interfaz serial. En cambio, cuando opera en modo API (Application Programming Interface), el programador puede cambiar parámetros sin entrar al modo comando (esto significa que es comunicación directa con el módulo), obtener el valor de RSSI y dirección de la fuente de un paquete, y recibir confirmación de paquete entregado (Digi International).

Según Digi International, entre los estándares abiertos con los que trabajan los XBee se encuentran ZigBee, 802.15.4 y Wi-Fi. Zigbee es una alianza y un estándar de redes mesh, que busca eficiencia energética

y de costos. La serie 1 de XBee (llamados XBee 802.15.4) se utilizan en comunicaciones punto-a-punto y no necesitan ser configurados. La serie 2 actual (llamados ZB) funcionan en modo transparente o por medio de comandos API; funcionan en redes mesh. Las series 1 y 2 no son compatibles entre sí. La serie 2B es una variante de la serie 2, la cual mejora el consumo de potencia. También existe la familia 900 MHz que puede funcionar con el firmware DigiMesh y el Point-to-Multipoint. Los módulos XSC son módulos 900 MHz, que tienen mayor alcance pero menor velocidad. También existe la denominada versión PRO, que tiene un mayor alcance (1.6 km en línea de vista), pero también incrementa el consumo de potencia. La versión regular y PRO se pueden comunicar entre sí (MCI Electronics, s.f.).

MCI Electronics menciona que existen diversos tipos de antenas para mejorar el alcance de los módulos XBee. Entre ellas se encuentran las antenas de chip, cable, u.FL y RPSMA. Los últimos dos son conectores para antenas externas. Las antenas de cable permiten cubrir una mayor distancia que las de chip en exteriores, pero ocupan más espacio. Se evalúan las condiciones del proyecto y se sacrifica espacio por distancia, o viceversa. Estas vienen integradas en el módulo XBee, por lo que son las opciones más baratas. En el caso del conector u.FL y RPSMA, se conectan antenas más grandes y de mayor alcance, pero se incrementa el consumo de potencia. (Digi International)

Figura 65. Tipos de antenas para dispositivos XBee.



(MCI Electronics, s.f.)

El indicador de intensidad de la señal recibida (RSSI por sus siglas en inglés) provee un valor aproximado de la intensidad de la señal. Los XBee poseen receptores que son capaces de amplificar señales de un transmisor distante. La señal más débil que es capaz de detectar con un error aceptable se denomina sensibilidad. Si el transmisor se acerca, la intensidad de la señal en la antena receptora se incrementa. Medir la intensidad de la señal en este punto es un método para determinar la calidad del enlace de comunicación. Sin embargo, no es el único que se debe considerar. El comando DB lee el nivel de la señal en decibeles del último paquete correctamente recibido, y el XBee reporta el valor absoluto de RSSI (dBm). Este valor es preciso entre -40 dBm y la sensibilidad del receptor (Digi International, s.f.).

13. Repositorio. La Universidad de Salamanca define un repositorio como «un sitio web centralizado donde se almacena y mantiene información digital, normalmente bases de datos o archivos informáticos». Los archivos pueden estar referenciados desde su web al sitio original o pueden estar contenidos en su servidor. También pueden ser de acceso público o estar protegidos. Los más comunes son los de carácter académico.

GitHub es un servicio de alojamiento de repositorios de software. Es un sistema descentralizado (Paramio, s.f.). Es una plataforma de desarrollo colaborativo, revisión de código y manejo de código, realizando un almacenamiento de forma pública o privada (esto último si se paga una cuota) (GitHub, Inc.).

14. Base de datos. Según CCM Benchmark Group, una base de datos «es una entidad en la cual se pueden almacenar datos de manera estructurada». Proporciona a los usuarios el acceso a datos, que pueden visualizar, ingresar o actualizar según los derechos de acceso concedidos.

Una base de datos puede ser local (un equipo) o distribuida (la información se almacena en equipos remotos y se puede acceder a ella a través de una red). La ventaja principal es que múltiples usuarios pueden acceder a ella al mismo tiempo (CCM Benchmark Group, s.f.).

«La administración de bases de datos se realiza con el sistema DBMS (Database management system)» (CCM Benchmark Group, s.f.). CCM Benchmark Group describe este sistema como un conjunto de servicios que permite un fácil acceso a los datos, el acceso de múltiples usuarios y manipulación de los datos que contiene la base de datos. Existen cinco modelos de bases de datos: jerárquico, de red, relacional, deductivo y de orientación a objetos. En el modelo relacional, los datos se almacenan en tablas de dos dimensiones y se manipulan según la teoría relacional de matemáticas (CCM Benchmark Group, s.f.) . En este modelo, cada fila de la tabla se interpreta como una relación ordenada de valores y cada relación consta de un conjunto de atributos (columnas) (López).

Entre los DBMS principales se encuentra MySQL, que es el sistema de manejo de base de datos abierto más popular, y es desarrollado por Oracle. Las bases de datos de MySQL son relacionales y permite que cualquiera utilice el software sin costo así como modificar el código fuente según las necesidades del usuario. Es un sistema cliente-servidor que consiste de un servidor multi-hilos que soporta diferentes programas clientes y librerías, herramientas administrativas y APIs (Oracle Corporation, s.f.).

Figura 66. Logotipo de MyQSL.

(Oracle, s.f.)

E. METODOLOGÍA

Se planeó una tarea semanal para entregar paulatinamente, presentando las conclusiones y resultados importantes aproximadamente cada dos semanas. Se resume en tres categorías:

- Se realizó una investigación (en Internet principalmente) sobre los diferentes medios y tecnologías disponibles para llevar a cabo la transmisión y recepción de datos, así como de las plataformas disponibles para el dispositivo de control.
- Se realizaron pruebas de campo con la tecnología seleccionada, para asegurar que era la más adecuada y determinar sus límites de aplicación.
- Se desarrolló un programa para el módulo de control de comunicación y almacenamiento de datos, verificando su funcionamiento y compatibilidad con la tecnología de comunicación seleccionada.

1. Cronograma

Cuadro 12. Cronograma de actividades

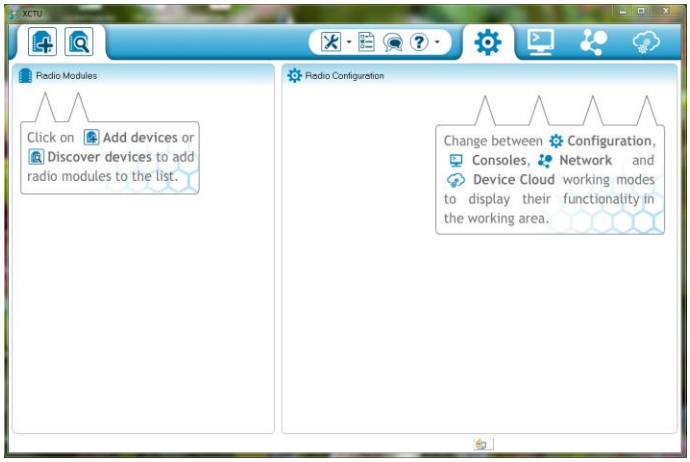
Actividad	Fecha de inicio	Fecha de entrega
Investigación y selección del medio de comunicación	Enero 16, 2015	Abril 17, 2015
Pruebas con dispositivos seleccionados	Abril 10, 2015	Mayo 8, 2015
Investigación, selección y pruebas con la plataforma de control	Abril 17, 2015	Mayo 8, 2015
Pruebas de comunicación con plataforma de control	Mayo 8, 2015	Mayo 22, 2015
Desarrollo del programa encargado del almacenamiento de información	Mayo 22, 2015	Junio 5, 2015
Desarrollo del programa encargado de la comunicación con dispositivo portátil	Junio 26, 2015	Septiembre 4, 2015
Integración de los programas para el funcionamiento del módulo	Julio 17, 2015	Septiembre 25, 2015
Diseño de placa y conexión por 3G	Septiembre 25, 2015	Noviembre 4, 2015

2. Pruebas de campo con la tecnología seleccionada. Después de llevar a cabo la investigación de los medios de transmisión y la selección del dispositivo de comunicación, se realizaron las pruebas de campo siguiendo los pasos siguientes:

a. Conexión directa entre dispositivos XBee. Las pruebas en esta sección se realizan con dispositivos XBee serie 2.

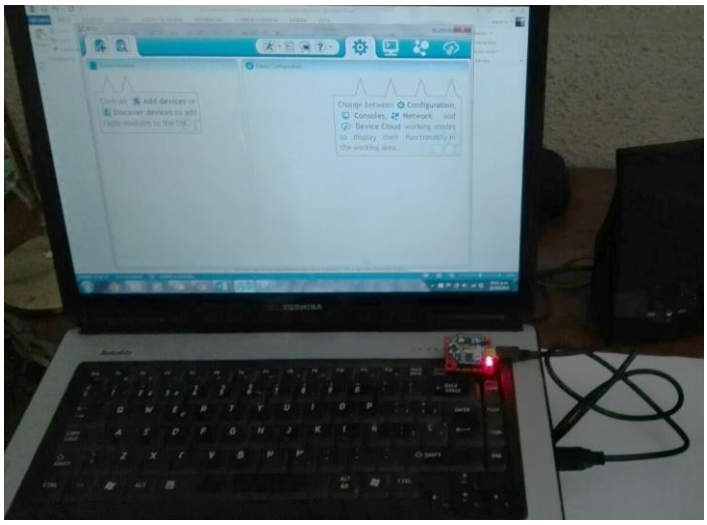
1) Descargar e instalar el software X-CTU, la interfaz para la configuración de los dispositivos XBee.

Figura 67. Interfaz de XCTU para trabajar con los dispositivos XBee.



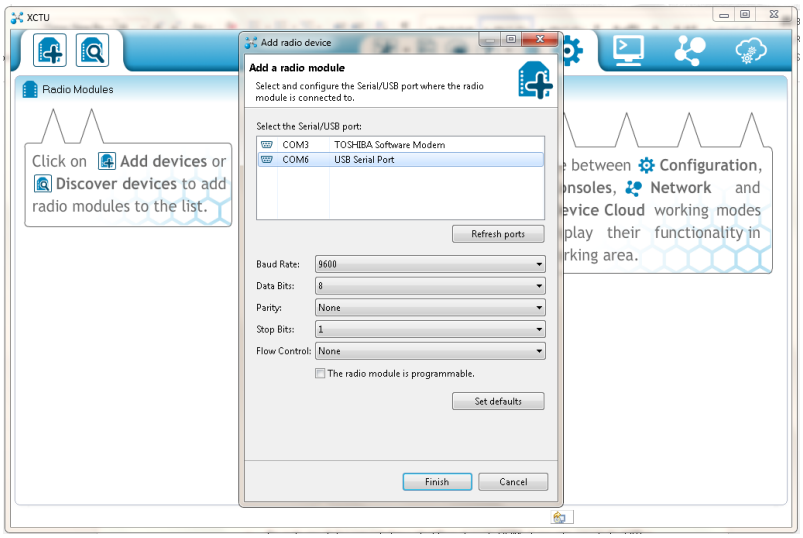
2) Conectar dos XBee, por medio de cable USB, a una computadora con el software instalado.

Figura 68. Conexión del dispositivo XBee a una computadora con XCTU instalado.



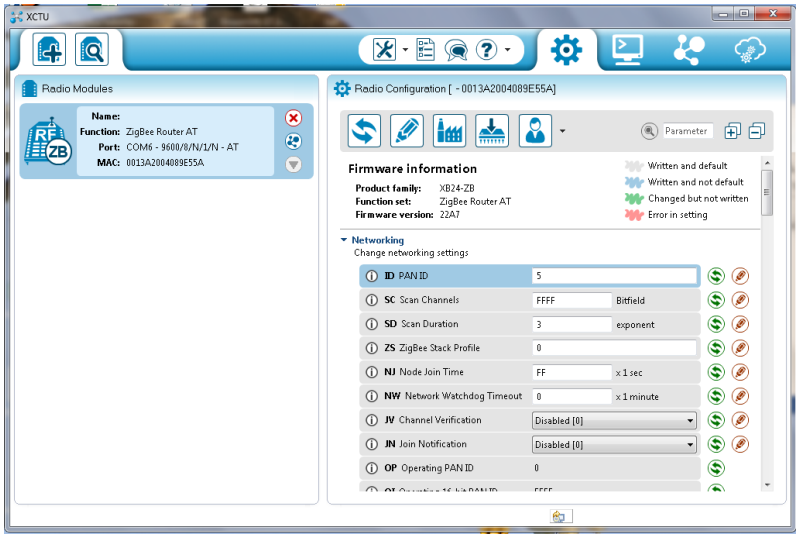
3) En cada una de las computadoras, identificar el puerto COM al que está conectado el XBee, así como el baud rate.

Figura 69. Identificación del puerto de conexión con el dispositivo XBee por medio de XCTU.



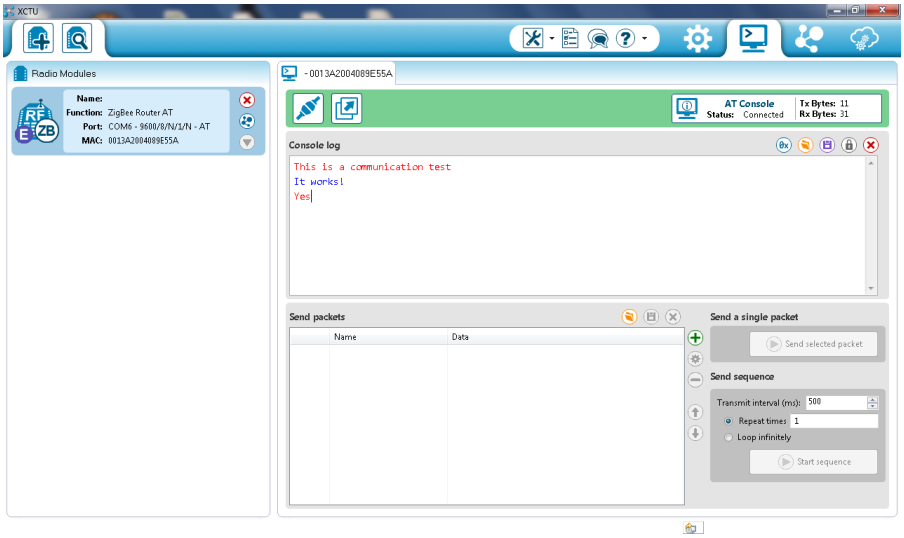
4) Configurar los dispositivos XBee en la misma PAN, uno como coordinador y otro como router.

Figura 70. Configuración del XBee por medio de XCTU.



5) Transmitir información de un XBee a otro en la ventana de comunicación.

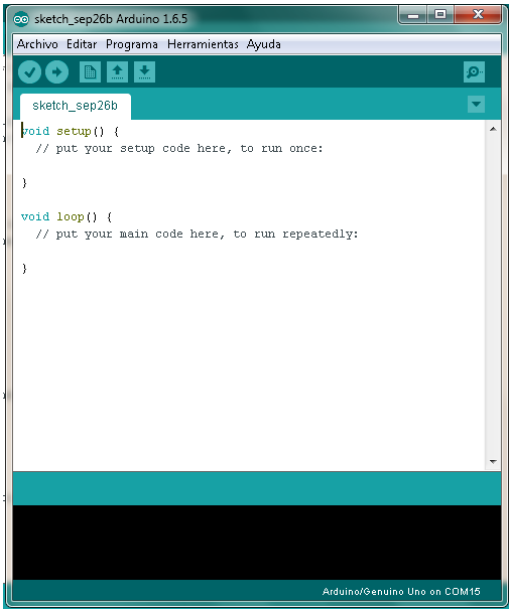
Figura 71. Verificación de comunicación entre dispositivos XBee por medio de XCTU.



b. Programación de Arduino para pruebas preliminares. Se utilizan los dispositivos XBee programados en la sección anterior.

1) Descargar e instalar el software para programar Arduino.

Figura 72. Interfaz para programación de Arduino.

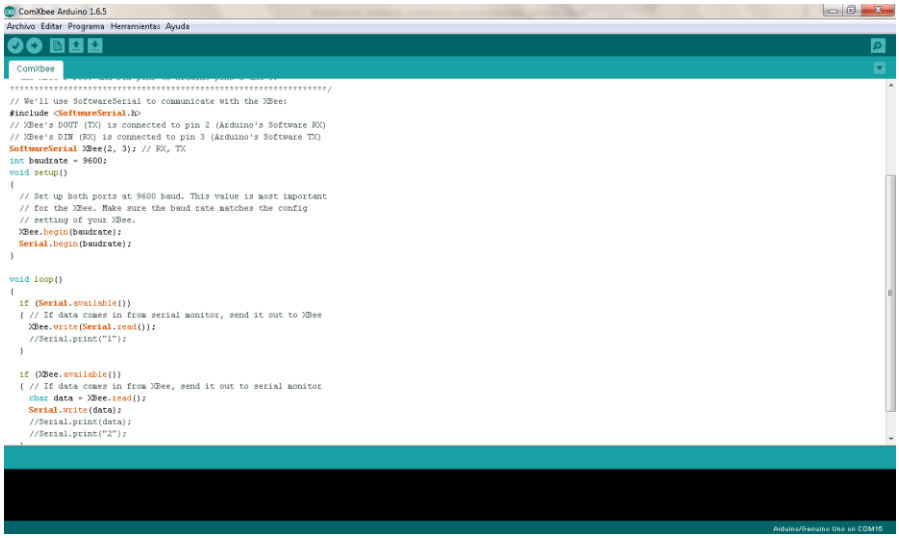


2) Conectar el Arduino Uno a la computadora con el software instalado por medio del puerto USB.

Figura 73. Conexión de Arduino Uno a una computadora con la interfaz de programación de Arduino instalada.

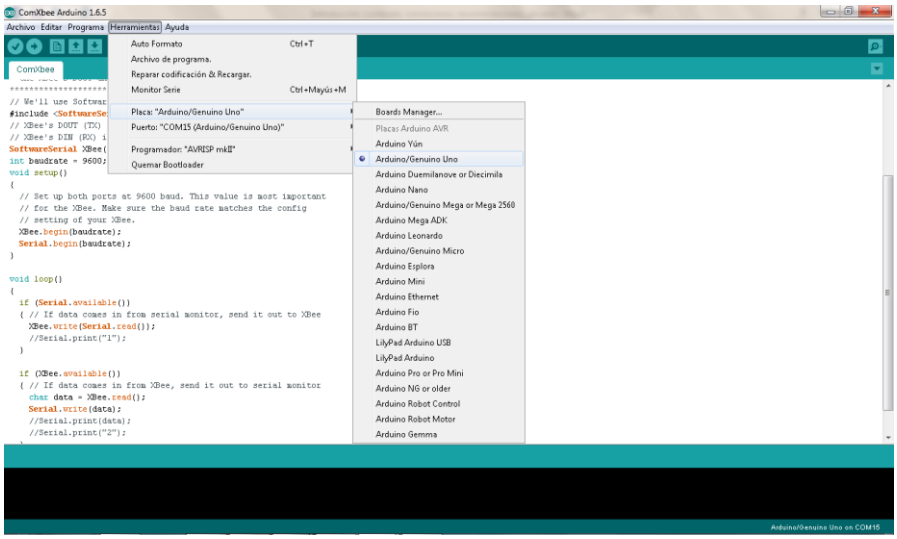
3) Realizar un programa en C, utilizando la interfaz de Arduino, para comunicar los dispositivos XBee.

Figura 74. Desarrollo del programa para comunicar el Arduino con el dispositivo XBee.



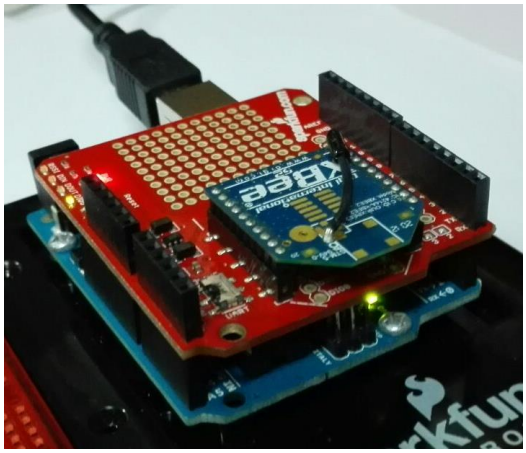
4) Cargar el programa en el Arduino Uno y abrir el Monitor Serie. Verificar que el puerto y la placa sean las correctas.

Figura 75. Verificación de placa (Arduino Uno).



5) Conectar el dispositivo XBee al Arduino Uno.

Figura 76. Conexión de dispositivo XBee con Arduino Uno.



6) Conectar el otro dispositivo XBee a la computadora con el software XCTU instalado y abrir la ventana de comunicación después de reconocerlo.

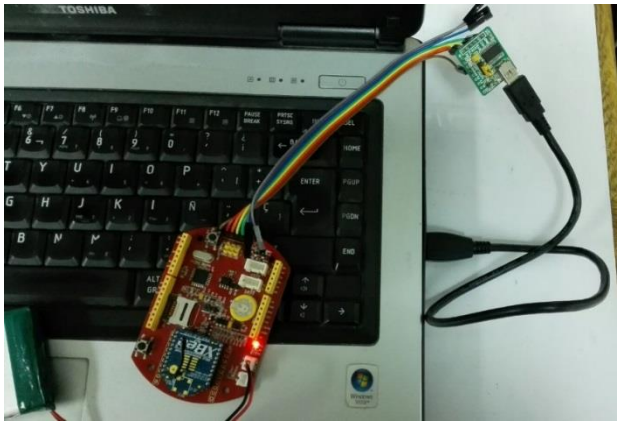
7) Enviar información de un dispositivo XBee a otro utilizando el Monitor Serie y la ventana de comunicación.

8) Incrementar el valor del baud rate y repetir el paso anterior para determinar hasta qué velocidad funcionan bien los dispositivos.

c. Programación de Seeeduino para pruebas en campo. Se utilizan los dispositivos XBee 900 y Serie 2.

1) Conectar el Seeeduino a la computadora con el software de Arduino instalado, por medio de cable USB y módulo serial.

Figura 77. Conexión del Seeeduino a una computadora con la interfaz de programación de Arduino instalada.



2) Realizar un programa en C, utilizando la interfaz de Arduino, para que el dispositivo XBee envíe la cadena de datos recibida del otro dispositivo XBee. Configurar el baud rate a 19200.

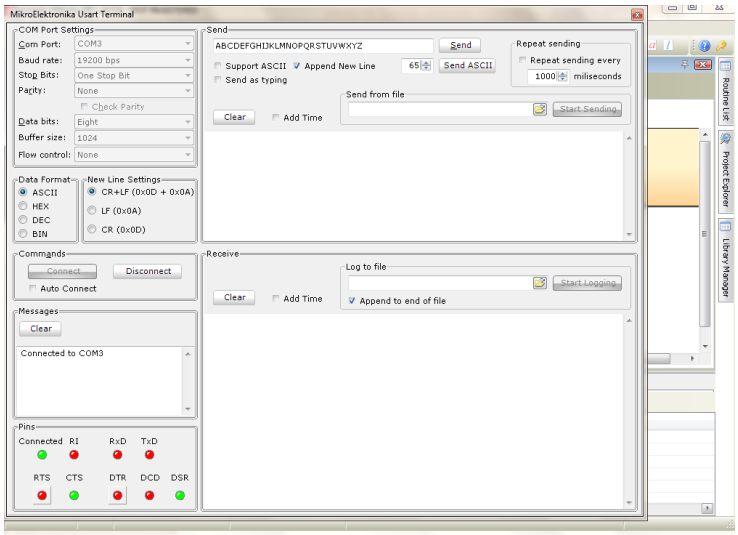
3) Cargar el programa en el Seeeduino. Verificar que el puerto sea el correcto y en la placa esté seleccionado Arduino Pro o Pro Mini (3.3V, 8 MHz).

4) Conectar el dispositivo XBee al Seeeduino.

Figura 78. Conexión del dispositivo XBee al Seeeduino.

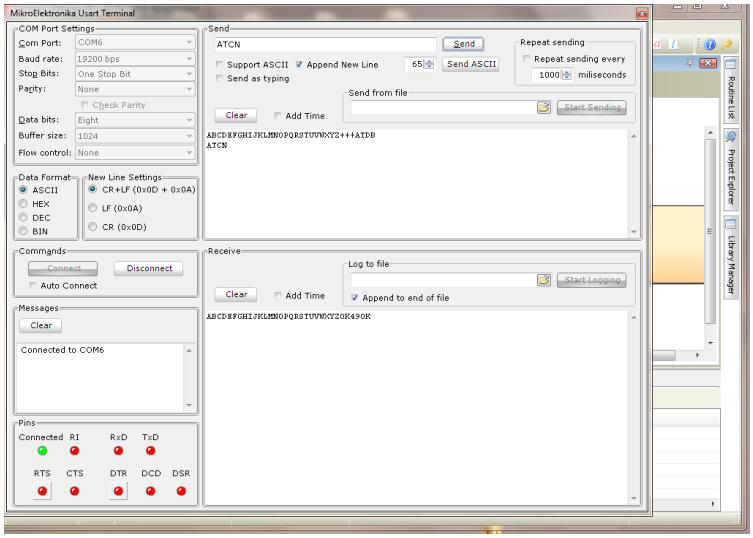
5) Conectar el otro dispositivo XBee a una computadora con mikroC Pro o cualquier otro programa con terminal serial y abrir dicho programa. Configurar la interfaz para observar los datos recibidos por el dispositivo XBee.

Figura 79. Terminal UART de mikroC para observar los datos del dispositivo XBee.



6) Estando en un sitio abierto, colocar un dispositivo XBee a un metro de distancia del otro dispositivo XBee y transmitir un dato a través de la terminal serial; observar que sí hay recepción y obtener el valor de RSSI. Para obtener el RSSI, se envía +++ (sin agregar nueva línea), ATDB (con nueva línea) y ATCN (con nueva línea).

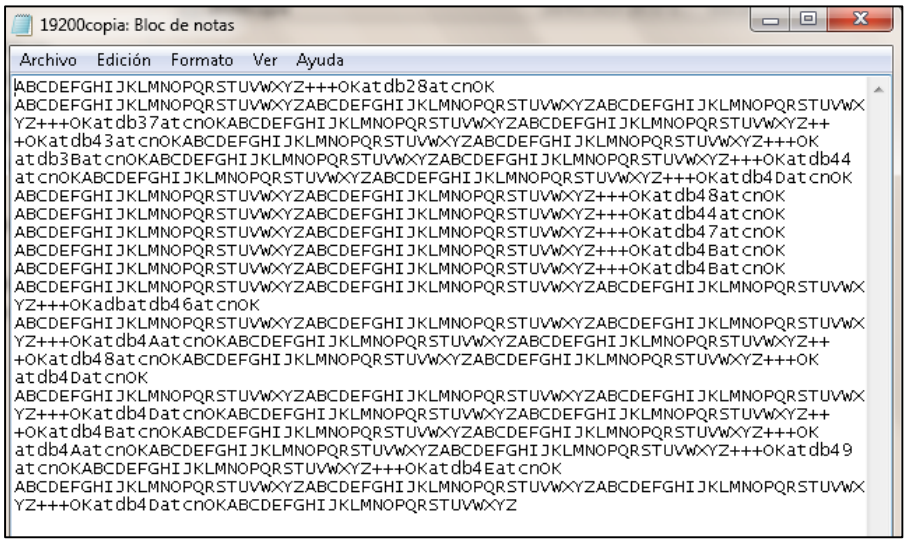
Figura 80. Terminal serial con el valor de RSSI de la última transferencia de datos realizada por los dispositivos XBee.



7) Realizar el paso anterior aumentando la distancia de 10 en 10 metros, hasta que la comunicación falle por completo.

8) Guardar los valores de RSSI obtenidos en un archivo de texto.

Figura 81. Archivo de texto con datos recibidos y lecturas de valor RSSI del dispositivo XBee.



9) Configurar los dispositivos XBee a un baud rate de 38400 y repetir las mediciones anteriores.

10) Configurar los dispositivos XBee a un baud rate de 115200 y repetir las mediciones anteriores.

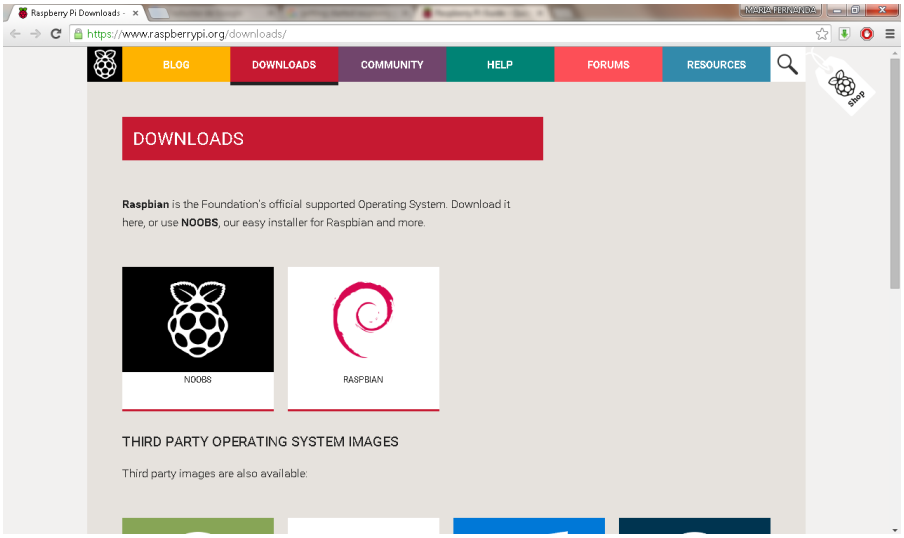
11) Al finalizar las mediciones con ambos módulos, se tabulan y grafican los valores de RSSI usando Microsoft Excel.

3. Módulo de control. Después de seleccionar el módulo de control, se procedió como sigue:

a. Configuración de Raspberry Pi

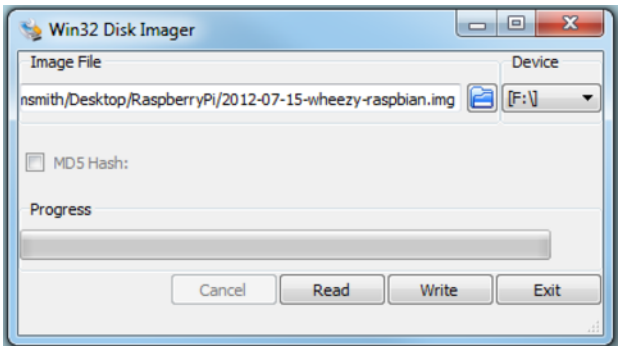
1) Descargar un sistema operativo (Raspbian) de la página de Raspberry Pi.

Figura 82. Sitio de Raspberry Pi para descarga del sistema operativo.



2) Quemar el sistema operativo en una tarjeta SD usando Win32 Disk Imager.

Figura 83. Interfaz de Win32 Disk Imager para quemar el sistema operativo en una tarjeta SD.



3) Colocar la tarjeta SD en su conector en la placa de Raspberry Pi.

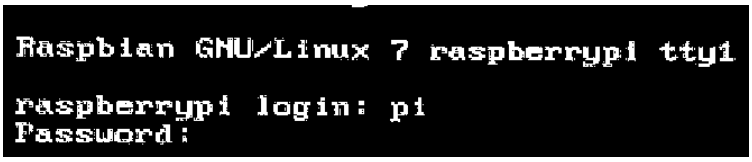
Figura 84. Conexión de la tarjeta SD a la Raspberry Pi.

4) Conectar el teclado, ratón, monitor y Ethernet. Luego conectar la alimentación.

Figura 85. Conexión de elementos externos a la Raspberry Pi.

5) Cuando la Raspberry Pi termine el proceso de arranque, ingresar el nombre de usuario (pi) y contraseña (raspberry).

Figura 86. Inicio de sesión en la Raspberry Pi.



6) Instalar MySQL en la Raspberry Pi para la base de datos.

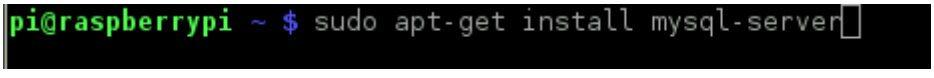
Obtener las actualizaciones de la Raspberry Pi con el comando

```
sudo apt-get update.
```

Instalar el servidor de MySQL con el comando

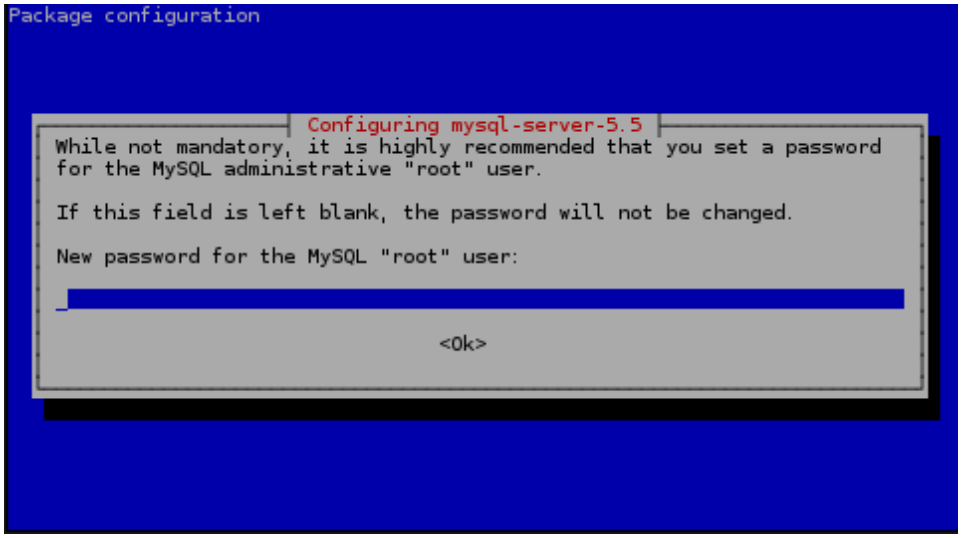
```
sudo apt-get install mysql-server.
```

Figura 87. Instalación de MySQL en la Raspberry Pi.



Ingresar una contraseña para el usuario “root”.

Figura 88. Interfaz de solicitud de contraseña para el usuario "root".



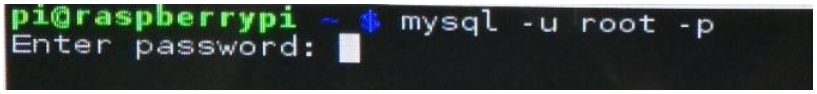
El servidor se inicia una vez termina la instalación. Reiniciar la Raspberry.

7) Configurar la base de datos. En la terminal, ingresar a MySQL con el comando

```
mysql -u root -p
```

Ingresar la contraseña.

Figura 89. Ingreso a la base de datos MySQL con el usuario "root".



Crear una base de datos, llamada serial, con el comando

```
create database serial;
```

Usar la base de datos con el comando

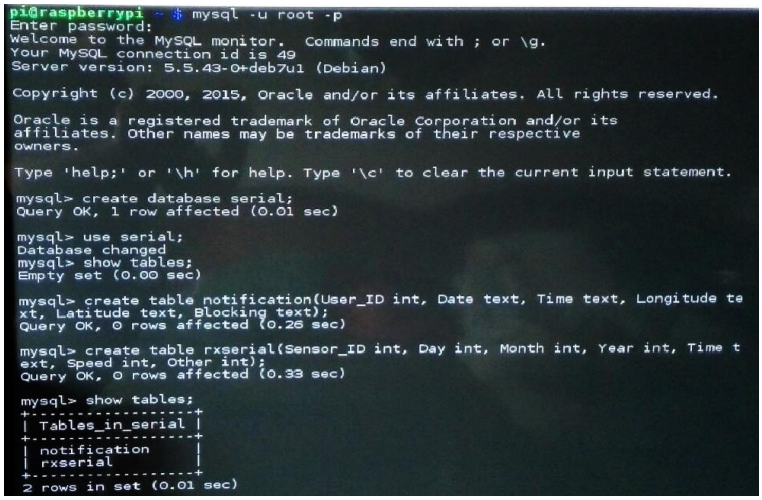
```
use serial;
```

Crear las tablas de la base de datos con sus parámetros usando los comandos

```
create table notification(User_ID int, Date text, Time text,
Longitude text, Latitude text, Blocking text);
```

```
create table rxserial(Sensor_ID int, Day int, Month int, Year
int, Time text, Speed int, Other int);
```

Figura 90. Creación de la base de datos y sus tablas en MySQL.



Salir de MySQL con el commando `exit;`

8) Cargar la interfaz gráfica con el comando `startx`.

Figura 91. Comando para cargar la interfaz gráfica en la Raspberry Pi.

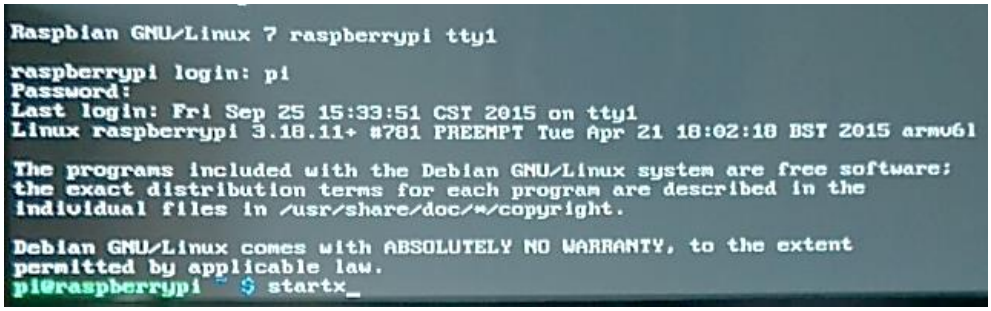
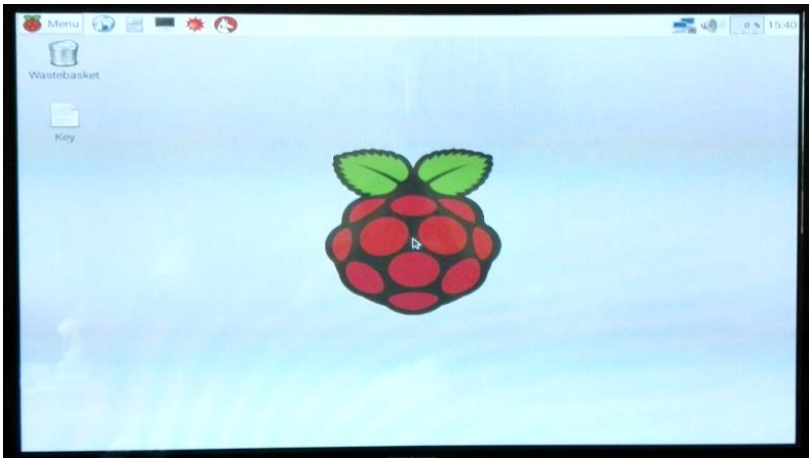
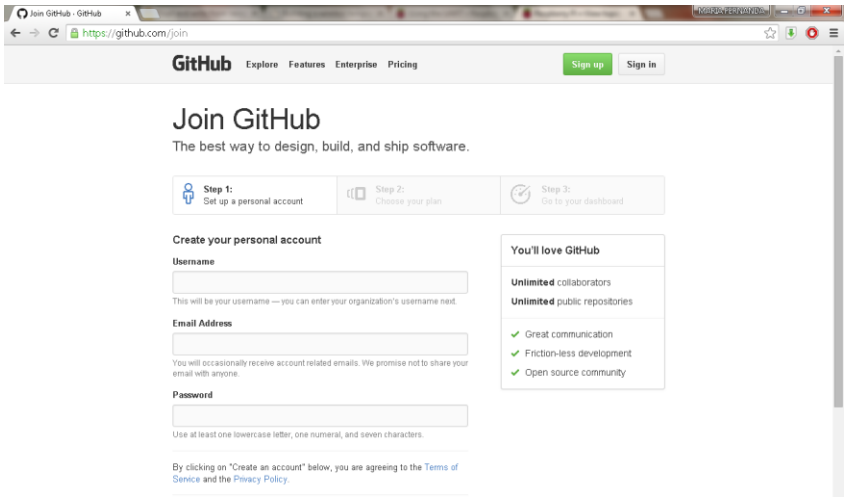


Figura 92. Interfaz gráfica de la Raspberry Pi.



9) Configurar el repositorio. Crear una cuenta en GitHub si no se posee una, ingresando a https://github.com/join.

Figura 93. Sitio de GitHub para crear una cuenta.



Una vez iniciada la sesión, crear un repositorio.

Figura 94. Sitio de GitHub del usuario.

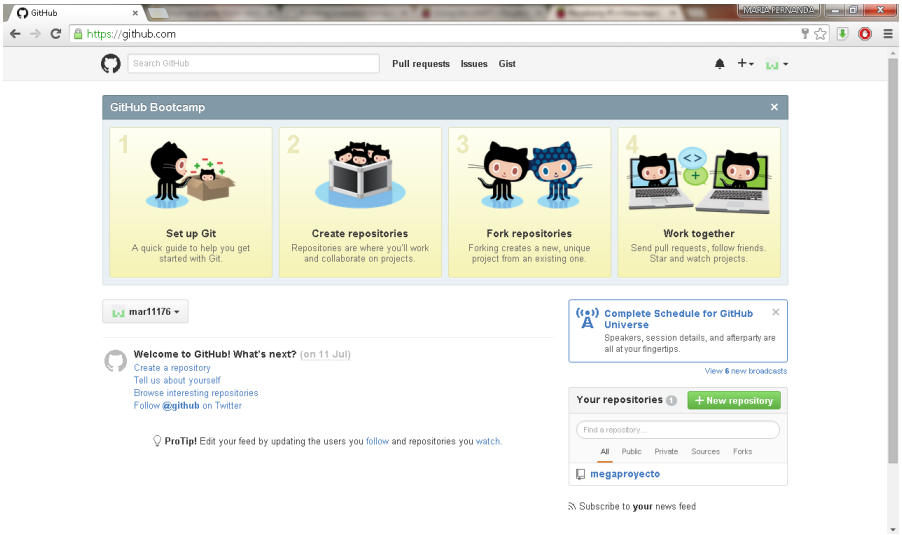
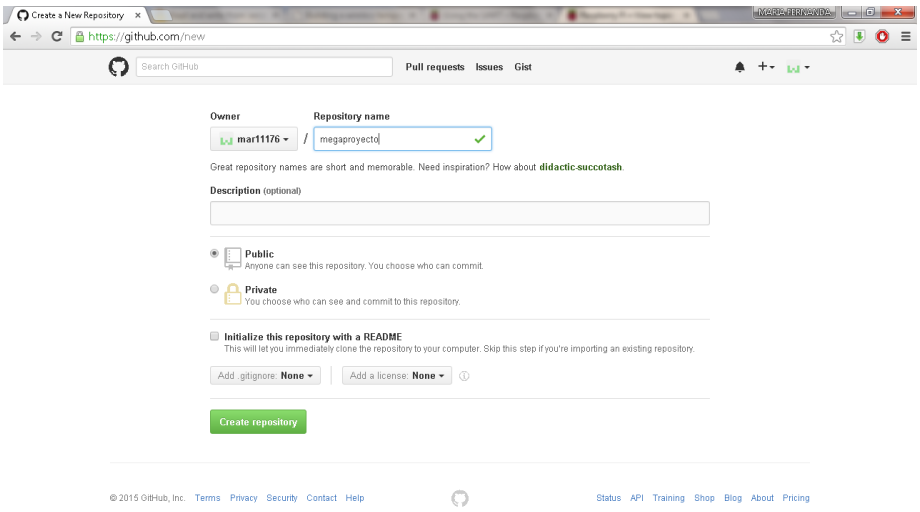
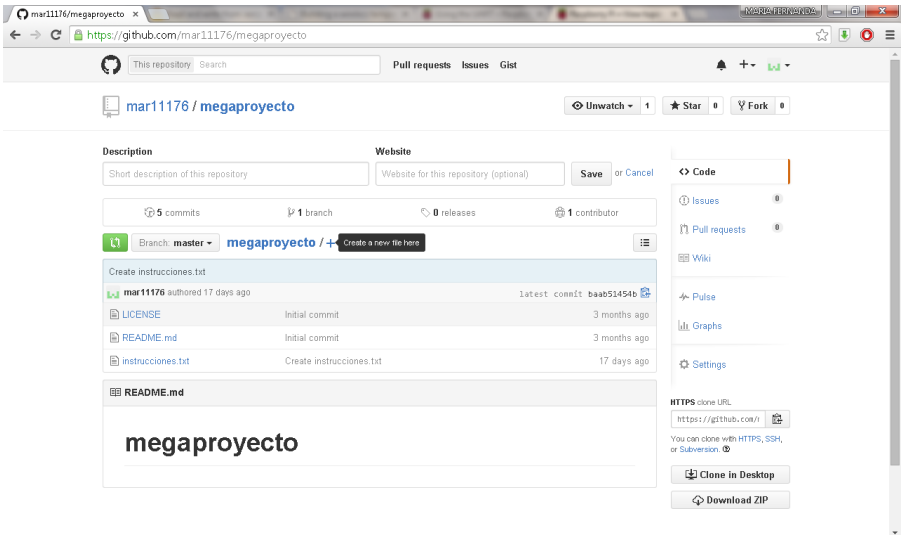


Figura 95. Creación de un nuevo repositorio en GitHub.



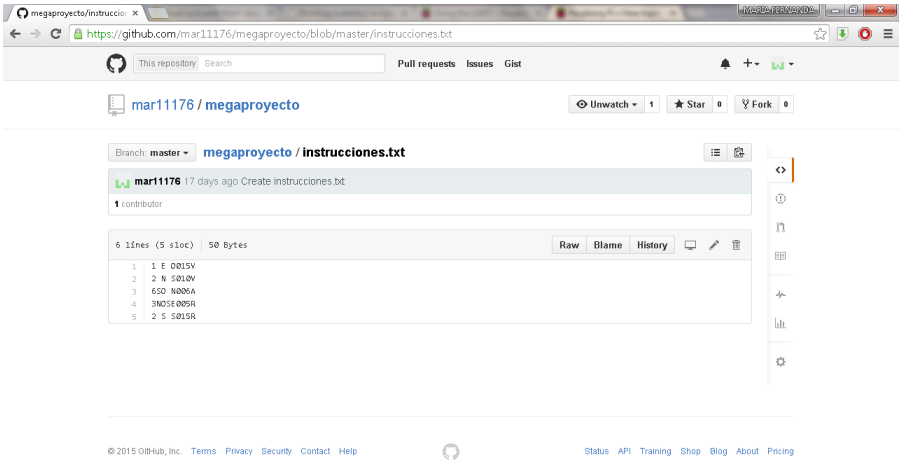
Crear un nuevo archivo.

Figura 96. Creación de un nuevo archivo dentro del repositorio seleccionado en GitHub.



Escribir los datos referentes a las instrucciones en el nuevo archivo y guardarlo.

Figura 97. Escritura de datos en el nuevo archivo del repositorio en GitHub.



Para acceder al repositorio desde Python, se utiliza la dirección

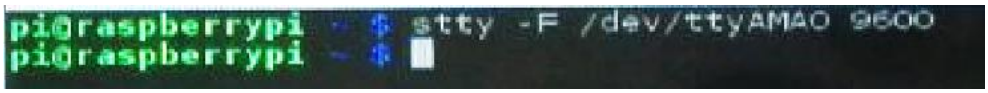
https://raw.github.com/mar11176/megaproyecto/master/instrucciones.txt

b. Pruebas con dispositivos XBee

1) En la terminal de la Raspberry Pi, configurar el puerto serial al baud rate de 9600 con el comando

```
stty -F /dev/ttyAMA0 9600.
```

Figura 98. Configuación del puerto serial en la Raspberry Pi.



2) Configurar los dispositivos XBee en la misma PAN y un baud rate de 9600.

3) Conectar el dispositivo XBee al puerto serial de la Raspberry Pi.

Figura 99. Conexiones en los pines GPIO de la Raspberry Pi correspondientes al puerto serial.

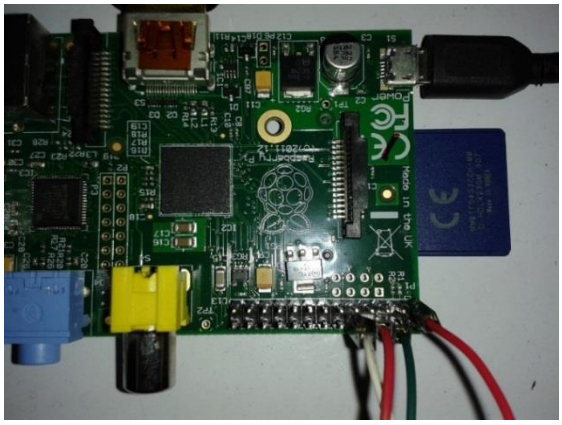
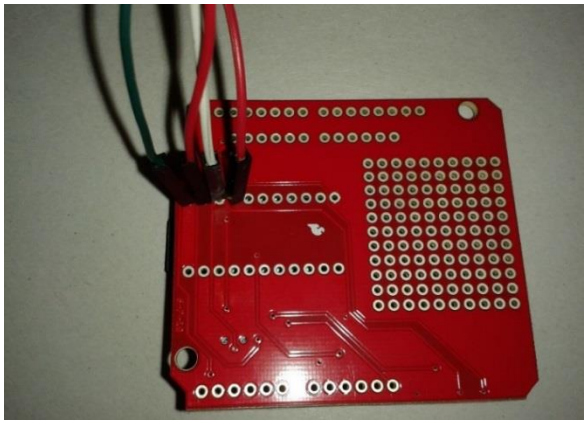


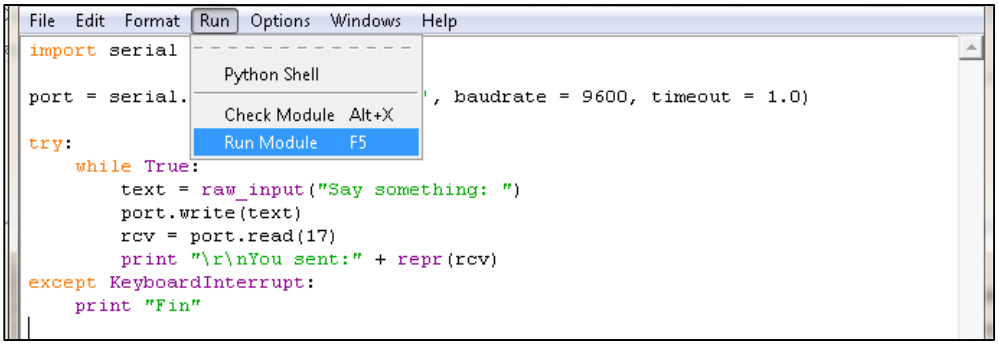
Figura 100. Conexiones a la tarjeta del dispositivo XBee.



4) Realizar un programa en Python que permita enviar y recibir información por el puerto. serial de la Raspberry Pi, al cual se conecta el dispositivo XBee.

5) Correr el programa.

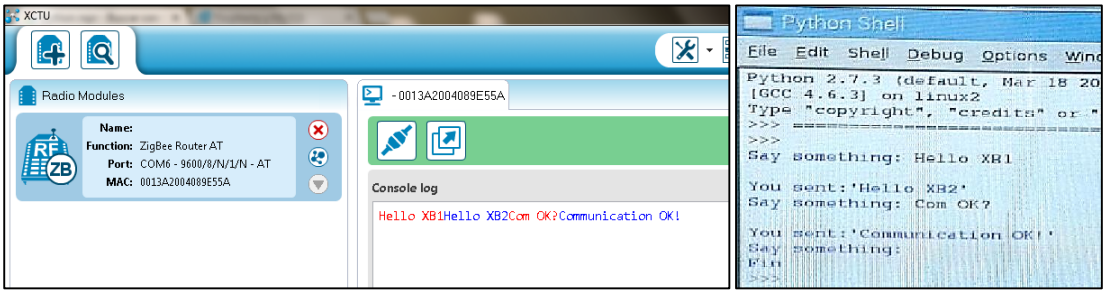
Figura 101. Ejecución del programa para comprobar la comunicación entre dispositivos.



6) Conectar el otro dispositivo XBee a una computadora con el software XCTU instalado, y después de reconocido, abrir la ventana de comunicación.

7) Intercambiar información entre los dispositivos escribiendo en la interfaz de Python y en la ventana de comunicación de XCTU. Verificar que la información es recibida.

Figura 102. Ventana de comunicación de XCTU y ventana de Python para verificar la comunicación.



8) Repetir los pasos anteriores para un baud rate de 115200, cambiando el valor de 9600 por 115200 en cada uno de los incisos.

c. Programa para la comunicación con el sensor

1) Realizar un programa en la Raspberry Pi que reciba los datos del sensor por el puerto serial y separe la cadena en los datos que se guardan en la tabla rxserial de la base de datos.

2) Conectar el dispositivo XBee al puerto serial de la Raspberry Pi.

3) Correr el programa y realizar las correcciones necesarias para tener una buena comunicación con el sensor, verificando que los datos sean almacenados correctamente.

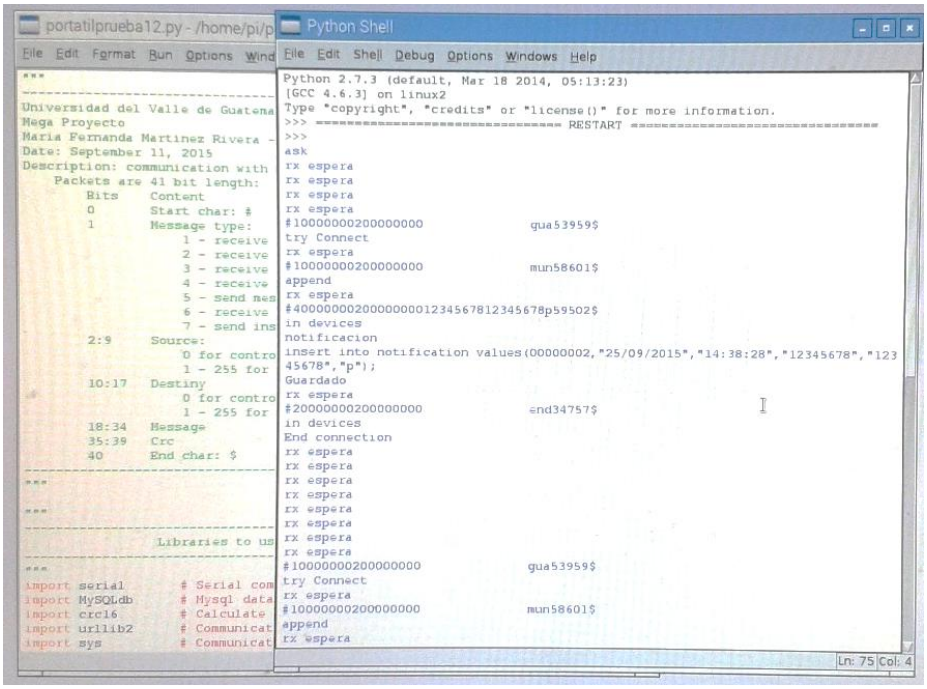
d. Programa para la comunicación con el dispositivo portátil

1) Realizar un programa en la Raspberry Pi que permita intercambiar información con el dispositivo portátil. Con esto, desarrollar el protocolo de comunicación con el cual se verifica la recepción de los datos, se realizan reenvíos y se desconecta si no hay respuesta.

2) Conectar el dispositivo XBee al puerto serial de la Raspberry Pi.

3) Correr el programa y realizar las modificaciones necesarias para asegurar una comunicación efectiva entre el dispositivo portátil y el sistema de control.

Figura 103. Ejemplo de comunicación efectiva entre dispositivo portátil y sistema de control.



e. Integración de programas y construcción de sistema final

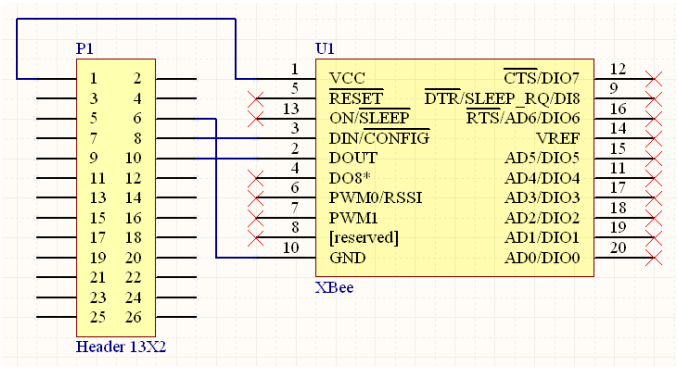
1) Una vez verificada la comunicación correcta con cada uno de los dispositivos, unificar los programas en uno solo, el cual permita recibir información del sensor continuamente al mismo tiempo que se efectúa un intercambio de información con un dispositivo móvil.

2) Realizar el diseño de la placa para conectar el dispositivo XBee a la Raspberry Pi.

Descargar el footprint del XBee.

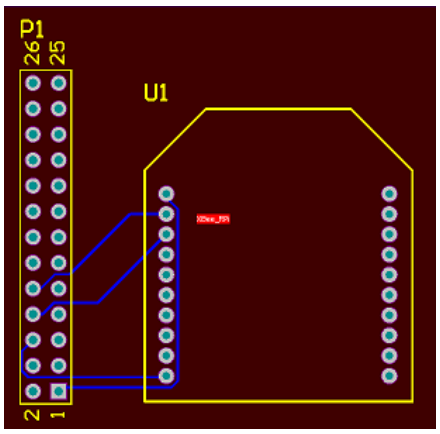
Realizar el esquemático en Altium Designer.

Figura 104. Diagrama esquemático de la conexión de dispositivo XBee con Raspberry Pi.



Generar la placa a partir del esquemático.

Figura 105. Diseño de la placa para conectar dispositivo XBee con Raspberry Pi.



Imprimir la placa.

Soldar los componentes.

Colocar el XBee en la placa y conectar a la Raspberry Pi.

Probar que funciona como se hizo anteriormente.

f. Conexión por módem ZTE MF626

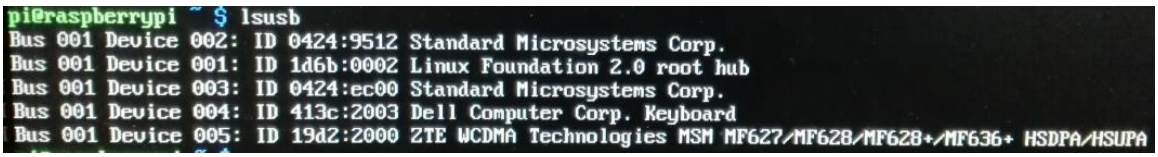
1) Con la Raspberry Pi conectada a Internet, instalar ppp, usb-modeswitch y wvdial en la Raspberry Pi utilizando el comando

```
sudo apt-get install ppp usb-modeswitch wvdial
```

2) Obtener los códigos del USB y modem, ya que el dispositivo puede trabajar en cualquiera de los dos modos. Desconectar la Raspberry Pi de Internet y conectar el módem. Obtener el código por defecto del dispositivo (última línea de la Figura 106. Obtención del código por defecto del módem ZTE MF626.Figura 106) con el comando

```
lsusb
```

Figura 106. Obtención del código por defecto del módem ZTE MF626.



Reiniciar la Raspberry Pi y repetir el paso anterior. Se observa que el código ha cambiado, y corresponde al código objetivo.

Figura 107. Obtención del código objetivo del módem ZTE MF626.

3) Crear el archivo de configuración para el cambio de estado del módem. Para ello, se utiliza el siguiente código.

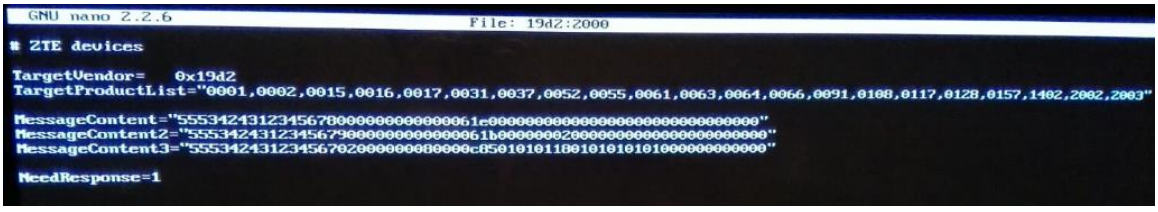
```
cd /tmp

tar -xzvf /usr/share/usb_modeswitch/configPack.tar.gz
19d2\:2000

leafpad 19d2:2000
```

Con este últlimo comando, se abre el editor de texto con el contenido que se muestra a continuación.

Figura 108. Contenido del archivo del módem ZTE MF626.



Tomar nota del “TargetVendor” y cada uno de los “MessageContent”. Salir del editor.

Abrir el archivo de usb_modeswitch con

```
sudo leafpad /etc/usb_modeswitch.conf
```

Agregar los campos de “DefaultVendor = 0x19d2”, “DefaultProduct = 0x2000”, “TargetVendor = 0x19d2”, “TargetProduct = 0x0031” y los “MessageContent” obtenidos anteriormente.

Guardar el documento.

4) Crear el archivo de configuración para conexión a 3G.

Abrir el archivo de configuración de wvdial con

```
sudo leafpad /etc/wvdial.conf
```

Reemplazar el contenido con

```
[Dialer 3gconnect]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Init3 = AT+CGDCONT=1,"IP","internet.movistar.gt"
Stupid Mode = 1
Modem Type = Analog Modem
ISDN = 0
Phone = *99#
Modem = /dev/gsmmodem
Username = { }
Password = { }
Baud = 9600
```

5) Iniciar sesión como usuario “root”. Para ello es necesario crear una contraseña con

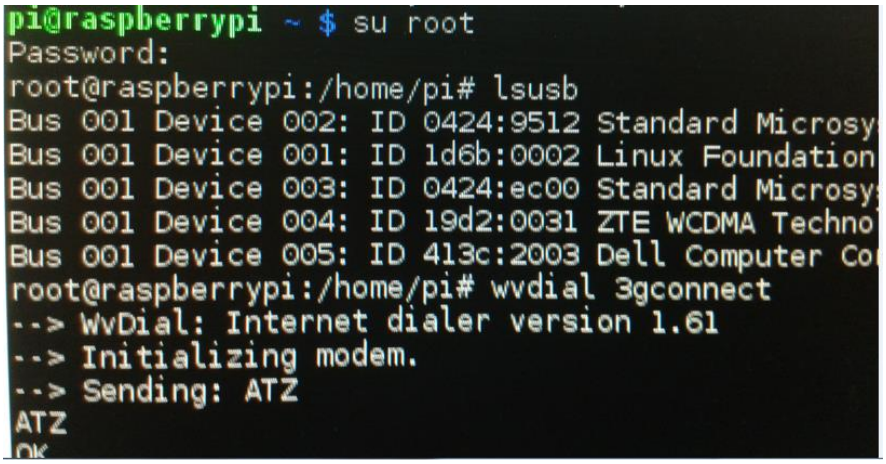
```
sudo passwd root
```

6) Conectar a Internet, verificando primero que el dispositivo se encuentre en el modo módem.

```
sudo usb_modeswitch -c /etc/usb_modeswitch.conf

wvdial 3gconnect
```

Figura 109. Conexión a Internet en la Raspberry Pi por medio del módem ZTE MF626.



F. RESULTADOS

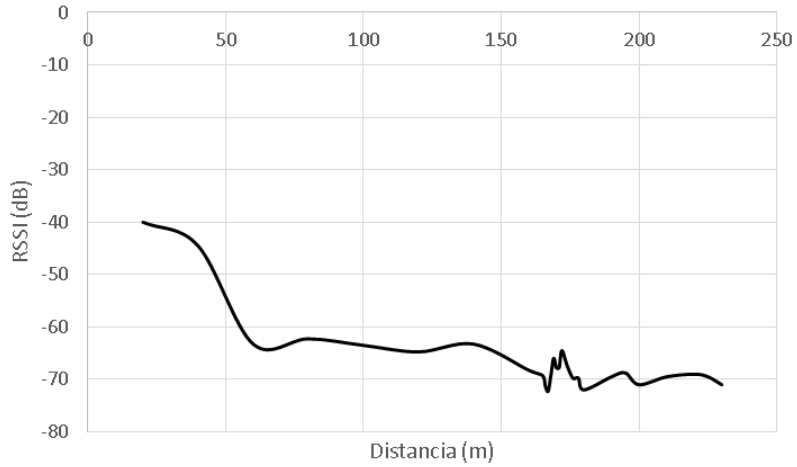
1. Mediciones con los XBee

a. XBee 900 – Abril. Se realizaron cuatro envíos en el mismo punto a un baud rate de 19200 y luego se calculó el promedio. Las mediciones fueron realizadas el 10 de abril en el Campo Marte. Eventualmente soplaba el viento. Al llegar a 230 m, se intentó avanzar más pero ya no se logró establecer comunicación.

Cuadro 13. Medión de RSSI con XBee 900 a un baud rate de 19200 en las pruebas de abril.

Distancia (m)	RSSI (dB)				
	Envío 1	Envío 2	Envío 3	Envío 4	Promedio
20	-40	-40	-40	-40	-40
40	-51	-45	-39	-43	-44.5
60	-63	-62	-59	-69	-63.25
80	-61	-61	-63	-64	-62.25
100	-64	-59	-65	-66	-63.5
120	-60	-74	-63	-62	-64.75
140	-69	-59	-64	-61	-63.25
160	-64	-68	-68	-73	-68.25
165	-71	-70	-68	-68	-69.25
166	-71	-69	-72	-73	-71.25
167	-77	-69	-73	-70	-72.25
168	-66	-73	-70	-68	-69.25
169	-66	-67	-66	-65	-66
170	-68	-71	-65	-67	-67.75
171	-71	-68	-64	-68	-67.75
172	-60	-68	-67	-63	-64.5
174	-63	-73	-70	-64	-67.5
176	-67	-75	-70	-67	-69.75
178	-69	-67	-69	-74	-69.75
180	-76	-67	-71	-74	-72
190	-74	-65	-73	-66	-69.5
195	-67	-70	-72	-66	-68.75
200	-73	-69	-69	-73	-71
210	-72	-67	-68	-71	-69.5
220	-65	-68	-71	-72	-69
225	-67	-69	-70	-72	-69.5
230	-72	-69	-70	-73	-71

Figura 110. Valor de RSSI según la distancia para un XBee 900 configurado a un baud rate de 19200.



b. XBee 900 – Mayo. Se realizaron tres envíos en cada punto y luego se calculó el promedio. Las mediciones fueron realizadas el 5 de mayo en el parqueo 1 de la Universidad del Valle de Guatemala.

Cuadro 14. Medición de RSSI con XBee 900 a un baud rate de 19200 en las pruebas de mayo.

Distancia (m)	RSSI (hex)	RSSI (dB)
1	28	-40
10	37	-55
20	43	-67
30	3B	-59
40	44	-68
50	4D	-77
60	48	-72
70	44	-68
80	47	-71
90	4B	-75
100	4B	-75
110	46	-70
120	4A	-74
130	48	-72
140	4D	-77
150	4D	-77
160	4B	-75
170	4A	-74
180	49	-73
190	4E	-78
200	4D	-77

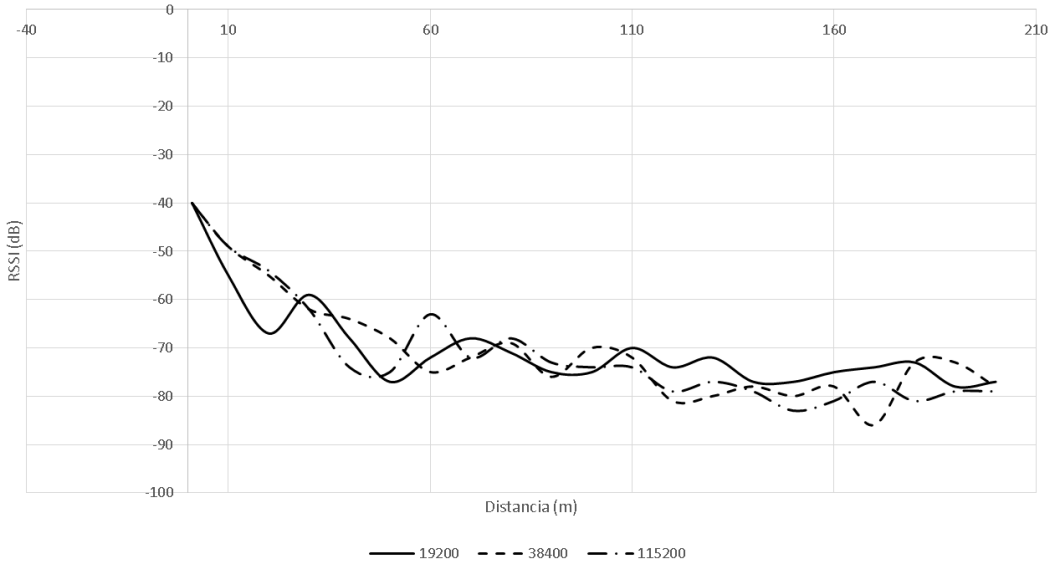
Cuadro 15. . Medición de RSSI con XBee 900 a un baud rate de 38400 en las pruebas de mayo.

Distancia (m)	RSSI (hex)	RSSI (dB)
1	28	-40
10	31	-49
20	37	-55
30	3E	-62
40	40	-64
50	44	-68
60	4B	-75
70	48	-72
80	45	-69
90	4C	-76
100	46	-70
110	48	-72
120	51	-81
130	50	-80
140	4E	-78
150	50	-80
160	4E	-78
170	56	-86
180	49	-73
190	49	-73
200	4E	-78

Cuadro 16. . Medición de RSSI con XBee 900 a un baud rate de 115200 en las pruebas de mayo.

Distancia (m)	RSSI (hex)	RSSI (dB)
1	28	-40
10	31	-49
20	36	-54
30	3E	-62
40	4A	-74
50	4B	-75
60	3F	-63
70	48	-72
80	44	-68
90	49	-73
100	4A	-74
110	4A	-74
120	4F	-79
130	4D	-77
140	4F	-79
150	53	-83
160	51	-81
170	4D	-77
180	51	-81
190	4F	-79
200	4F	-79

Figura 111. Comparación del valor de RSSI según la distancia a diferentes baud rates para el XBee 900.



c. XBee Serie 2 – Mayo. Se realizaron cuatro envíos en cada punto y luego se calculó el promedio. Las mediciones fueron realizadas el 5 de mayo en el parqueo 1 de la Universidad del Valle de Guatemala.

Cuadro 17. Medición de RSSI con XBee Serie 2 a un baud rate de 19200 en las pruebas de mayo.

Distancia (m)	RSSI (hex)	RSSI (dB)
0.5	39	-57
10	44	-68
20	4C	-76
30	50	-80
40	5F	-95
50	59	-89
60	60	-96
70	61	-97
80	61	-97
90	61	-97
100	-	

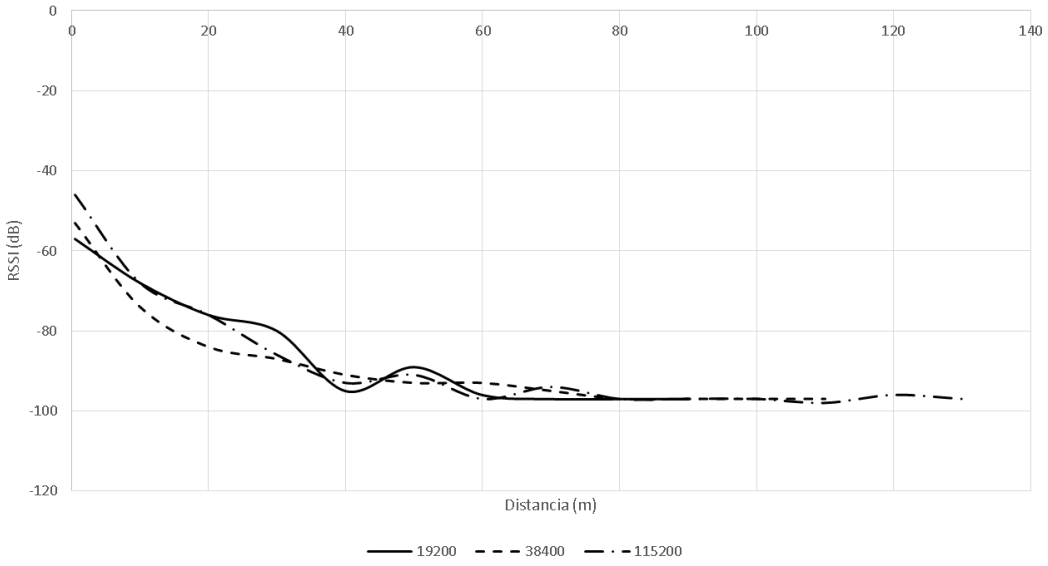
Cuadro 18. Medición de RSSI con XBee Serie 2 a un baud rate de 38400 en las pruebas de mayo.

Distancia (m)	RSSI (hex)	RSSI (dB)
0.5	35	-53
10	4A	-74
20	54	-84
30	57	-87
40	5B	-91
50	5D	-93
60	5D	-93
70	5F	-95
80	61	-97
90	61	-97
100	61	-97
110	61	-97
120	-	

Cuadro 19. Medición de RSSI con XBee Serie 2 a un baud rate de 115200 en las pruebas de mayo.

Distancia (m)	RSSI (hex)	RSSI (dB)
0.5	2E	-46
10	44	-68
20	4C	-76
30	56	-86
40	5D	-93
50	5B	-91
60	61	-97
70	5E	-94
80	61	-97
90	61	-97
100	61	-97
110	62	-98
120	60	-96
130	61	-97
140	-	

Figura 112. Comparación del valor de RSSI según la distancia a diferentes baud rates para el XBee Serie 2.



2. Pruebas en Raspberry Pi

a. Descripción del protocolo de comunicación. El sistema se basa en la comunicación serial configurado a un baud rate de 115200. Espera recibir un caracter con el cual identifica si es un mensaje del sensor o de un dispositivo portátil. Solo hay recepción del sensor, y con el dispositivo portátil sí hay intercambio de información. Los mensajes tiene la siguiente estructura:

Byte 0	Byte 1	Byte 2 - 9	Byte 10 - 17	Byte 18 - 35	Byte 36 - 40	Byte 41
Identificador	Tipo de instrucción	ID fuente	ID destino	Datos	CRC	Terminador

En el identificador se coloca una letra `s` si es mensaje del sensor o un numeral (#) si es mensaje de/hacia dispositivo portátil. El tipo de instrucción se utiliza en la comunicación con el dispositivo portátil, y describe si es una solicitud de conexión (1), solicitud de terminación (2), solicitud de instrucción (3), envío de notificación (4), mensaje del sistema de control (5) o envío de instrucción del sistema de control (7).

Cuando el dispositivo portátil envía un mensaje, en el ID fuente coloca su número de identificador, y en el ID destino coloca el número de identificador del sistema de control. Este último tiene un valor por defecto de cero. Ya que se tienen ocho caracteres disponibles, puede haber hasta 255 dispositivos en la red.

En el campo de datos, el dispositivo portátil coloca la información pertinente a cada tipo de instrucción. En el caso de una solicitud de conexión (1), se debe seguir la secuencia de mensajes adecuada para establecer la conexión. Para ello, el dispositivo portátil coloca la palabra “gua” en el extremo derecho del mensaje, dejando en blanco el resto del campo. El sistema de control responde con “502” en la misma posición, a lo

que el dispositivo debe contestar con “mun”, teniendo la misma estructura que el primer mensaje. Una vez establecida la conexión, el dispositivo puede enviar una solicitud de instrucción, enviar una notificación y terminar la conexión.

Cuando envía una solicitud de instrucción (3), el dispositivo portátil coloca la palabra “req” en el extremo derecho del mensaje, dejando el resto del campo en blanco. El sistema de control responde con la primera instrucción en su lista, la cual descarga del repositorio. Cuando la instrucción es enviada, el sistema la elimina de la lista. Cuando la lista se vacía, el sistema vuelve a conectarse al repositorio y descarga una nueva lista. La instrucción está formada por 17 caracteres con los siguientes campos.

Cuadro 20. Campos de una instrucción a transmitir al dispositivo portátil.

Campo	Datos	Codificación
Vía	Calle	1
	Avenida	2
	Boulevard	3
	Diagonal	4
	Arco	5
	Vía	6
	Ruta	7
Dirección de	Norte	N
	Sur	S
	Este	E
	Oeste	O
Dirección hacia	Noreste	NE
	Noroeste	NO
	Sureste	SE
	Suroeste	SO
Tiempo de vía	000 – 999	
Parámetro de congestión	Verde	V
	Amarillo	A
	Rojo	R
Hora de envío	HH:MM:SS	

Cuando el dispositivo portátil envía una notificación (4), el sistema desglosa los datos en los campos de la tabla notificación la base de datos serial, los guarda en dicha tabla y envía un acuse de recibo al dispositivo portátil.

Para terminar una conexión, el dispositivo portátil envía una instrucción de tipo 2. Siempre termina la conexión después de solicitar la instrucción o enviar la notificación. En caso esto no suceda, el sistema de control tiene un temporizador que le indica cuando dar fin a la conexión si no hay mensaje del dispositivo.

El sensor coloca su información correspondiente en el campo de datos, de manera que el sistema de control los divide para guardarlos en la tabla rxserial de la base de datos serial.

En el campo de CRC, tanto el dispositivo portátil como el módulo de control colocan un checksum, esto es para la detección de errores. Se utiliza el CRC-CCITT XMODEM como lo calcula Python. El sistema verifica si el checksum es correcto antes de analizar el contenido del mensaje.

Por último, el terminador de cadena es el símbolo de dólar $. Se utiliza para identificar el fin de la cadena de datos.

b. Comunicación con dispositivo portátil. Se muestra un ejemplo de comunicación con el dispositivo portátil con el envío de una notificación y la solicitud de una instrucción por parte de este.

Figura 113. Mensaje de notificación que transmite el dispositivo portátil.

```
20:24:04 #1000000
20:24:05 0200000000              gua53959$#10000000200000000
20:24:06                 mun58601$
20:24:07 #4000000020000000012345
20:24:08 67812345678p59502$
20:24:09 #20000000200000000
20:24:10               end34757$
```

Figura 114. Mensajes que recibe la Raspberry Pi y operaciones que ejecuta con la notificación.

```
rx espera
#10000000200000000              gua53959$
try Connect
rx espera
#10000000200000000              mun58601$
append
rx espera
#40000000200000000123456781234567 8p59502$
in devices
notificacion
insert into notification values(00000002,"25/09/2015","14:38:28","12345678","123
45678","p");
Guardado
rx espera
#20000000200000000              end34757$
in devices
End connection
rx espera
```

Figura 115. Datos recibidos guardados en la tabla notification de la base de datos serial.

```
|      2 | 23/09/2015 | 18:48:20 | 12345678 | 12345678 | t
|      2 | 25/09/2015 | 14:38:28 | 12345678 | 12345678 | p
+--------+------------+----------+----------+----------+---------
41 rows in set (0.01 sec)

mysql>
```

Figura 116. Mensajes de solicitud de información enviados por el dispositivo portátil.

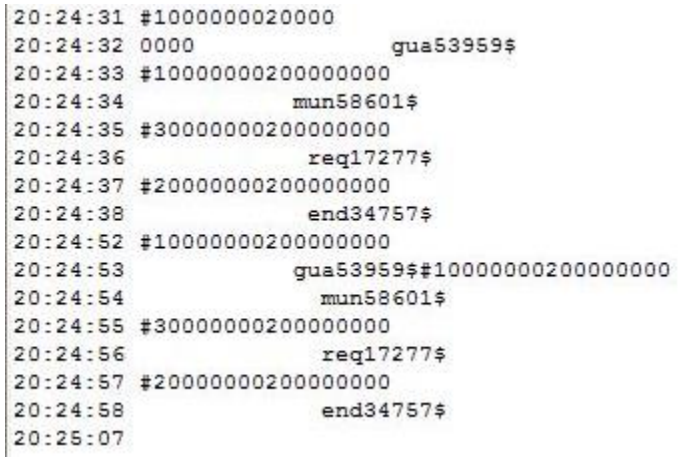


Figura 117. Mensajes que recibe la Raspberry Pi en una solicitud.

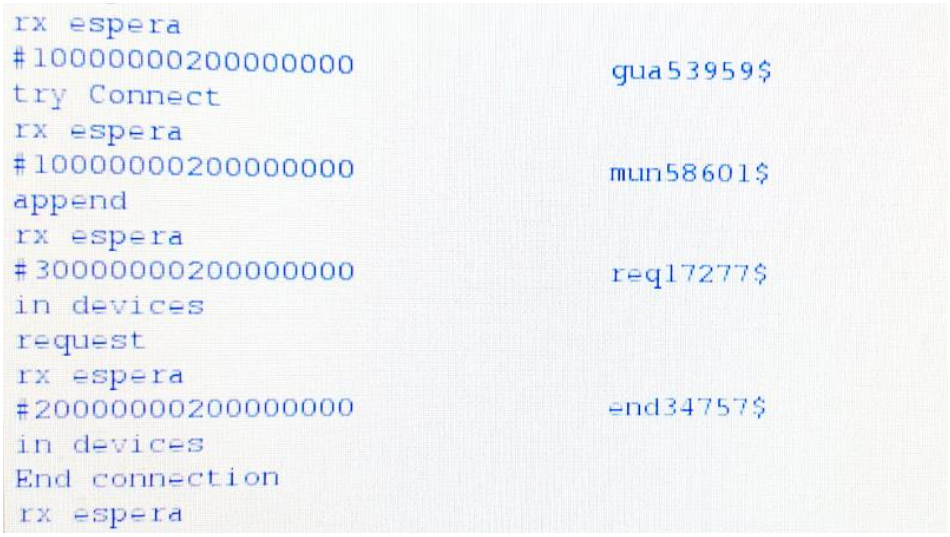
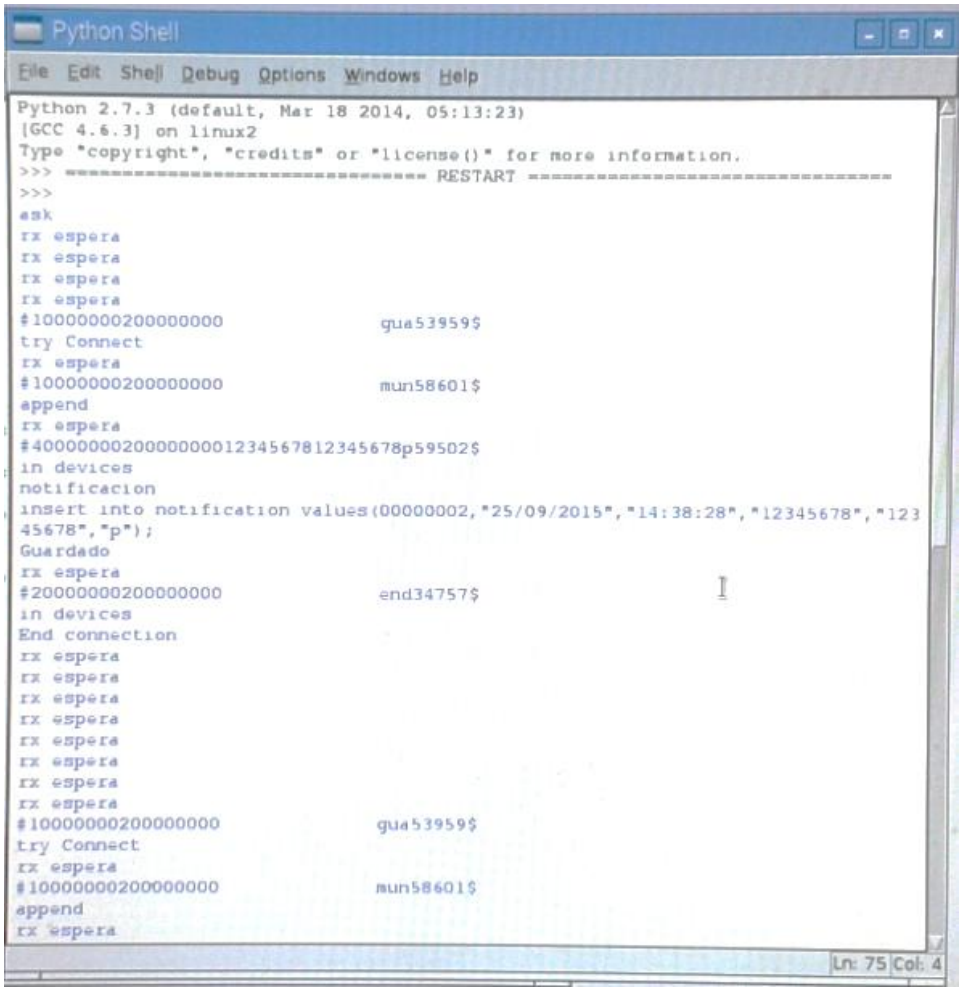
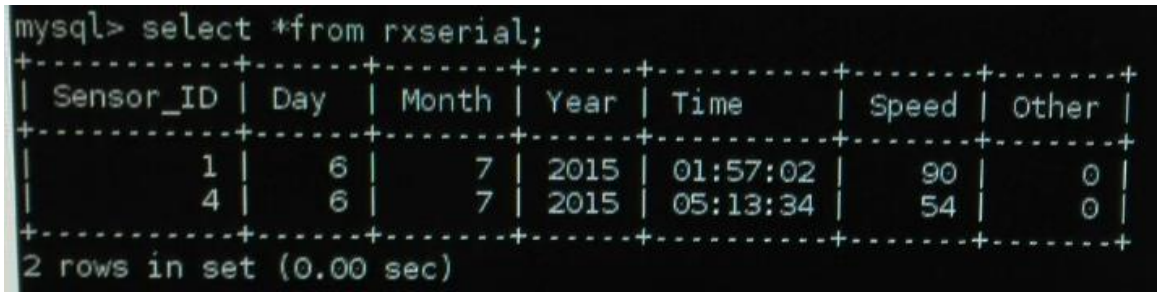


Figura 118. Ejemplo de secuencia de recepción de mensajes en la Raspberry Pi.



c. Comunicación con sensor. El programa recibe una cadena por UART, transmitida por el sensor, y la divide según los campos a guardar en la tabla rxserial de la base de datos serial.

Figura 119. Datos recibidos del sensor guardados en la tabla rxserial de la base de datos serial.

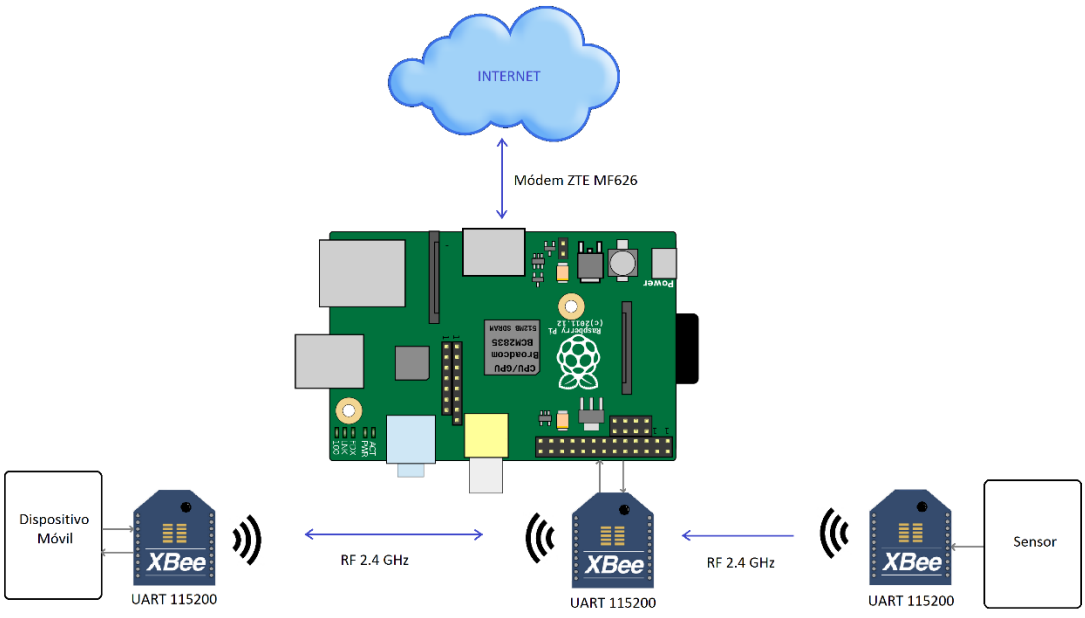


d. Sistema final. Después de tener la placa para conectar el dispositivo XBee a la Raspberry Pi, se prueba la red formada por los sensores, el dispositivo portátil y el módulo de control. Este último siempre se encuentra escuchando; los datos que recibe del sensor son almacenados en una tabla de la base de datos de MySQL mientras que con el dispositivo portátil realiza un intercambio de mensajes para establecer una comunicación segura y responder adecuadamente según el mensaje recibido.

Uno de los XBee está configurado como coordinador, otro como router y los demás como dispositivos terminales, todos en la PAN 5 a un baud rate de 115200 y en modo transparente. La topología resultante es una mesh ZigBee con módulos XBee serie 2, con una Raspberry Pi modelo B como módulo de control.

La Raspberry Pi se conecta a Internet para descargar las instrucciones para transmitir al usuario con el dispositivo portátil cada vez que este lo solicite. La conexión la realiza a través del módem ZTE MF626 de Movistar.

Figura 120. Diagrama de la arquitectura del sistema final.



G. ANÁLISIS DE RESULTADOS

Se seleccionó un sistema de comunicación inalámbrico serial debido a la aplicación a desarrollar. Teniendo en mente que este sistema se diseña para ser colocado en las calles de la ciudad de Guatemala, es más práctico que los módulos se comuniquen de forma inalámbrica, pues una red cableada implicaría un costo elevado en infraestructura de la red, además de consideraciones de protección contra exposición a la intemperie. La comunicación serial asíncrona permite que múltiples dispositivos trabajen de forma independiente y se comuniquen cuando lo necesiten, evitando una línea adicional para compartir reloj.

Se desarrolló una topología mesh ZigBee donde los nodos terminales corresponden a los sensores y a los dispositivos portátiles con su respectivo XBee; los routers son módulos intermedios que permiten extender la red, siendo el puente entre las unidades terminales y el coordinador. Solo hay un coordinador en la red, y en este caso es el módulo XBee que se conecta al sistema de control.

El sistema de control es una Raspberry Pi modelo B ya que los recursos que ofrece son más que suficiente para la aplicación que se desea implementar. Se requiere un puerto UART para conectar el coordinador de la red; a través de él recibe los datos recolectados por los sensores y puede intercambiar datos con los dispositivos portátiles. Posee la capacidad de conectarse a Internet por medio de un puerto Ethernet o un adaptador de Wi-Fi, con lo que se hace posible la conexión con un servidor remoto o la nube. La Raspberry Pi tiene un espacio de memoria limitado, pero esto no es un inconveniente pues se tiene pensado que por medio del elemento previamente mencionado, toda la información se almacene en una base de datos externa y no de forma local. Además, las instrucciones que se transmiten al dispositivo portátil se descargan periódicamente del repositorio, por lo que esto tampoco exige gran espacio de memoria.

Por otro lado, se trabajó con los módulos XBee serie 2 con antena de cable por dos motivos: esta serie presenta un rango aceptable en cuanto a la distancia que se puede cubrir (en condiciones de prueba en interiores), y que eran los que estaban disponibles para usarse inmediatamente. De ser posible, se utilizarían los XBee 900 pues como muestran las gráficas de la Figura 110 a la Figura 112, estos elementos tienen una mayor cobertura que los XBee serie 2. Además, si se tienen los recursos, se utilizaría una antena con conector RPSMA, pues estando al exterior y rodeados de metal, la intensidad de la señal se reduce considerablemente. Esto se debe a que la aplicación está orientada a que se coloquen sensores en cada calle y un router en cada dos o tres intersecciones. Cabe mencionar que los módulos Waspmote de Libelium se descartaron por su elevado costo.

El baud rate se fijó a 9600 para ejecutar las pruebas. Sin embargo, se busca incrementar este valor hasta 115200, pues así se tiene una velocidad de transmisión mayor, y a distancias mayores de 100 m la intensidad de la señal es similar en los baud rates que se analizaron, como se observa en la Figura 110. Obtener dicho baud rate dependerá de los dispositivos terminales, si son capaces o no de comunicarse a tal velocidad.

El programa del sistema de control fue desarrollado para atender tanto a los sensores como a los dispositivos portátiles. Esto implica que debe ser capaz de diferenciar los mensajes de cada unidad y procesarla de forma adecuada; almacenar la información proveniente del sensor y establecer una conexión con el dispositivo portátil. También se verifica la integridad de los datos en la comunicación con este último (por medio de verificación por redundancia cíclica), pues procesar un mensaje corrupto solo implica ocupar recursos innecesariamente.

La comunicación con el sensor es en una sola vía, pues esta unidad terminal envía sus datos cuando le es posible y no se envía ninguna notificación de entrega. Se considera que un paquete perdido no representa una pérdida significativa de información (al menos en esta fase de prueba); en caso sea una falla en el enlace, los usuarios lo notarían al no obtener información de esa región. En cambio, el dispositivo portátil requiere que se le envíe información o realizar una notificación de forma instantánea. Para brindar seguridad, es necesario establecer la comunicación por medio de un intercambio de mensajes; si estos no se realizan, no puede haber un intercambio de información entre los dispositivos.

La información que se intercambia con el dispositivo portátil se codifica para reducir el tamaño del paquete ya que la transmisión de información innecesaria se traduce en pérdidas, ya sea de potencia o en costos. Estas unidades terminales solo establecen conexión para realizar la acción que desean (recibir instrucción o enviar notificación) e inmediatamente terminan la sesión. Esto permite que los recursos se encuentren disponibles cuando hay más usuarios intentando conectarse. En caso la comunicación falle, ambos módulos tienen temporizadores o contadores que le indican cuando finalizar una conexión si no hay repuesta de la otra parte.

Por último, se intentó realizar un programa que fuese fácilmente integrable a cualquier sistema de cómputo que se encargue de procesar los datos obtenidos del sensor y entregar resultados para el dispositivo móvil. Esto significa que dicho sistema es el encargado de extraer la información de los sensores guardada en las bases de datos y generar las instrucciones para el dispositivo portátil que son colocadas en el repositorio. Cada red soporta 255 dispositivos portátiles y 255 sensores; y en cada una se debe reservar una dirección para el sistema de control. En el programa desarrollado, el sistema de control posee la dirección cero. El sistema de cómputo debe tener la localización del cada módulo terminal para saber a qué región pertenece cada conjunto de datos que analice y saber a dónde debe dirigir sus resultados, identificados por el número de sistema de control y el número de dispositivo móvil. (The Fan Club, s.f.)

H. CONCLUSIONES

Los módulos de comunicación seleccionados fueron los dispositivos XBee serie 2 en modo transparente con antena de cable por su rango de operación adecuado para la fase de prueba y su disponibilidad, resultando en una red mesh ZigBee inalámbrica que opera por medio de UART; además, estos dispositivos se adecúan a los requerimientos prácticos de la aplicación.

La unidad de control seleccionada fue la Raspberry Pi ya que posee los recursos necesarios para comunicarse por UART, almacenar los datos temporalmente y de forma local, y permite la conexión a Internet por medio de su puerto Ethernet o un adaptador Wi-Fi.

En esta fase de prueba, el sistema opera a un baud rate de 115200.

El programa del sistema de control permite conectar a 255 sensores, de los cuales solo recibe una cadena con la información a guardar en la base de datos, y no responde ninguna confirmación de entrega.

El programa del sistema de control permite conectar a 255 dispositivos portátiles, con los cuales debe establecerse una conexión a través de un intercambio de mensajes específicos para dar seguridad. Dependiendo del tipo de instrucción que la unidad terminal envíe, el módulo de control envía una instrucción según es solicitado o guarda la notificación recibida. La sesión se cierra al finalizar la transacción.

Para asegurar la integridad de los datos, se utiliza la verificación por redundancia cíclica CCITT XMODEM que provee las librerías de Python.

Las instrucciones a transmitir al dispositivo portátil se descargan de un repositorio en Internet, y estas se encuentran codificadas de tal manera que se reduce el largo del mensaje y se aprovechan mejor los recursos al no realizar transmisiones innecesarias.

I. RECOMENDACIONES

1. Se debe entender el funcionamiento del programa desarrollado para facilitar la integración con un sistema de cómputo, y conocer las limitaciones del mismo para ver en qué aspectos se puede mejorar.

2. Se debe evaluar si en el futuro existen módulos tanto de comunicación como de control a un costo accesible con mejores características que los actuales, para mejorar el alcance de la aplicación.

3. Se recomienda realizar pruebas en exteriores con diferentes tipos de antenas para determinar la que mejor se adapta al proyecto.

4. Se debe realizar un análisis del flujo de datos en el sistema, determinando los puntos que podrían convertirse en un cuello de botella y evitar que esto suceda.

X MÓDULO DE ASISTENCIA PORTÁTIL PARA AGENTE DE POLICÍA DE TRÁNSITO

A. INTRODUCCIÓN

El dispositivo de asistencia portátil, determina la ubicación geográfica del agente de policía de tránsito, recibe instrucciones de manera inalámbrica del sistema de medición y análisis de tránsito vehicular. Dichas instrucciones se despliegan en la pantalla, se reproducen mediante audio. Además, se permite al agente de policía de tránsito, reportar incidentes por medio de una pantalla táctil, con un menú intuitivo y fácil de activar.

El control del dispositivo de asistencia portátil se realiza mediante el uso de un microcontrolador. La ubicación geográfica del dispositivo se obtiene mediante un módulo de GPS, que indica la posición actual, altitud y latitud del policía que esté utilizando el dispositivo. Esta información se envía mediante un módulo de radiofrecuencias, conectado al dispositivo, que se comunica al sistema de detección de tránsito vehicular el cual responde mediante instrucciones para el policía.

Las instrucciones contienen: avenida o calle de interés de disminución de tránsito, tiempo de duración de paso vehicular, dirección que llevan los automóviles (norte, sur, este u oeste) y un indicador de afluencia vehicular. Al recibir la información anterior, el policía puede: omitir la instrucción sugerida y pedir una nueva instrucción y reproducir nuevamente la instrucción inicial. El dispositivo también cuenta con la opción de reportar cualquier incidente que se de en el área en la que se encuentra, para que así puedan ser considerado en el sistema de medición y análisis vehicular.

El dispositivo de asistencia portátil permitirá, por medio de una pantalla táctil y menú intuitivo y fácil de utilizar, una mejor gestión del tránsito vehicular sin afectar el desempeño laboral del policía.

B. OBJETIVOS

1. Objetivo general

Desarrollar un dispositivo portátil para un agente de policía de tránsito, que se comunique con el sistema de medición y análisis de tránsito vehicular e intercambie información efectivamente por medio de una interfaz de usuario para la gestión del flujo vehicular.

2. Objetivos específicos

a. Implementar un dispositivo portátil, como herramienta de trabajo para el agente de policía de tránsito.

b. Diseñar una interfaz gráfica para la interacción entre el agente de policía de tránsito y el sistema de medición y análisis de tránsito vehicular.

c. Comunicar inalámbricamente el dispositivo de asistencia portátil con el resto del sistema de medición de análisis de tránsito vehicular.

d. Localizar la ubicación del agente de policía de tránsito con el dispositivo de asistencia portátil, en las vías de tránsito, por medio de GPS.

e. Reproducir por audio las instrucciones sugeridas para el agente de policía de tránsito para la gestión de tránsito vehicular.

f. Permitir el reporte de incidentes de tránsito a través del dispositivo de asistencia portátil.

C. JUSTIFICACIÓN

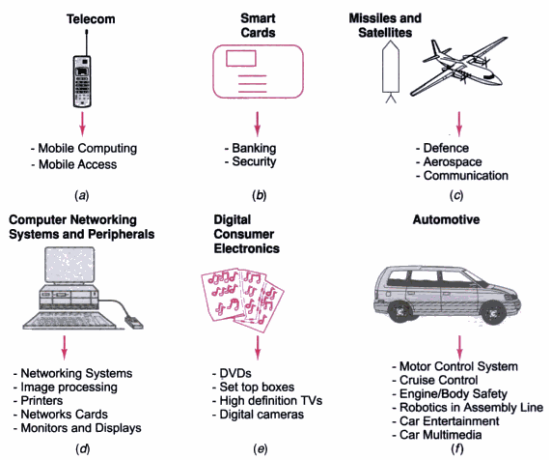
El agente de policía de tránsito será el usuario del sistema de medición y análisis del tránsito vehicular y requiere de un dispositivo para la comunicación con el sistema. El módulo de asistencia portátil es el encargado de permitir un intercambio de información eficiente, fácil de entender y activar, entre el agente de policía de tránsito y el sistema de medición y análisis de tránsito, permitiendo la recepción de instrucciones y el reporte de incidentes. Además, concede movilidad al agente de policía de tránsito para laborar sin limitar sus capacidades mientras porta el dispositivo. Y brinda instrucciones de acuerdo a la localización geográfica del agente de policía de tránsito.

D. MARCO TEÓRICO

1. Sistema embebido. Un sistema es una forma de organizar varias partes de un todo que realizan una tarea de acuerdo a un plan. (Kamal, 2008:9). No existe un consenso sobre la definición de un sistema embebido, la cantidad de dispositivos que están considerados dentro del término sistema embebido es demasiado variada. Un sistema embebido puede ser desde un microcontrolador de 8 bits hasta sistemas embebidos dentro del chip (System-on-chip SoC) y la utilización del diseño de circuitos VLSI.

Los sistemas embebidos nacieron cuando los microprocesadores se volvieron más baratos y pequeños; entonces salieron al mercado más productos con microprocesadores incrustados, computadoras ocultas, (Simon, 2005:12) que volvían más inteligentes a los productos: relojes digitales, elevadores, lavadoras, juguetes, discos duros, máquinas dispensadoras de comida, multímetros, etc.

Figura 121. Ejemplos de sistemas embebidos



(Kamal, 2008:22)

A diferencia de una computadora de escritorio, llamadas de propósito general, un sistema embebido contiene una computadora y todo un conjunto de elementos desarrollados para cumplir con una tarea específica. Además, debe cumplir con varias tareas que realiza paralelamente y responder a eventos externos, todo ello sin intervención humana y bajo restricciones de tiempo y desempeño.

a. Requerimientos de Sistema. La especificación del problema es la documentación inicial requerida. No debe incluir ninguna solución particular, sino explicar en detalle lo que el programa debe o no debe hacer. Zurell (2000:18) recomienda realizarlo desde el punto de vista del usuario. Otros autores, subdividen esta fase en otras más específicas.

- Análisis de requerimientos: tiene como entrada la información del usuario y como salida la definición de los requerimientos del sistema. Los datos que el usuario brinda son analizados desde el punto de vista ingenieril.

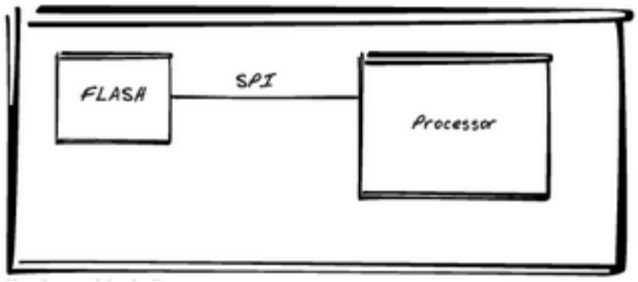
  - Requerimientos funcionales: conjunto de tareas que el sistema debe realizar.
  - Requerimientos no funcionales: restricciones de tiempo real, ejecuciones síncronas y asíncronas, desempeño el software, confiabilidad, restricciones de recursos (potencia, CPU, ancho de banda, volumen, memoria), nivel de autonomía (interacción con el ser humano).
  - Requerimientos de arquitectura: qué elementos deben ser software o hardware, distribución del sistema respecto a potencia, red, elementos mecánicos.
  - Requerimientos físicos: consumo energético, confiabilidad, geometría, volumen, robustez, peso.
  - Requerimientos de ciclos de vida: certificaciones, mantenimiento.

b. Arquitectura de un sistema embebido La arquitectura de un sistema embebido es una abstracción; es decir, una generalización del sistema donde no se muestran detalles de implementación del hardware o del software. En esta etapa los elementos representados interactúan entre sí. Estos elementos pueden contenerse dentro del sistema embebido o ser externos a él, pero importantes en su comportamiento (Noergaard, 2012:24). A través de una arquitectura de un sistema embebido se logra resolver retos de implementación o funcionalidad en etapas tempranas del proyecto.

Un sistema embebido contiene múltiples detalles con dependencias que deben ser reconocidos. Luego de establecer los requerimientos del sistema, se debe empezar a planificar la arquitectura de software (de ser necesaria) y la arquitectura de hardware. Los sistemas embebidos dependen de sobremanera de su hardware. Para cumplir con todos los requerimientos se utilizan diagramas de bloques de sistemas.

1) Diagrama de bloques. Se diseña orientado a objetos, cada componentes adjunto al procesador es un objeto y se comunican entre ellos. En esta etapa los detallas sobre, por ejemplo, qué marca de memoria se utilizará y sus especificaciones técnicas, no son importantes.

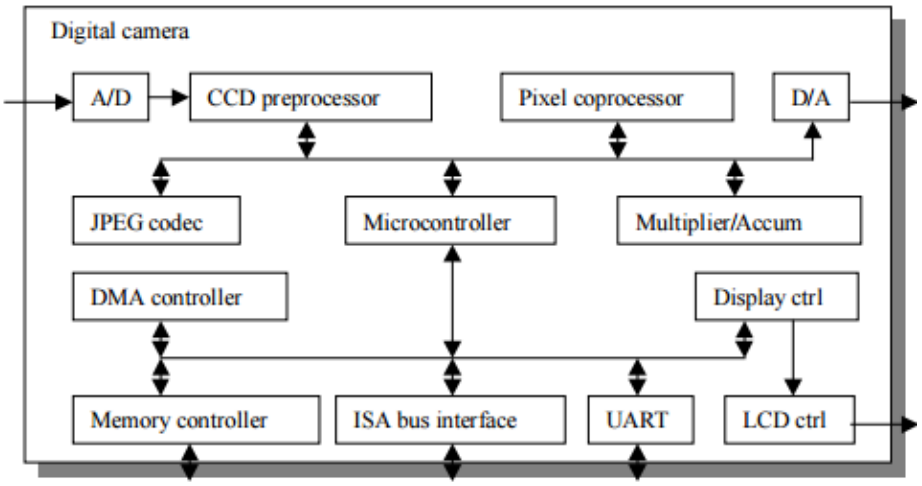
Figura 122. Ejemplo de diagrama de bloques de sistemas.



La tendencia general de los sistemas embebidos es la de considerar el hardware y el software coexistiendo y no como dos dominios distintos. Esto se debe a que los sistemas embebidos son sensibles a las métricas de diseño: costo, desempeño, potencia, flexibilidad, portabilidad, durabilidad, seguridad, confiabilidad, tamaño de memoria de programa, etc. (Vahid, 1999:3).

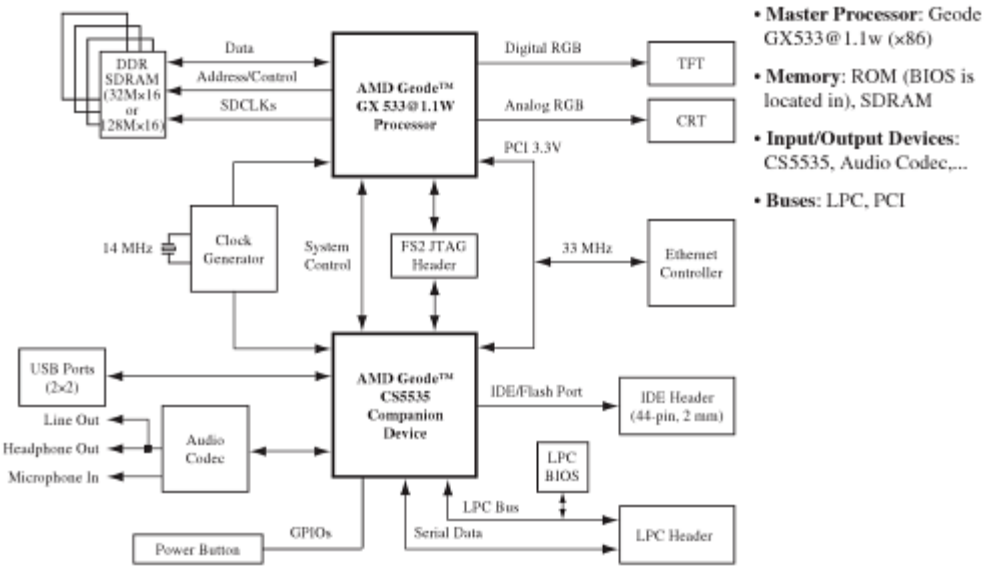
Se muestran un ejemplo específico a continuación.

Figura 123. Diagrama de bloques del sistema de una cámara digital



(Vahid. 1999:6)

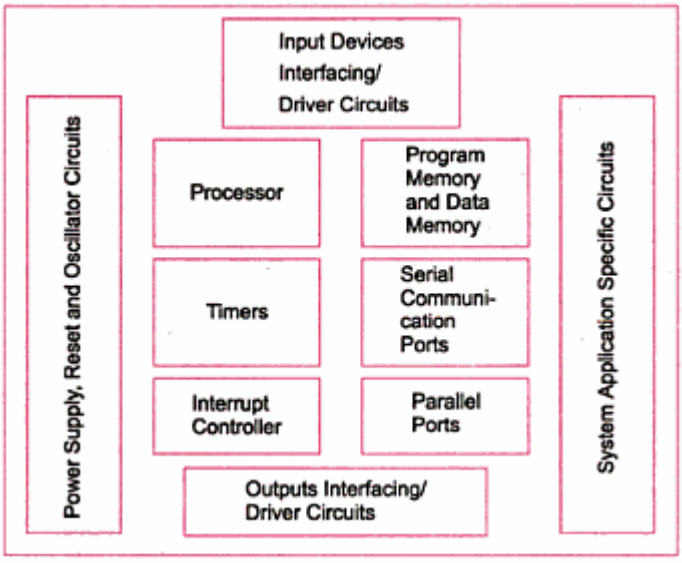
Figura 124. AMD/National Semiconductor x86 placa de desarrollo



(Noergaard, 2012:95)

De una manera más general, Raj Kamal (2008:10) presenta los componentes de un sistema embebido.

Figura 125. Componentes de un sistema embebido



(Kamal, 2008:10)

Aunque en la

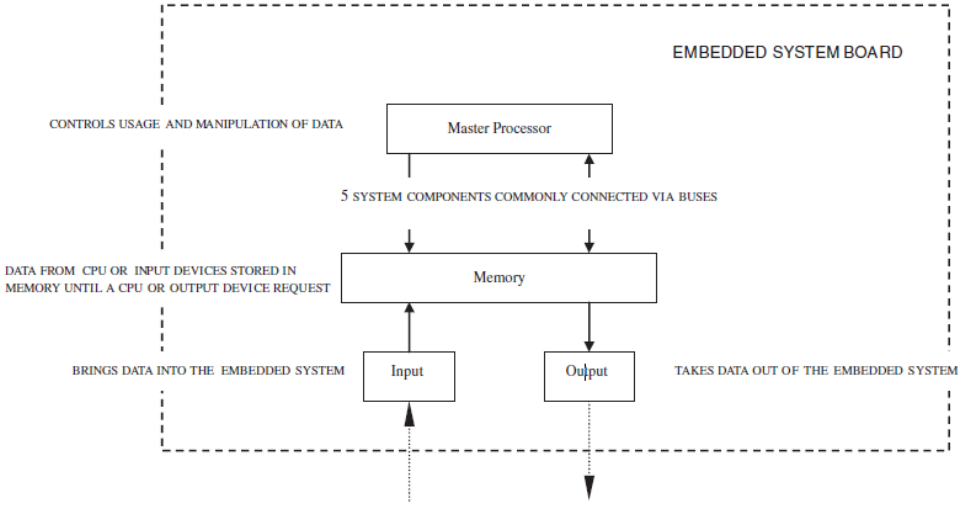
Figura 123 y Figura 124 se muestra un microcontrolador como unidad de procesamiento, existen más opciones que deben ser consideradas. Cada una presenta ventajas, desventajas y desafíos de diseño.

Una decisión importante será la de determinar cuáles elementos serán implementados en hardware y cuáles serán implementados en software. Sin embargo, primero deben especificarse todos los elementos. Algunos criterios los brinda Noergaard (2012:540):

- Costo: ¿la compra, integración y desarrollo del elemento encaja en las restricciones de costo?
- Tiempo al mercado: ¿el elemento cumple con los requerimientos de tiempo de desarrollo?
- Desempeño: ¿el elemento será lo suficientemente rápido para la satisfacción del usuario y/o los demás elementos?
- Herramientas de desarrollo y depuración de errores (debugging): ¿qué herramientas de hay disponibles que harán el diseño y desarrollo más fácil y rápido?

c. Arquitectura de Hardware. En el nivel más alto, los componentes de hardware pueden clasificarse en: unidad central de procesamiento (CPU), memoria, dispositivos de entrada, dispositivos de salida y buses.

Figura 126. Componentes de hardware de un sistema embebido



Existen distintas opciones como unidades de procesamiento:

- Procesador de propósito general (GPP)
  - Microprocesador
  - Procesador embebido

- Procesador con set de instrucciones específicas para la aplicación (ASIP)
    - Microcontrolador
    - Microcontrolador embebido
    - Procesador de señales digitales (DSP) y procesador de media
    - Procesador de redes
- Procesador de propósito único
    - Coprocesador
    - Acelerador
    - Controlador
- GPP o ASIP integrado en un ASIC, circuito VLSI o un arreglo de compuertas programable (FPGA)
    - ASIC
    - FPGA
- Procesador específico para la aplicación (ASSP)
- Múltiples procesadores o procesadores multinúcleo

Cada una de las opciones mencionadas con anterioridad posee sus cualidades y la elección se basa en los requerimientos de la aplicación. Algunos de los requerimientos que generalmente determinan la elección de una unidad de procesamiento son: nivel computacional requerido, requerimiento de procesamiento de señales, manufactura, costo de la ingeniería para el desarrollo, tiempo de desarrollo. (Kamal, 2008:26)

1) Microprocesador. La unidad central de proceso trae las instrucciones de un conjunto de instrucciones de propósito general que incluyen instrucciones para transferencia de datos, operaciones aritméticas y lógicas, de stack, de entrada y salida, supervisión. Puede poseer memoria caché, unidad de operaciones de punto flotante, pipelining, etc.

2) Microcontrolador. Es un microprocesador con memoria y otros hardwares dedicados en él: controlador de acceso directo a memoria (DMA), convertidores analógico-digital (ADC), circuitos para modulación de ancho de pulso (PWM), etc. Tiene limitadas sus capacidades computacionales.

3) Procesadores de propósito único. Estos procesadores poseen sistemas para aplicaciones específicas como: coprocesadores de punto flotante, compresión JPEG, compresión MPEG, controladores de periféricos, etc.

4) Utilización de circuitos VLSI como unidad de procesamiento

a) Circuito integrado específico para la aplicación (ASIC). Se realizan utilizando lenguajes descriptores de hardware y usan herramientas con circuitos VLSI. Este tipo de unidad de procesamiento no posee un código que ejecutar, ni instrucciones, el circuito descrito y sintetizado realiza la tarea requerida.

b) Núcleos con propiedad intelectual (IP). Existen módulos dentro del diseño VLSI que pueden pertenecer a otras compañías que los venden como cajas negras. Sólo el dueño y diseñador posee la propiedad intelectual del dispositivo, su sintetización e implementación. Por ejemplo, módulos que realicen multiplicaciones de matrices, implementaciones de protocolos HTTP o FTP, etc.

c) FPGA (arreglo de compuertas lógicas programables). Consiste en un gran conjunto de compuertas lógicas que pueden interconectarse según la necesidad y así implementar una tarea. Algunos ejemplos de tareas implementadas con FPGA son: transformada de Fourier y transformada inversa de Fourier, compresión y de compresión, encriptación y desencriptación.

d) Microprocesador embebido. Significa incluir el diseño de un microprocesador dentro del diseño VLSI.

e) Microcontrolador embebido. Significa incluir el diseño de un microcontrolador dentro del diseño VLSI

f) Procesador de señales digitales embebido. Significa incluir el diseño de un procesador de señales digitales dentro del diseño VLSI, por ejemplo el filtrado, cancelación de ruido, encriptación.

d. Arquitectura de Software. La arquitectura de software organiza los componentes de un sistema de software, algoritmos y estructuras de datos para integrarlos y cumplir con los requerimientos del sistema (Garlan, 1994:5). Está basada en la abstracción y es el más alto nivel de abstracción en el área del software. Debe seleccionarse según los requerimientos y restricciones del sistema. Los distintos estilos de arquitectura de software, son un conjunto de patrones que organizan al software y resultan en un sistema con características conocidas (Medvidovic, 2003:3).

Un sistema embebido generalmente administra múltiples tareas paralelamente con restricciones de tiempo real y desempeño. Esto quiere decir que además de la cantidad de tareas con las cuales debe cumplir existen prioridades y cierto tiempo para atenderlas. La tarea a realizar puede requerir ser atendida inmediatamente (prioridad) pero no requiere un resultado próximamente (restricción de tiempo) (Simon,

2005:69). Además, dependiendo del contexto del sistema embebido, el desempeño puede referirse al uso de memoria, CPU, batería, ancho de banda, escalabilidad, confiabilidad, portabilidad, seguridad, etc.

Habitualmente, la calidad de un sistema embebido está determinada por qué tan bien gestiona todos los requerimientos del sistema (Cook, 2008:1). Al mismo tiempo, No todas las arquitecturas de software pueden ser implementadas en todos los sistemas. Un sistema embebido interactua con el exterior a través de sus actuadores y sensores (Hardware). La arquitectura de software debe poder ser implementada en la arquitectura de hardware. Uno de los principales inconvenientes en la elección de una arquitectura de software en un sistema embebido es que el software se desarrolla en paralelo con el hardware y muchas veces este aún no existe (Medvidovic, 2003:2).

Finalmente, no existe una arquitectura de software ideal; cada una posee ventajas y desventajas que deben ser evaluadas. Algunos consejos en la literatura son: utilizar la arquitectura más simple que cumpla con los requerimientos del sistema; si se poseen restricciones de tiempos de respuesta utilizar un sistema operativo de tiempo real (posteriormente descrito); y no descartar la idea de realizar híbridos (mezclar arquitecturas) (Simon, 2005:79). Algunas de las arquitecturas más utilizadas son la de cliente-servidor, pipeline, orientada a objetos, basada en eventos, sistemas por capas.

1) Round robin. El software se organiza de tal forma que existe un único ciclo principal donde las tareas son atendidas. No existen interrupciones. Se verifica cada tarea una a una y se determina si es necesario atenderla; si es así, es atendida hasta ser finalizada. Al finalizar todas las tareas, vuelve a empezar el ciclo principal.

Es la solución más simple y el intercambio de información entre tareas es sencillo. Sin embargo, la peor latencia del sistema puede ocurrir cuando el tiempo de ejecución sea el tiempo del ciclo principal completo. Además, en caso de ser necesario agregar funcionalidades puede no cumplir con las restricciones de tiempo debido a la latencia del sistema o puede no cumplir únicamente cuando la latencia del sistema es el peor de los casos, haciendo más difícil la detección del error.

```
Void main (void) {
      While (TRUE) {
            If (tarea_A requiere atención)
                  Servicio a tarea A
            If (tarea_B requiere atención)
                  Servicio a tarea B
            If (tarea_C requiere atención)
                  Servicio a tarea C
```

```
            ...
      }
}
```

2) Round robin con interrupciones. Al agregar interrupciones a la arquitectura round robin se obtiene una mejora considerable. Las tareas con mayor urgencia son atendidas vía interrupciones; pero si nada urgente sucede, entonces se continua con las tareas en el main loop (round robin). Debido a que las interrupciones tienen prioridad sobre el ciclo principal, la latencia del sistema a las tareas con interrupciones es mejorada y el programa continúa siendo relativamente sencillo. Además, existen arquitecturas de microcontroladores que permiten utilizar prioridades a las interrupciones obteniendo como resultado un mejor manejo en el tiempo de respuesta a ciertas tareas según su prioridad.

Por otro lado, el problema de los datos compartidos emerge. Esto quiere decir que si hay datos que están siendo cambiados o utilizados en el ciclo principal y una interrupción que utiliza esos datos es ejecutada, si no se toman en cuenta las debidas precauciones, los datos están o serán corruptos. Los datos compartidos deben ser validados. Por lo tanto, hay que utilizar técnicas para lidiar con los problemas de los datos compartidos.

```
ISR_A (void) {
      Flag_tareaA
}

ISR_B (void) {
      Flag_tareaB
}

ISR_C (void) {
      Flag_tareaC
}

Void main (void) {
      While (TRUE) {
            If (Flag_tareaA)
                  Servicio a tarea A
            If (Flag_tareaB)
                  Servicio a tarea B
            If (Flag_tareaC)
                  Servicio a tarea C
```

```
		...
	}
}
```

3) Cola de espera de funciones. Esta arquitectura consiste en que si una tarea requiere ser atendida, entonces la función que la atiende es agregada a una cola de espera por medio de un puntero que apunta hacia la función. La cola es atendida en el ciclo principal, sino está vacía. La ventaja de esta arquitectura es que la tarea es agregada a la cola según su prioridad y no debe ser agregada hasta el final de ella. Sin embargo, estrictamente hablando, la tarea debe terminar de ser atendida para atender la siguiente y los elementos que se encuentran al final de la cola pueden nunca ser atendidos si siempre hay un elemento adelante de ellos en la cola (latencia infinita).

```
ISR_A (void) {
	...
	Agregar a cola función A
}

ISR_B (void) {
	...
	Agregar a cola función B
}

ISR_C (void) {
	...
	Agregar a cola función C
}

Void main (void) {
	While (TRUE) {
		While (si la cola no está vacía) {
			Llamar primera función en cola
		}
	}
}

Void función A (void) {
	...
```

```
}
Void función B (void) {
       ...
}
Void función C (void) {
       ...
}
```

4) Sistema operativo de tiempo real. En esta arquitectura las interrupciones se encargan de las tareas con más prioridad y avisan que hay que atenderlas. No puede utilizarse datos compartidos debido a la naturaleza de esta arquitectura. El sistema operativo sabe acerca de las tareas a realizar y es capaz de atender la más urgente en cualquier momento (no existe un ciclo principal donde se dice que tarea debe realizarse, esto está a cargo del sistema operativo). Además, el sistema operativo puede detener momentáneamente una tarea en medio de su ejecución para atender a otra.

Esta arquitectura agrega complejidad, consume memoria y recursos valiosos para el sistema embebido. Pero, se encarga de gestionar las tareas a realizar y generalmente agregar nuevas funciones o tareas de menor prioridad no afecta a las tareas de mayor prioridad.

```
ISR_A (void) {
       ...
       Señal de tarea A
}

ISR_B (void) {
       ...
       Señal de tarea B
}

ISR_C (void) {
       ...
       Señal de tarea C
}

Void función A (void) {
       ...
}
Void función B (void) {
```

```
        ...
}
Void función C (void) {
        ...
  }
```

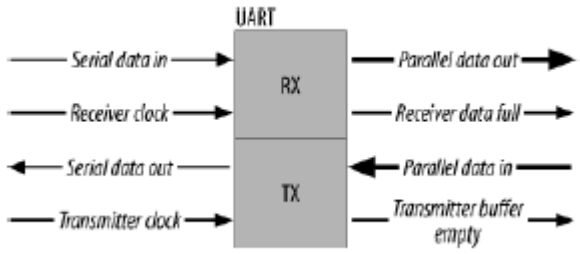
e. Módulos

1) Comunicaciones seriales. Las comunicaciones seriales se utilizan cuando no es práctico utilizar comunicación paralela. Generalmente, esto se debe a limitantes físicas, de recursos disponibles y de costo. Para los sistemas embebidos la comunicación serial es la forma más barata y fácil de comunicarse entre dispositivos. (Catsoulis, 2005:266).

a) UART. La interfaz más simple de comunicación serial es la UART (transmisor receptor universal asíncrono). La comunicación serial se refiere a la transferencia de datos a través de un único cable. Es llamad asíncrona por qué no hay un reloj dentro de la información transmitida. Todos los módulos de comunicación serial convierten datos paralelos en una cadena serial de caracteres y viceversa.

La funcionalidades de una interfaz UART están divididas en dos secciones: el receptor (Rx), encargado de convertir la cadena de bits serial en datos paralelos para el procesador y el transmisor (Tx), encargado de convertir los datos paralelos en una cadena de datos seriales y organizarlos para transmitirlos. Además, la interfaz UART brinda información de su estatus, por ejemplo: hay datos recibidos no leídos, no hay datos pendientes de transmisión, detección de errores (bit de paridad).

Figura 127. Bloque funcional de la interfaz UART



(Catsoulis, 2005:267)

Asociado con la comunicación serial, uno de los mayores problemas es la interpretación de la información recibida debido a detectar dónde termina un bit y comienza el siguiente. Por ejemplo “00” o “111”. El problema se resuelve compartiendo una señal de reloj en ambos dispositivos (volver síncrona la comunicación). Sin embargo, esto requiere otro hilo de comunicación entre el transmisor y el receptor.

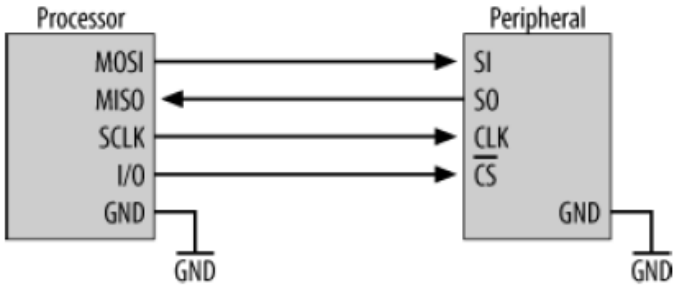
Los dispositivos con comunicación asíncrona, detectan el reloj a partir de la información transmitida. Utilizan un bit de inicio y uno o dos bits de fin. Se detecta el bit de inicio, se muestrea la información enviada y luego se espera el bit de fin (Tomasi, 2003, 549).

b) SPI. La interfaz SPI (interfaz síncrona de periféricos) es barata y sencilla de utilizar. Ya que es una comunicación síncrona, existe una señal de reloj de sincroniza la detección de los bits recibidos y transmitidos. Generalmente, la comunicación SPI se utiliza a cortas distancias, con otros microcontroladores y periféricos, en el mismo circuito impreso o cercano. A diferencia de la comunicación UART, que es utilizada para relativamente grandes distancias; SPI utiliza altas velocidades, en cortas distancias, con un mínimo de pines (Barnett, 2004:138).

Utiliza una modelo de esclavo-maestro. La señal de reloj es producida por el maestro. El esclavo utiliza la señal de reloj para sincronizar la información recibida. Muchos dispositivos pueden ser conectados a la misma interfaz SPI y el maestro selecciona un dispositivo para recibir la información.

Una interfaz SPI posee las siguientes señales: maestro salida esclavo entrada (MOSI), maestro entrada esclavo salida (MISO), reloj serial SCLK o CLK y selección de chip CS.

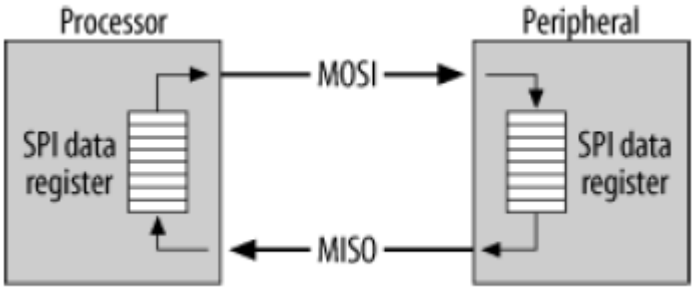
Figura 128. Interfaz básica SPI



(Catsoulis, 2005:242)

Maestro y esclavo poseen un shift register. El maestro es quien debe iniciar la comunicación. Mientras el maestro transmite la información dentro del shift register a través del pin MOSI, el esclavo introduce el bit recibido en su shift register y transmite la información a través del pin MISO. Al final de la comunicación, los contenidos de los registros son intercambiados de maestro a esclavo.

Figura 129. Transmisión SPI



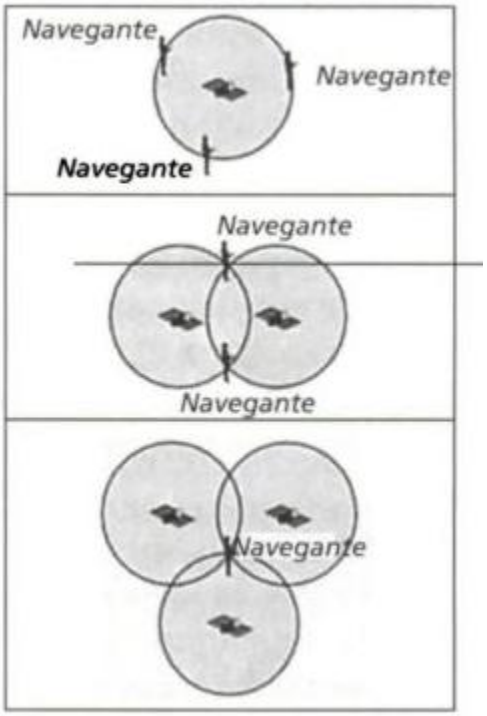
(Catsoulis, 2005:243)

2) PWM. La técnica conocida como modulación por ancho de pulso (PWM), consiste en la generación de una señal cuadrada, con un ciclo de trabajo configurable. El ciclo de trabajo es la relación entre el tiempo en alto y el tiempo en bajo de la señal cuadrada.

Figura 130. Señal PWM con ciclo de trabajo de aproximadamente 10%

3) GPS. El sistema de posicionamiento global (GPS) es un conjunto de satélites utilizados para determinar la posición de un elemento las 24 horas del día. Este sistema consiste en 24 satélites que circundan el planeta Tierra y envían señales de radio frecuencia a la superficie. Un receptor GPS permite recibir la señal de estos satélites e interpretarla para calcular su posición. Este tipo de tecnología no funciona en selvas o bosques frondosos, en general, las señales GPS no pueden penetrar grandes obstáculos. Para poder obtener la ubicación por medio del sistema de posicionamiento global, es necesario poseer la cobertura mínima de 4 satélites.

Figura 131. Ejemplo de cálculo de la posición mediante triangulación



(Letham, 2001:12)

Para este ejemplo se utiliza el cálculo de la posición mediante triangulación. Se necesitan tres satélites. Se calcula la distancia entre el receptor y el primer satélite; esto significa que el navegante se encuentra dentro de algún lugar que rodea el satélite 1. Luego, el satélite mide la distancia al satélite 2; esto significa que el navegante se encuentra dentro de la intersección de alguno de los dos circuitos. Finalmente, el receptor mide la distancia al satélite 3. Por lo tanto, solo existe un punto donde esto ocurre. (Letham, 2001:12)

A diferencia del ejemplo, el sistema GPS utiliza intersecciones de esferas para determinar: latitud, longitud y altitud. Con 3 satélites basta, el cuarto satélite es utilizado para la sincronía de tiempo entre los satélites. Además del tiempo, es importante que el satélite conozca su propia localización y la de los demás.

4) Pantalla TFT. Es una matriz activa de cristal líquido (LCD) controlada por un arreglo de transistores de película delgada. Cada pixel utiliza un capacitor y un transistor. A todos los pixeles se les asigna una dirección basadas en columnas y filas.

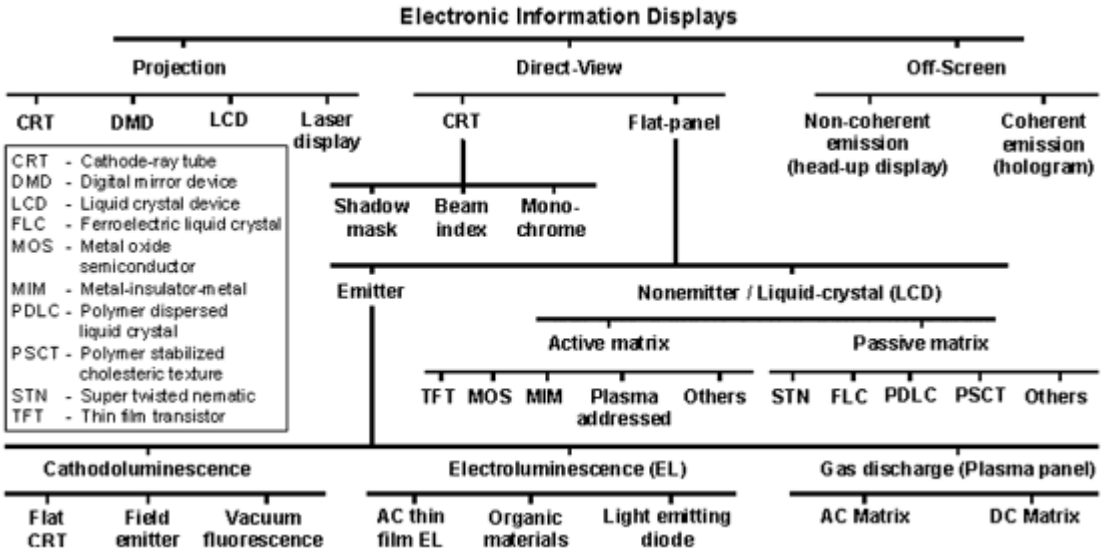
Dentro de los dispositivos electrónicos para mostrar información entra en la categoría de pantalla, de visión directa; esto quiere decir que la luz se produce directamente sobre la pantalla con el objetivo de operar mejor en lugares con luz muy brillante (exteriores) y poseen mayor iluminación que las pantallas

proyectadas. Las pantallas LCD se caracterizan también por su bajo consumo energético, bajo voltaje de operación y tiempo de vida prolongado (Sears, 2007: 210).

Las pantallas LCD consisten en dos vidrios paralelos con líneas microscópicas en la superficie interna y un cristal líquido entre ellas. El cristal líquido no emite luz propia, así que debe iluminarse (en este caso directamente). Mediante modulación eléctrica los cristales dentro del líquido se posicionan creando contrastes y se vuelven traslúcidos u oscuros.

En el caso de matrices activas, para garantizar buenas resoluciones, se requiere un gran número de transistores. Para una resolución de 1024x768 pixeles, se requieren 2.36 millones de transistores. La latencia de respuesta es aproximadamente 20 – 30 ms. Producir el color negro técnicamente es imposible en una pantalla TFT. Finalmente, en este tipo de pantallas se produce una distorsión al ser vista desde un ángulo (Sears, 2007: 210).

Figura 132. Tecnologías de pantallas electrónicas para mostrar información



(Sears, 2007:210)

5) MP3. Es un formato de compresión de audio digital patentado que utiliza un algoritmo que consigue un tamaño más pequeño en los archivos y provee alta calidad de audio. Fue creado por Moving Pictures Experts Group (MPEG), posee los estándares ISO/IEC 11172-3 y 13818-4. En el dispositivo diseñado en este trabajo escrito se utilizó la reproducción de audio con archivos MP3 a través de un módulo que contiene un decodificador de la compresión mencionada previamente.

6) Micro SD. Es un tipo de memoria no volátil desarrollada por SanDisk. Se diferencia por su tamaño, 15x11x11mm. Posee 8 pines, 165 mm^3, 0.258 g. Existen varios estándares sobre los cuales ha sido diseñada: SDC (Secure Digital Memory) estándar para las memorias externas para dispositivos móviles y MMC (Multi Media Card). Esto significa que son un arreglo de memoria tipo flash, con un controlador dentro de ellas. La memoria puede borrarse, leerse y escribirse. El intercambio de información entre la memoria y otro dispositivo se realiza por bloques de memoria de 512 bytes. Poseen formato FAT12/16. En el dispositivo diseñado en este trabajo escrito se utilizó una memoria micro SD para almacenar archivos MP3 debido a su volumen pequeño y gran capacidad para almacenar datos.

2. Lenguajes de programación

a. Ensamblador. El lenguaje ensamblador es un lenguaje de programación de bajo nivel. Cada arquitectura de computadora posee su propio lenguaje ensamblador, a diferencia de los lenguajes de alto nivel que son portátiles entre dispositivos de distinta arquitectura y luego pueden ser compilados en otra arquitectura. En este lenguaje, generalmente una instrucción en el lenguaje ensamblador, corresponde a una instrucción de la máquina.

Hasta hace unos años la forma más eficiente de programar un sistema embedidos era con el lenguaje ensamblador debido a la correspondencia directa entre instrucciones del lenguaje e instrucciones de máquina. El lenguaje ensamblador asegura que el código sea el mínimo y más eficiente, generalmente requisitos en un sistema embebido (Paolo, 2014:96).

b. C. Aunque existen otros lenguajes de programación (Pascal y Basic), el lenguaje C es el más utilizado en el desarrollo de sistemas embebidos debido a ser flexible. Esto significa que permiten al desarrollador que su programa sea “legible” (readable) en comparación con el lenguaje ensamblador, sin perder el control de lo que se realiza.

La principal mejora que se ha tenido que realizar a los compiladores C, es la optimización de código según la arquitectura de la unidad de proceso en el sistema embebido. Con el aumento en la complejidad de los sistemas creados y la sensibilidad a los costos, el lenguaje C con su portabilidad y legibilidad garantiza completar los requerimientos de los dispositivos en un tiempo al mercado menor. (Paolo, 2014:97).

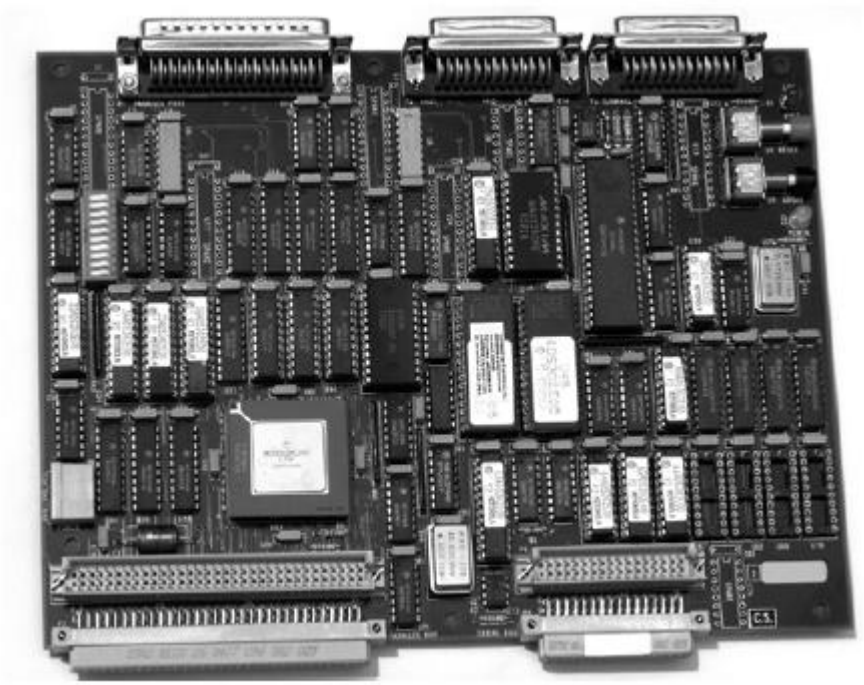
c. Java. En la actualidad hay sistemas embebidos que utilizan lenguajes orientados a objetos, como Java. Desarrollar entorno a Java asegura la confiabilidad y seguridad en el entorno computacional, además es completamente portátil y reusable lo cual reduce los tiempos y costos de desarrollo. Como desventajas, Java no posee comportamientos determinísticos respecto a las restricciones de tiempo. Sin embargo, se está trabajando en estandarizar ciertos procesos para asegurar operaciones en tiempo real (Real-Time Java Expert Group y Java Consortium) (Kleinjohann, 2013:1966).

3. Herramientas de desarrollo. Las herramientas de desarrollo son un factor importante en la selección del procesador. Algunos de los aspectos que deben verificarse son los compiladores, debugger, profiler, librerías disponibles, comunidad en el internet, soporte técnico, simulador, etc.

a. Hardware

1) Panel de desarrollo. Un panel de desarrollo contiene el microcontrolador que se utilizará en el sistema embebido, programador, debugger en circuito, cables y herramientas para programar el microcontrolador, diagramas esquemáticos del panel, puertos de entradas y salidas, interfaces para comunicarse con distintos protocolos y todo tipo de herramientas que puedan ayudar al desarrollador a probar el sistema.

Figura 133. Ejemplo de panel de desarrollo.



2) Depurador en circuito. Es un circuito que permite comunicarse con el microcontrolador y determinar cuál es el valor de los registros, memoria de datos y otros valores del dispositivo. Sirve para detectar errores dentro del programa o durante la ejecución. Los depurardores en circuito se diferencian entre sí porque los datos obtenidos se encuentren más próximos a la realidad respecto al tiempo y las características eléctricas de las señales o estados del microcontrolador (Catsoulis, 2005:203).

b. Software

1) Compilador. El microcontrolador ejecuta instrucciones de máquina a través de las instrucciones en lenguaje ensamblador. Sin embargo, ninguna computadora comprende el lenguaje ensamblador directamente. Las instrucciones deben ser compiladas (convertidas al opcode apropiado), entre otras cosas, este trabajo lo realiza el compilador, específicamente el ensamblador.

Los compiladores de alto nivel (por ejemplo, de lenguaje C) poseen información del procesador y pueden convertir el código de alto nivel en lenguaje ensamblador. Un buen compilador de alto nivel debe crear un código en lenguaje ensamblador optimizado para obtener el mejor desempeño y el menor tamaño (Catsoulis, 2005:51).

2) Ambiente de diseño integrado. Por sus siglas en inglés, IDE. En este tipo de programas posee el editor del código de programa, interfaces para detección de errores, simuladores, compiladores y distintas herramientas que le sirven al desarrollador. Para utilizar un IDE es necesario una computadora de escritorio, desde la computadora de escritorio se programa el microcontrolador a través de un puerto de comunicación y un circuito especial que sirve para guardar el código de programa en la memoria del dispositivo (programador).

3) Otras herramientas de desarrollo. Existen otras herramientas que pueden ayudar al desarrollador, entre ellas: administrador de interrupciones, estadísticas sobre el programa y la utilización de los recursos, generadores de código a través de diagramas de máquinas de estados finitos, calculadoras de tiempos de ejecución, etc.

E. METODOLOGÍA

1. Planeación

a. Requerimientos

1) Entrevistas. Se planeó entrevistar a ejecutivos y agentes de la policía de tránsito de la ciudad de Guatemala. De esta manera se pretende obtener ambos puntos de vista y proponer una oferta que cumpla con los requerimientos más importantes. La entrevista se realizó en base a los siguientes objetivos:

- Desarrollar una entrevista para ejecutivo.
- Desarrollar una entrevista para agente.
- Establecer los requerimientos claves del producto a desarrollar.
- Establecer una oferta y la respuesta de los entrevistados al producto.
- Comparación del producto con alternativas.

2) Análisis y obtención de requerimientos. El objetivo de esta etapa es establecer los requerimientos funcionales, no funcionales, de arquitectura, físicos y de ciclos de vida según las entrevistas realizadas.

3) Selección de arquitectura de hardware. Se selecciona la tecnología a utilizar para el procesamiento de la información y cumplir con los requerimientos del sistema de manera general, basados en la abstracción de los componentes necesarios y no en cómo serán implementados. Los criterios a utilizar son el nivel computacional requerido, manufactura, costo, tiempo de desarrollo, herramientas disponibles y complejidad.

Mencionados con anterioridad, algunas de las opciones disponibles son:

1. Procesador de propósito general (GPP)
    a. Microprocesador
    b. Procesador embebido
2. Procesador con set de instrucciones específicas para la aplicación (ASIP)
    a. Microcontrolador
    b. Microcontrolador embebido
    c. Procesador de señales digitales (DSP) y procesador de media
    d. Procesador de redes

3. Procesador de propósito único
    a. Coprocesador
    b. Acelerador
    c. Controlador
4. GPP o ASIP integrado en un ASIC, circuito VLSI o un arreglo de compuertas programable (FPGA)
    a. ASIC
    b. FPGA
5. Procesador específico para la aplicación (ASSP)
6. Múltiples procesadores o procesadores multinúcleo

b. Selección de herramientas de desarrollo

1) Hardware de desarrollo. Se evaluarán las distintas opciones disponibles según la documentación, herramientas integradas, comunidad en internet, soporte técnico, utilización de estándares, disponibilidad en general, herramientas para detección de errores, simulación, etc.

2) Software de desarrollo. Se evaluarán las distintas opciones disponibles y evaluarán según la documentación, herramientas integradas, comunidad en internet, soporte técnico, utilización de estándares, disponibilidad en general, librerías, etc.

c. Selección de metodología para el desarrollo. El dispositivo de asistencia portátil posee distintas tareas que atender para cumplir con su funcionamiento. Las tareas se separan y organizan en distintos módulos. Se trabajará cada módulo por separado verificando su funcionamiento y analizando las capacidades disponibles y útiles para el proyecto. Esta parte de la metodología funciona como una prueba unitaria y sirve para asegurar que el módulo utilizado funciona correctamente y es capaz de implementarse y cumplir con los objetivos y requerimientos en el dispositivo de asistencia portátil.

Se desarrolla un programa en lenguaje de programación de alto nivel en un microcontrolador que cumple con la tarea de hacer operar el módulo cómo se supone trabajará en el dispositivo de asistencia portátil (implementación modular). Además, se utiliza una arquitectura con desempeño relativamente bajo, pero diseñada para ser escalable a otros microcontroladores de alto desempeño, con compatibilidad en los pines de salida, registros y hardware de módulos periféricos. Después, se implementa el mismo funcionamiento en el dispositivo de asistencia portátil (implementación integral), que utiliza un microcontrolador de alto desempeño; para este paso el software y hardware deben modificarse para ser compatibles. El proceso se repite para cada submódulo, hasta dar por terminada la implementación de todas las tareas que debe realizar el dispositivo de asistencia portátil.

Se realizarán entregas quincenales, basados en metas y ponderadas por igual. En cada entrega se evaluará el funcionamiento y, si aplica, la compatibilidad al implementar el submódulo con el dispositivo de asistencia portátil.

Divididos en una secuencia de pasos, la metodología a seguir para el desarrollo del proyecto es:

- División del sistema en módulos según sus requerimientos funcionales (diagramas de bloques).
- Desarrollo de software y hardware para el correcto funcionamiento de uno de los módulos por separado.
- Se desarrollan pruebas unitarias para verificar que la operación de cada módulo por separado cumpla con los requerimientos del sistema (hardware y software).
- Se modifica el software y hardware para que cada módulo por separado sea compatible con su integración con el sistema entero.
- Se realizan pruebas verificando el correcto funcionamiento del sistema y que se cumplan con los requerimientos.

d. Selección de arquitectura de software. Según los requerimientos del sistema y la arquitectura de hardware, debe seleccionarse la arquitectura de hardware más simple que cumpla con las expectativas. Sin descartar la posibilidad de realizar híbridos. Algunas de las arquitecturas disponibles y mencionadas con anterioridad son:

- Round robin
- Round robin con interrupciones
- Cola de espera de funciones
- Sistema operativo de tiempo real

e. Cronograma. Se creará un cronograma según la selección de metodología para el desarrollo, arquitectura de hardware, arquitectura de software, herramientas de desarrollo, tiempo disponible para el desarrollo del sistema embebido y objetivos.

2. Ejecución

a. Requerimientos

1) Entrevistas. Las entrevistas realizadas se adjuntan completas en el Anexo A.

a) Objetivos

- Conocer sus estrategias
- Planes u objetivos a corto plazo.
- Cómo funciona el sistema que actualmente gestiona del tránsito vehicular.
- Con qué estudios eligen los tiempos y la distribución de agentes.
- Se posee con planes en caso de emergencias controlables.
- Comprender los métodos de trabajo de los agentes.

b) Entrevista prototipo para ejecutivo

- ¿Qué estrategias usan para agilizar el tráfico?
- ¿Tienen algún centro de monitoreo del tráfico?
- ¿Cómo distribuyen a los agentes de PMT en la ciudad?
- ¿Cómo monitorean o evalúan el trabajo de los agentes de PMT?
- ¿Cómo capacitan a los agentes de PMT? O Cuénteme acerca del proceso de entrenamiento de los agentes de la PMT. ¿Qué buscan en un agente? ¿Qué habilidades debe tener?
- ¿Manejan algún programa de incentivos para los agentes de PMT?
- ¿Qué hace un buen agente PMT? ¿Puede pensar en una persona específica que sea un buen agente de PMT? ¿Qué características tiene?
- A parte de los agentes de PMT y el suyo, ¿qué otros puestos de trabajo hay en la PMT? ¿Qué hacen?
- Si pudiera inventar una nueva plaza, ¿cuál sería? ¿por qué?
- Si pudiera inventar un nuevo sistema de tráfico, ¿cuál sería? ¿por qué?
- ¿Cree que el tráfico en Guatemala tiene problemas únicos? ¿Por qué?
- ¿Qué equipo le da a los agentes? ¿Cree que es efectivo?

c) Entrevista prototipo para agentes

- ¿Qué es lo más difícil de su trabajo?
- ¿Se comunica con sus compañeros para conocer el tráfico que está antes y después de su puesto de trabajo?
- ¿Cómo determina a quién le da vía?
- ¿Tiene herramientas que lo ayuden a dirigir el tráfico?
- ¿Cuál es la descripción oficial de su trabajo? ¿Siente que puede cumplirse todo?
- Llévame paso a paso que hiciste la última vez que estabas de turno como agente de PMT. ¿Qué sucedió primero? ¿Segundo?
- ¿Realizas otro trabajo adicional a dirigir el tráfico?
- ¿Puedes indicarme en qué consiste un buen día para un agente PMT? ¿Y un mal día?
- ¿Qué herramientas utilizas para tu trabajo?

d) Resumen de resultados. Pese a los intentos varios de entrevistar a ejecutivos y agentes, únicamente se logró entrevistar a dos agentes de tránsito. La mayoría argumentó que no podían brindar una entrevista sin autorización. Por otro lado, no se obtuvo respuesta positiva de ejecutivos pese a poseer su nombre, correo y teléfono. Se entrevistó a las siguientes personas, se adjunta un resumen del resultado de la entrevista:

- Persona 1 (Agente de policía de tránsito): es necesario conocer los diferentes puntos de vista de las personas involucradas en el tránsito en la ciudad. Está dispuesto a un cambio, por lo que utilizaría un nuevo sistema a pesar de desconocer lo que se necesita para mejorar.
- Persona 2 (Agente de policía de tránsito): su punto de vista es importante y a pesar del esfuerzo realizado no fue posible obtener una entrevista de alguien con cargo más alto. Tiene claras algunas de las deficiencias del sistema actual, pero no puede asegurar que un nuevo sistema sea de beneficio sin conocerlo.

e) Requerimientos claves. A continuación mostramos los requerimientos claves obtenidos de las entrevistas.

Cuadro 21. Requerimientos claves obtenidos de las entrevistas

Criterio	Persona 1	Persona 2
Obligatorio	Fácil aprendizaje	Período de prueba
Interesantes	Ayuda para toma de decisiones.	Modificación de tiempos de semáforos.
Extraordinario	Coordinación entre agentes.	

f) Diseño de oferta. A continuación se muestra cómo percibiría (según la entrevista realizada) las ofertas de productos que cumplen con los requerimientos las dos personas entrevistadas.

Cuadro 22. Ofertas de productos según requerimientos de personas entrevistadas.

Oferta	Persona 1	Persona 2
Ayuda para toma de decisiones	Con bastante información	No lo utiliza
Control de tiempos de semáforos	Indiferente	De forma fácil
Período de prueba	Indiferente	Verificar funcionamiento
Aprendizaje sencillo	Muy sencillo y en poco tiempo.	Conocer el producto
Coordinación entre agentes	Con bastante información	Indiferente
Satisfacción en general	Lo adquiere sino representa una carga tener que aprender a utilizarlo y si agiliza la coordinación y toma de decisiones.	Lo adquiere pero solamente después de verificar que le es útil.

g) Comparación con alternativas. No se encontró información de alternativas en el mercado.

Cuadro 23. Comparación del producto con alternativas en el mercado.

Atributos	Dispositivo desarrollado
Ayuda en la toma de decisiones	Aplica
Modificación de tiempos de semáforos	No aplica
Comunicación efectiva entre agentes	Aplica
Comparación en general	No existe comparación, la solución se trata de adaptar a las necesidades.

Debido a las complicaciones en la coordinación con los entes ejecutivos de la policía de tránsito y a la variedad desconocida entre semáforos utilizados en la ciudad, se decidió no implementar la opción de modificación de tiempos de semáforos en el dispositivo a desarrollar.

2) Análisis y obtención de requerimientos

a) Requerimientos funcionales

- ✓ Debe ser portátil.
- ✓ Debe poder ser utilizado como herramienta de trabajo.
- ✓ Debe poseer una interfaz gráfica fácil de entender.
- ✓ Debe comunicarse inalámbricamente.

- ✓ Debe permitir la interacción entre agentes.
- ✓ Debe ayudar en la toma de decisiones.

b) Requerimientos no funcionales

- ✓ La interacción entre el usuario y la máquina debe ser fluida.
- ✓ Debe asegurar la confiabilidad de los datos transferidos en la comunicación inalámbrica.
- ✓ La comunicación inalámbrica entre dispositivos debe ser lo más autónoma posible.
- ✓ El usuario debe poder notificar incidentes en cualquier momento.
- ✓ El usuario debe poder reproducir de nuevo la instrucción recibida.

c) Requerimientos de arquitectura

✓ Debe poseerse un dispositivo de salida tipo pantalla para poseer una interfaz fácil de entender y con la cual interactuar.

✓ No existen elementos mecánicos ni de electrónica de potencia.

✓ El sistema debe considerar la existencia de varios dispositivos del mismo tipo, operando simultáneamente.

d) Requerimientos físicos

✓ El dispositivo debe consumir una potencia aceptable para ser utilizado con una batería recargable.

✓ El dispositivo debe ser lo suficiente pequeño para ser considerado como portátil.

✓ El dispositivo debe ser lo suficientemente robusto para ser utilizado como herramienta de trabajo.

✓ El dispositivo debe ser lo suficientemente ligero para ser considerado como portátil.

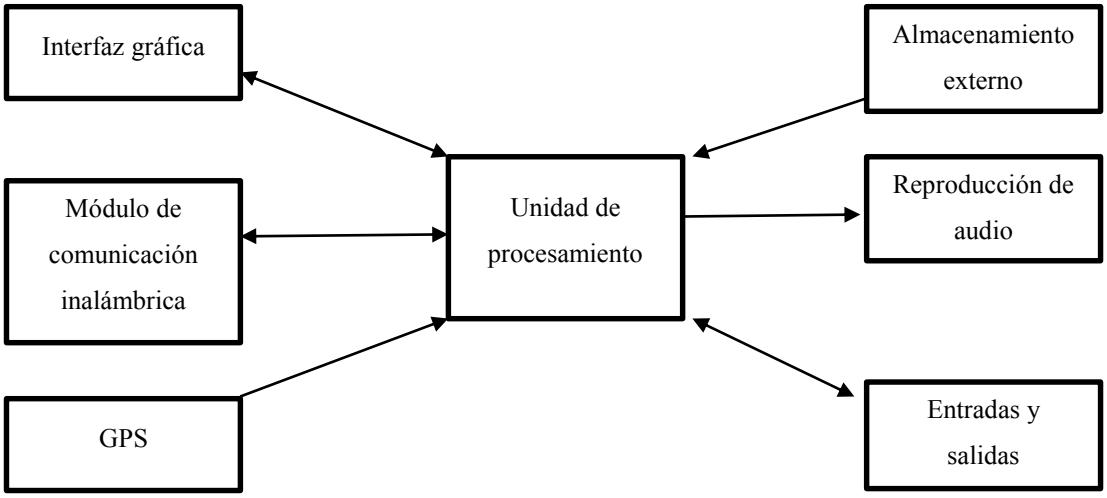
e) Requerimientos de ciclos de vida

✓ Como un primer desarrollo y prototipo, no se exigirán requerimientos de ciclo de vida, actualización de firmware, ni certificaciones.

b. Selección de arquitectura de hardware

1) Diagrama de bloques. Dados los requerimientos del sistema se propuso la siguiente solución.

Figura 134. Diagrama de bloques de sistema embebido



Cuadro 24. Validación del diagrama de bloques de sistema embebido respecto a los requerimientos del sistema.

Requerimiento	La propuesta tiene la capacidad de cumplir con el requerimiento	Notas
Debe poder ser utilizado como herramienta de trabajo	Sí	Con el controlador y definición de estados apropiados
Debe poseer una interfaz gráfica fácil de entender	Sí	Con el elemento de interfaz gráfica apropiado y una interfaz gráfica simplificada e intuitiva.
Debe comunicarse inalámbricamente	Sí	
Debe permitir la interacción entre agentes	No	Con la implementación completa del sistema la interacción no es necesaria.
Debe ayudar en la toma de decisiones	Sí	Con la implementación completa del sistema, esta es la función principal.
La interacción entre el usuario y la máquina debe ser fluida	Sí	Con el controlador e interfaz gráfica apropiada.
Debe asegurar la confiabilidad de los datos transferidos en la comunicación inalámbrica	Sí	Con un protocolo de comunicación adecuado.
La comunicación inalámbrica entre dispositivos debe ser lo más autónoma posible	Sí	Con un protocolo de comunicación adecuado.
El usuario debe poder notificar incidentes en cualquier momento	Sí	Con la implementación apropiada.
El usuario debe poder reproducir de nuevo la instrucción recibida	Sí	Con la implementación apropiada.
Debe poseerse un dispositivo de salida tipo pantalla para poseer una interfaz fácil de entender y con la cual interactuar	Sí	Con la selección del dispositivo correcto.

Continuación Cuadro 25

Requerimiento	La propuesta tiene la capacidad de cumplir con el requerimiento	Notas
El sistema debe considerar la existencia de varios dispositivos del mismo tipo, operando simultáneamente	Sí	Con la implementación apropiada.
El dispositivo debe consumir una potencia aceptable para ser utilizado con una batería recargable	Sí	Con los dispositivos adecuados.
El dispositivo debe ser lo suficiente pequeño para ser considerado como portátil	Sí	Con el diseño y selección de dispositivos correcto.
El dispositivo debe ser lo suficientemente robusto para ser utilizado como herramienta de trabajo.	Sí	Con el diseño y selección de dispositivos correcto.

2) Selección de unidad de procesamiento. Debido a la variedad de unidades de procesamiento existentes, se seleccionó la más apropiada según los criterios que se creyeron convenientes y sugeridos por la literatura. Se ponderó cada celda con un número donde 5 es positivo para el desarrollo del proyecto y 0 es negativo para el desarrollo del proyecto. Debe considerarse que esta ponderación se realiza conforme a los requerimientos del sistema a desarrollar.

Cuadro 26. Ponderación de unidades de procesamiento para el desarrollo del sistema embebido.

Unidad de procesamiento	Manufactura	Costo	Tiempo de desarrollo	Disponibilidad de herramientas	Complejidad del desarrollo	TOTAL
Microprocesador	5	4	5	4	3	21
Microcontrolador	5	5	5	5	3	23
Procesador de propósito único	5	3	5	3	2	18
VLSI	2	2	3	3	2	12

Existen muchas arquitecturas de microcontroladores, sin embargo, se seleccionaron las más reconocidas internacionalmente. Se compararon según sus características y luego se seleccionó la más apropiada. Se presenta el cuadro de comparación realizado y el listado de características de cada una de las arquitecturas.

Cuadro 27. Comparación entre arquitecturas de microcontroladores.

	ARM	8051	AVR	PIC	MPSP430
Protocolos de comunicación	UART, USART, I2C, SPI, CAN, USB, Ethernet, DSP	UART, USART, SPI, I2C	UART, USART, SPI, I2C	UART, USART, SPI I2C, CAN, Ethernet, USB	UART, USART, SPI, I2C
CPI	1	12	1	4	6
Memoria	Flash, SDRAM, EEPROM	ROM, SPRAM, FLASH	Flash, SRAM, EEPROM	SRAM, Flash	Flash, SRAM
ISA	RISC	CISC	RISC	RISC	RISC
Comunidad	Excelente	Excelente	Muy buena	Muy buena	Buena
Potencia	Baja	Regular	Baja	Baja	Muy baja
Costo	Bajo	Muy bajo	Regular	Regular	Regular
Tamaño de bus (bits)	32	8	8/32	8/16/32	16
Disponibilidad en el país	Bajo	Normal	Bajo	Alto	Bajo

Se seleccionó la arquitectura ARM ya que cumplía con las capacidades para el desarrollo del proyecto, existe una gran comunidad en internet para resolución de problemas, suficiente documentación y la gran variedad de protocolos de comunicación; pese a la no disponibilidad en el país y la complejidad que el desarrollo en una arquitectura desconocida representa.

c. Selección de herramientas de desarrollo. Existe una gran variedad de herramientas de desarrollo. Se seleccionaron cuatro de las más reconocidas, se nombraron sus características y se compararon para poder seleccionar la que mejor se adapta a los requerimientos del sistema embebido a desarrollar.

Cuadro 28. Comparación entre herramientas de desarrollo de arquitectura ARM.

	STM (Núcleo)	Mikroelectronika (EasyMx)	ARM (mbed)	ARM (Keil)
Programador	Sí	Sí	Sí	Sí
Debugger en circuito	Sí	Sí	Sí	Sí
Interfaces de comunicación	USART	USART, CAN, USB, USB host, Ethernet	Ethernet, USB, USB Host, CAN	Ethernet, USB, USB host, CAN, UART
Interfaces periféricas	Botones, LEDs, conexiones de extensión	Botones, LEDs, micro SD, Reproducción de audio, micrófono, conexiones de extensión, Buzzer, memoria externa, TFT, GLCD, LCD, joystick	PWM, analógica entrada y salida	Memoria externa, TFT, micro SD, Joystick, acelerómetro, reproducción de audio, micrófono
Documentación	Alta	Media	Media	Media
Ejemplos	Alta	Alta	Alta	Alta
Costo	Bajo	Medio	Bajo	Alto

Se seleccionó la herramienta de desarrollo de Mikroelectronika. Las demás herramientas a pesar de ser mejores en el costo que representa para el desarrollo, no poseen variedad de interfaces de comunicación y de periféricos necesarios para facilitar el desarrollo el proyecto. La herramienta que dispone Mikroelectronika ya posee todas las interfaces de comunicación necesarias y vende tarjetas con dispositivos de entrada, salida, expansión y otras interfaces que se adaptan a las necesidades.

Cuadro 28. Selección de dispositivo para interfaz gráfica

	Display alfanumérico	LCD	GLCD	TFT
Costo	Bajo	Medio	Alto	Alto
Dificultad en implementación	Bajo	Medio	Alto	Alto
Amigable con el usuario	Bajo	Bajo	Alto	Alto
Intuitiva	Bajo	Medio	Medio	Alto

Se seleccionó la pantalla TFT pese a su costo, debido a que los requerimientos exigen una interfaz amigable e intuitiva, esto se puede lograr ya que muestra datos a color y los gráficos con alta definición. Sin embargo, se tendrá que lidiar con las dificultades de su implementación en el sistema.

1) Hardware. Luego de seleccionadas las herramientas de desarrollo, se muestra el hardware a utilizar. Debido a que Mikroelectronika provee de distintas interfaces dentro de su inventario, solo seleccionó la que mejor se adecuan al dispositivo. Si fue necesario aplicar algún criterio para seleccionar un dispositivo, este es mencionado en la descripción el dispositivo.

a) EasyMx Pro STM32. Es un panel para desarrollo de microcontroladores STM32 ARM Cortex –M3 y Cortex –M4. Posee incorporado módulos variados para desarrollar aplicaciones multimedia, Ethernet, USB, CAN, etc. con debugger incorporado, programador incorporado, compatible con más de 180 microcontroladores ARM. Entre las interfaces incluidas también se encuentran, botones, socket Click, pull-up, pull-down, entrada analógica, conexión a pantallas TFT, tarjeta Micro SD, módulo para reproducción de MP3 y micrófono, USB host, convertidores UART a USB, buzzer. Memoria EEPRO, socket para sensores de temperatura.

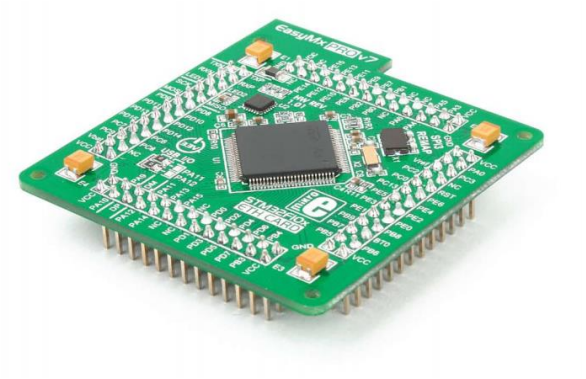
Figura 135. Fotografía del panel de desarrollo EasyMx Pro STM32 para ARM (Mikroelectronika)

El diagrama esquemático se encuentra en el siguiente enlace:
http://www.mikroe.com/downloads/get/1837/easymx_pro_v7_stm32_schematics_v102.pdf
La manual se encuentra en el siguiente enlace:
http://www.mikroe.com/downloads/get/1836/easymx_pro_v7_stm32_manual_v102.pdf

b) MCUcard STM32F107VGT6. Es una tarjeta que permite la interconexión rápida al panel EasyMx Pro STM32 para ARM. Contiene un microcontrolador (STM32F107VGT6) de 32 bits basado en la arquitectura ARM, Cortex –M3, con 64 kB de memoria flash, USB, Ethernet, 10 temporizadores, 2 puertos CAN, 2 ADC, 14 interfaces de comunicación. Opera a una frecuencia máxima de operación de 72 MHz. Una memoria de propósito general de 64 kB SRAM, DMA, modos de debugger, 80 pines de propósito general de entrada y salida, calcula CRC y posee un ID único de 96 bits, entre las características más importantes.

Además de este dispositivo el panel de desarrollo, EasyMX Pro STM32 para ARM, acepta más de 180 dispositivos. Se seleccionó este debido a que cumplía con el mínimo de requerimientos y que la mayoría de ejemplos se encuentran disponibles para este dispositivo.

Figura 136. Módulo MCUcard STM32F107VGT6 (Mikroelectronika)



El manual de la tarjeta se encuentra en el siguiente enlace:
http://www.mikroe.com/downloads/get/1839/stm32_mcu_stm32f107vgt6_manual.pdf
La hoja de datos del microcontrolador STM32F107VGT6 se encuentra en el siguiente enlace:
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00220364.pdf

c) Panel EasyTFT. Es un panel que contiene un controlador para una pantalla TFT compatible con el panel de desarrollo EasyMx Pro STM32 para ARM. Posee 320x240 pixeles (pantalla de 2.83'') y 262,000 colores y luz de fondo LED. Además, incluye un panel táctil.

Figura 137. Panel EasyTFT (Mikroelectronika).

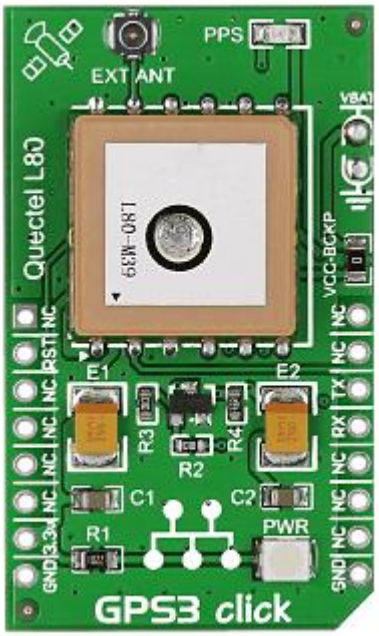
El manual puede encontrarse en el siguiente enlace:
http://www.mikroe.com/downloads/get/1928/easytft_manual_v101.pdf

El controlador de la pantalla es MI0283QT-9A y su manual puede encontrarse en el siguiente enlace:
http://www.mikroe.com/downloads/get/1975/tft_320_240_mi0283qt_9a_v1_3_spec.pdf

d) GPS3 Click. Esta tarjeta posee un chipset Quectel L80 con un módulo ultra delgado de GPS y una antena incrustada. Se comunica vía UART con 3.3 V y comandos AT. Posee tecnología Easy que permite encontrar y predecir la posición basándose en la memoria interna; AlwayLocate es una tecnología adaptativa que ajusta el tiempo de encendido y apagado para mejorar el consumo energético y una conmutación de antena automático que permite cambiar entre la antena incrustada y la externa manteniendo los datos de la posición durante el cambio.

Mikroelectronika provee 4 soluciones de GPS, todas con el mismo precio. La diferencia entre cada una de ellas es la tecnología utilizada, cada generación es mejor que la anterior. Se seleccionó la tercera y última generación disponible a la fecha.

Figura 138. Módulo GPS3 Click

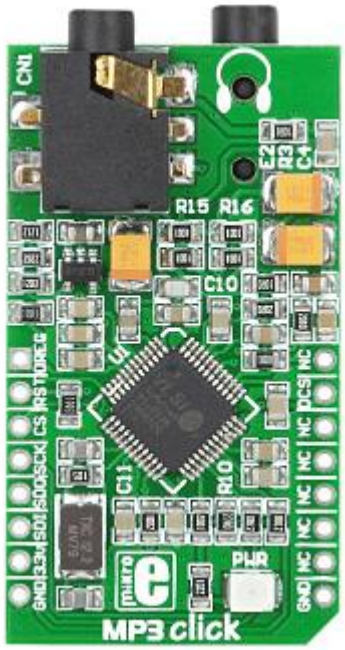


El manual de la tarjeta GPS3 Click puede encontrarse en el siguiente enlace:
http://www.mikroe.com/downloads/get/2168/gps3_click_manual_v100.pdf

La hoja de datos del chipset Quectel L80, dentro de la tarjeta GPS3 Click puede encontrarse en el siguiente enlace:
http://www.quectel.com/product/prodetail.aspx?id=62

e) MP3 Click. Es una tarjeta que contiene un decodificador MP3 estéreo con un circuito integrado VS1053 que puede además decodificar (MP1, MP2, MPEG4, WMA, FLAC, WAW, MIDI) y codificar a través de un micrófono en formatos IMA ADOCM, 16bits PCM en estéreo. Se comunica vía SPI y opera a 3.3 V. Este módulo se encuentra implementado dentro del panel de desarrollo EasyMx Pro STM32 para ARM.

Figura 139. Módulo MP3 Click



El manual del módulo puede encontrarse en el siguiente enlace:

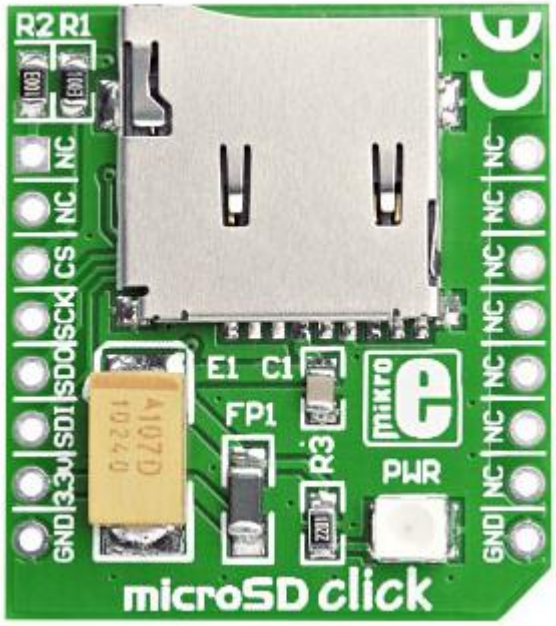
http://www.mikroe.com/downloads/get/1723/mp3_click_manual_v100.pdf

La hoja de datos del circuito decodificador y codificador de audio VS1053 puede encontrase en el siguiente enlace:

http://www.vlsi.fi/fileadmin/datasheets/vs1053.pdf

f) Micro SD Click. Es una tarjeta que incluye comunicación con una memoria micro SD para utilizarla como un dispositivo de almacenamiento masivo portátil. Se comunica por medio de SPI a alta velocidad y opera a 3.3V. Este módulo se encuentra implementado dentro del panel de desarrollo EasyMx Pro STM32 para ARM.

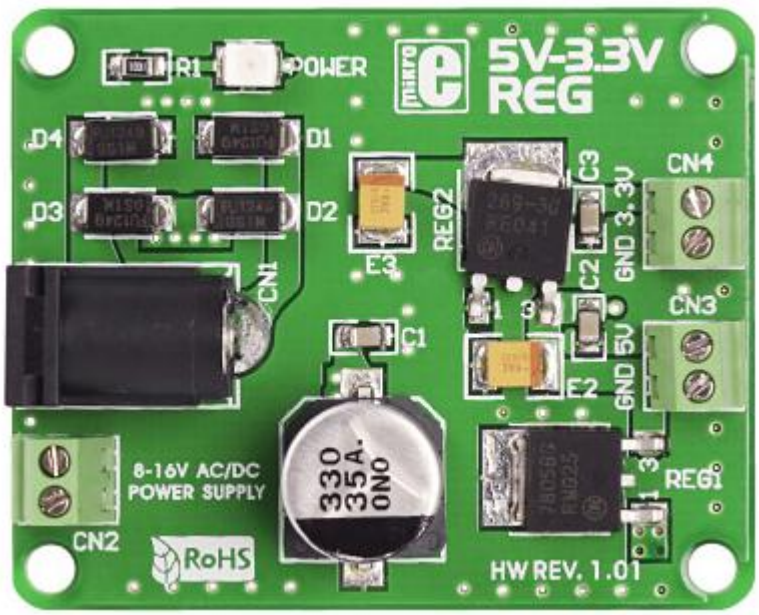
Figura 140. Módulo micro SD Click



El manual de la tarjeta micro SD Click puede encontrarse en el siguiente enlace: http://www.mikroe.com/downloads/get/1714/microsd_click_manual_v100b.pdf

g) Panel regulador 5V-3.3V. Es una módulo que se utiliza para poseer voltajes de operación estables de 5 V y 3.3 V con entradas de 8 – 16 V AC/DC. Para la salida de 5 V la corriente máxima es 1 A y para la salida de 3.3 V la corriente máxima es de 0.8 A. Este módulo se encuentra implementado dentro del panel de desarrollo EasyMx Pro STM32 para ARM.

Figura 141. Módulo panel regulador 5V – 3.3V (Mikroelectronika)



El manual del módulo se puede encontrar en el siguiente enlace: http://www.mikroe.com/downloads/get/1257/5v3_3v_reg_manual_v100.pdf

h) XBee. Los módulos XBee permiten la comunicación por radio frecuencias utilizando el estándar IEEE800.15.4, son de bajo consumo y aseguran la entrega correcta de la información entre los dispositivos. Cada módulo opera a 2.4 GHz y son compatibles entre modelos distintos.

Figura 142. Módulos XBee (Digi, 2015)

La hoja de datos del dispositivo se puede encontrar en el siguiente enlace: http://ftp1.digi.com/support/documentation/90000982_S.pdf

2) Herramientas de medición y construcción. Multímetro: dispositivo portátil utilizado para medir voltaje, corriente, resistencia y otras magnitudes eléctricas.

Osciloscopio: dispositivo utilizado para observar señales eléctricas en una pantalla. Generalmente se utiliza para mostrar señales que varían en el tiempo.

Fuente de poder, generador de señales, protoboard, componentes discretos, pinzas, cortaalambres, soldador, desolador, porta cautín, estaño y de más componentes básicos de electrónica.

Figura 143. Ejemplo de herramientas de construcción

(Catsoulis, 2005:204)

3) Software. Existen varios compiladores para microcontroladores ARM. A continuación se presentan sus características y se comparan.

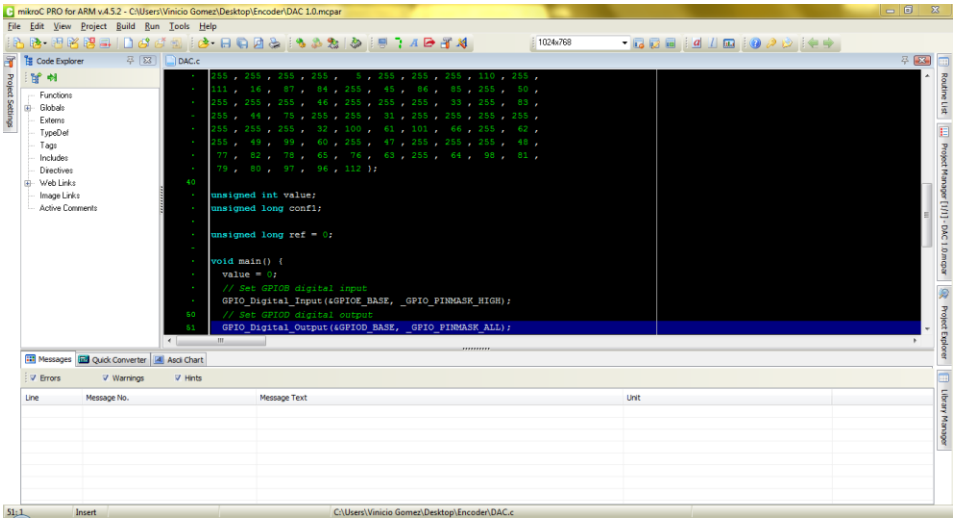
Cuadro 30. Comparación entre compiladores para microcontroladores ARM.

	MikroC	Keil	AIR
Optimización disponible	Sí	Sí	Sí
Librerías	Media	Alta	Media
Asistente de código	Sí	Sí	Sí
Asistente de interrupciones	Sí	No	No
Simulación	Sí	Sí	Sí
Comunidad	Media	Alta	Media
RTOS	No	Sí	Sí
Documentación	Alta	Alta	Media
Ejemplos	Media	Alta	Media
Otras herramientas	Alta	Alta	Media

Se seleccionó la herramienta MikroC como entorno de desarrollo. Sin embargo, puede observarse que la comparación entre los distintos software es muy pareja. Finalmente, la decisión está basada en que el hardware de desarrollo es completamente compatible por ser del mismo proveedor, la alta documentación y que existe la herramienta VisualTFT, posteriormente descrita, que facilita la creación de interfaces gráficas compatibles con el compilador MikroC y el hardware utilizado.

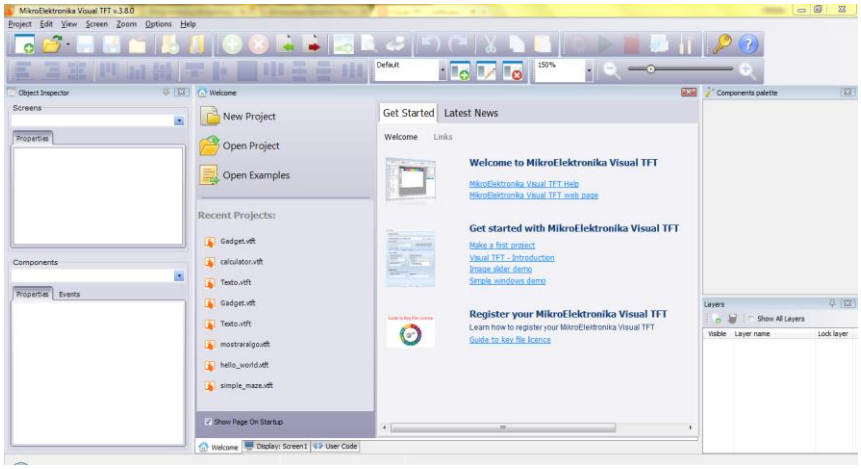
a) MikroC PRO for ARM. Es un compilador compatible con el estándar ANSI C para ARM Cortex-M0, Cortex-M3 y Cortex-M4. El compilador posee niveles configurables de optimización y librerías de hardware y software. Posee debuger, estadísticas del uso de memoria del proyecto, administrador de librerías, editor gráfico de las propiedades del proyecto, asistente de código, asistente de interrupciones, asistente de parámetros, explorador de proyecto, comentarios activos, convertidor rápido, herramientas varias integradas (terminales USART, USB HID, UDP, exportador a HTMLS, etc.), debuger en circuito, simulación por software, etc.

Figura 144. Interfaz gráfica de MikroC Pro for ARM



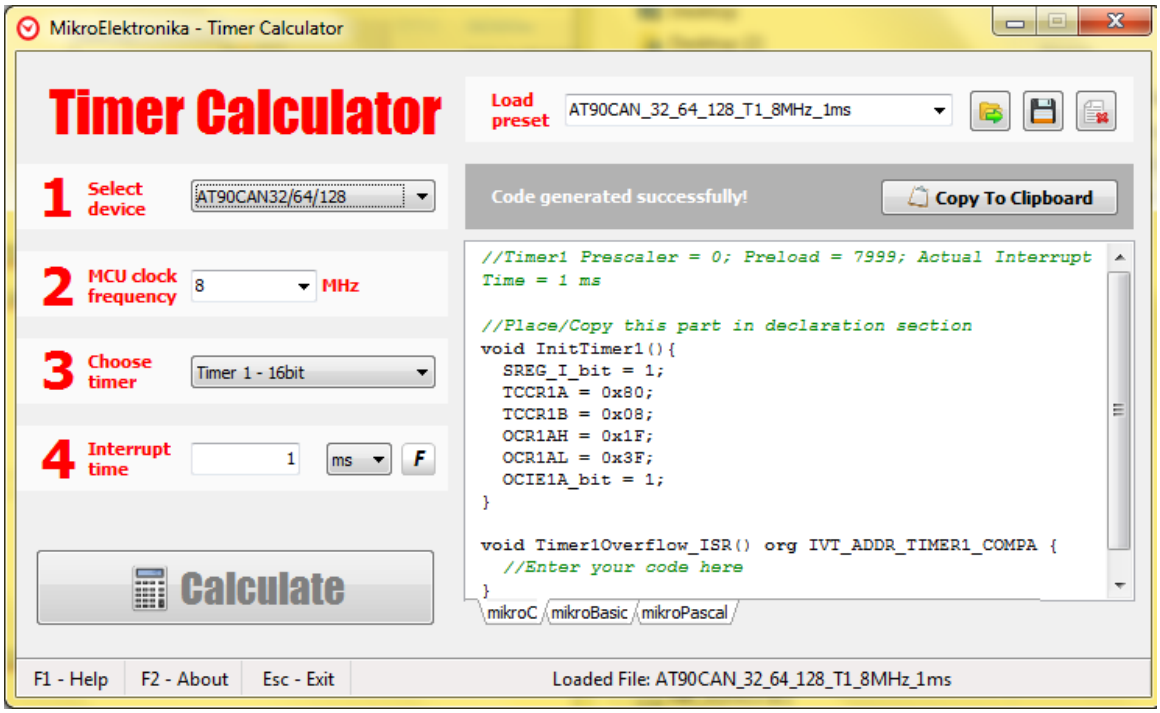
b) VisualTFT. Es una aplicación utilizada para el desarrollo rápido de interfaces gráficas para pantallas TFT. El software crea código compatible con el compilador MikroC, MikroBasic, MikroPascal para microcontroladores de arquitectura PIC, dsPIC, PIC24, PIC32, AVR, ARM, FT90x. Posee características como arrastra y suelta (drag-and-drop), con interfaz intuitiva, configuraciones por medio de interfaz gráfica, paleta de componentes, configuración para distintas pantallas en el mismo proyecto, agrupación de componentes, alineación y distribución, etc.

Figura 145. Interfaz gráfica de VisualTFT



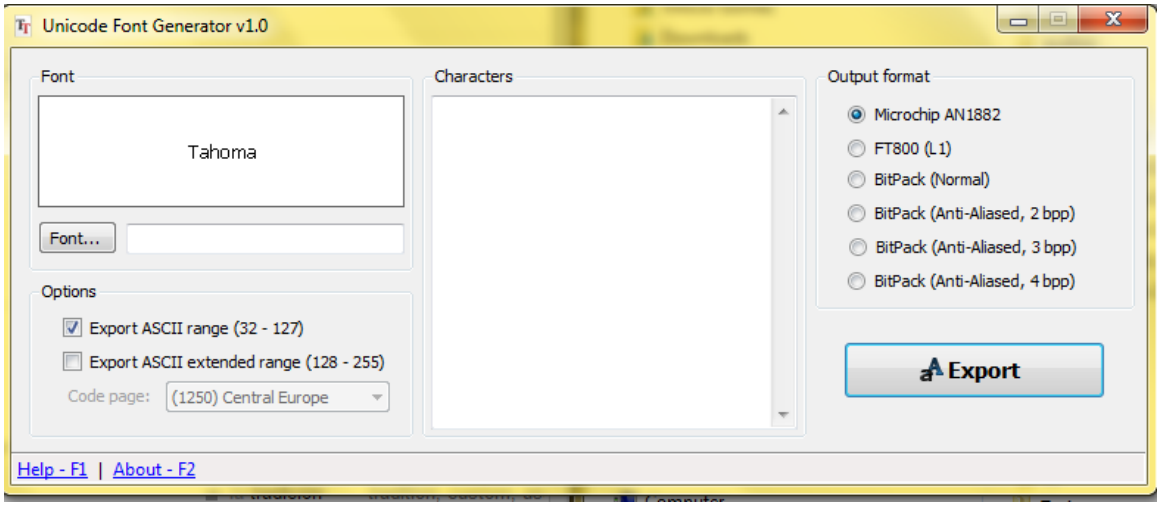
c) TimerCalculator. Es un programa que calcula tiempos de interrupción y genera el código de configuración e interrupción, para distintas arquitecturas de microcontrolador y distintos temporizadores dentro del mismo microcontrolador. Únicamente se debe introducirse la familia del microcontrolador, el temporizador a utilizar, la frecuencia de operación del microcontrolador y el tiempo de interrupción o frecuencia de interrupción.

Figura 146. Interfaz gráfica del programa TimerCalculator



d) Unicode Font Generator. Es una aplicación que permite generar y exportar caracteres personalizados en una tabla Unicode. Con esta aplicación se pueden crear los caracteres que se deseen. Luego, es posible mostrarlos en la pantalla TFT a través de VisualTFT.

Figura 147. Interfaz gráfica del programa Unicode Font Generator



d. Selección de arquitectura de software. La selección de arquitectura de software se basa en la arquitectura más sencilla que cumpla con los requerimientos. Conociendo las características de los módulos que serán utilizados y la forma de comunicación con ellos se establecieron cuales arquitecturas de software aplican

Cuadro 31. Evaluación de arquitecturas de software compatibles con la implementación de los módulos.

	Máquina de estados finitos	Round robin	Round robin con interrupciones	Cola de espera de funciones	Interrupciones	RTOS
GPS	No**	No	Sí	Sí	Sí	Sí
Pantalla TFT	No**	No	Sí *	Sí	Sí	Sí
Comunicación RF	Sí	Sí	Sí	Sí	Sí	Sí
Reproducción de audio	No **	No	No	No	Sí *	Sí
Entradas y salidas	Sí	Sí	Sí	Sí	Sí	Sí

* Restringiendo la capacidad de desempeño computacional de sistema. ** Una implementación eficiente podría implementar el módulo, pero aumentando la complejidad de la implementación. Las arquitecturas de software se ordenaron de menos compleja a más compleja de izquierda a derecha.

A pesar que la arquitectura de software puede ser la misma para todo el sistema no será así, debido a que la más simple que cumple con todos los módulos sería RTOS y que la complejidad de la implementación aumentaría mucho respecto a la que necesita la comunicación RF, por ejemplo. Se utilizará una arquitectura híbrida de la siguiente manera:

Cuadro 32. Selección de arquitecturas de software por módulo.

Módulo	Arquitectura de software
GPS	Round robin con interrupciones
Pantalla TFT	Round robin
Comunicación RF	Máquina de estados finitos
Reproducción de audio	Interrupciones
Entradas y salidas	Round robin

Se sacrificará el desempeño del sistema implementando la reproducción de audio con interrupciones, esto debido a que la reproducción de audio llevará al sistema a realizar demasiadas interrupciones, separadas muy poco tiempo unas de otras y con la posibilidad de corromper operaciones no atómicas. Pese a ello, se tendrá el cuidado de realizar pruebas suficientes para determinar si el sistema opera bien las reproducciones de audio y realizando las modificaciones necesarias en el desempeño del dispositivo para alcanzar los requerimientos del sistema.

e. Cronograma

Cuadro 33. Cronograma de desarrollo de Sistema embebido.

Descripción actividad/fecha	2014 dic.	2015 ene	01-feb	15-feb	01-mar	15-mar	01-abr	15-abr	01-may	15-may	01-jun	15-jun	01-jul	15-jul	01-ago	15-ago
Delimitación del funcionamiento del dispositivo																
Pruebas unitarias con módulo GPS																
Pruebas unitarias con pantalla táctil y a color																
Desarrollo de interfaz gráfica en pantalla táctil																
Integración de pantalla y GPS																
Pruebas unitarias con módulo de reproducción de audio																
Integración de módulo de reproducción de audio																
Implementación de mejoras al dispositivo de asistencia móvil																
Pruebas unitarias con módulo de comunicación RF																
Integración de módulo de comunicación RF																
Desarrollo de PCB																
Empaquetado de proyecto																

3. Diseño e implementación. El diseño se realizará modularmente, cada módulo se diseña por separado cumpliendo con la arquitectura de software y hardware seleccionada para cada uno. La implementación existirá en dos niveles: modular e integral. La implementación modular consiste en desarrollar el software y hardware que haga funcionar el dispositivo llenando los requerimientos del sistema correspondientes al módulo. La implementación integral consiste en que el módulo desarrollado en la implementación modular también llene los requerimientos pero trabajando en conjunto a los otros módulos ya desarrollados. El proceso de integración se repetirá por módulo hasta que todos los módulos sean integrados.

Respecto al diseño modular y la implementación modular, es importante que se realice considerando la implementación integral. Esto se logra con una buena modularización, jerarquía y estandarización de interfaces dentro del mismo programa que contendrá todas las arquitecturas de software trabajando en conjunto para cumplir con los requerimientos del sistema embebido.

a. Módulo GPS

1) Comunicación con el módulo. Según la documentación del módulo GPS3 Click, el módulo se comunica automáticamente (al alimentar el dispositivo) vía UART. Envía toda la información en formato NMEA (estándar para interfaces de dispositivos electrónicos marinos). Se presenta un resumen de las características del protocolo de información:

- Baudrate: 9600
- Bit de stop: 1
- Control de flujo: no
- Byte de inicio en cadena enviada: $
- Byte de fin en cadena enviada: <CR LF>
- Checksum error: si
- Tamaño de cadena: variable

2) Obtención de latitud y longitud. Debido a que el estándar NMEA comunica información sobre la cantidad de satélites, velocidad, altitud, latitud, longitud, orientación respecto al norte magnético, etc. y en distintas formas, dimensionales y estándares, debe obtenerse la información importante analizando y extrayendo la latitud y longitud de los textos recibidos del módulo.

Esto se realizará con los siguientes pasos:

- Recibiendo carácter por carácter en una interrupción.
- Determinando si se recibió el byte de inicio de cadena "$".
- Guardar byte tras byte recibidos en una cadena hasta encontrar el byte de fin de cadena "<CR LF>".
- Determinar si es un código "$GPGGA", estándar NMEA.
- Determinar si el string 43 posee un carácter "1", que significa que la información en válida y contiene la latitud y longitud.
- Extraer la latitud y longitud de la cadena recibida.

b. Pantalla TFT

1) Interfaz gráfica

a) Definir características. Dado los requerimientos del sistema, se determinó que la interfaz gráfica cumple con las siguientes características:

- Intuitiva: utilizará iconos, poco texto, pocos botones.
- Amigable con el usuario: el texto existente debe ser grande para ser legible con facilidad, interfaz con colores.

Las funciones que se realizarán en ella serán:

- Mostrar instrucción recibida
  - Calle, avenida, boulevard, arco, vía a la que se le dará la vía
  - Dirección: norte, sur, este, oeste, noreste, sureste, noroeste, suroeste a la que se le dará la vía.
  - Tiempo en segundos que se le dará la vía.
- Permitir notificación de incidente.
  - Bloqueo parcial o total debido a incidente de tránsito.
- Volver a reproducir instrucción recibida.
- Pedir nueva instrucción.

b) Desarrollo. Utilizando el software VisualTFT, se desarrollará la interfaz gráfica que cumplirá con las funciones descritas con anterioridad para cumplir con los requerimientos del sistema.

2) Apagado automático. La pantalla TFT debe apagarse para ahorrar energía y disminuir el consumo de corriente del dispositivo. Se implementará el uso de un temporizador que detectará la inactividad de la pantalla durante un tiempo prudente y se apagará la pantalla. La pantalla debe poder volverse a encender rápidamente.

3) Implementación de cambio de brillo. Se cambiará la intensidad de la luz de fondo de la pantalla TFT para que posea tres niveles de intensidad. El usuario configurará la intensidad del brillo de la pantalla a la que desee con el fin de que se sienta cómodo trabajando y observando la pantalla, sin que perjudique su comodidad. La intensidad de la luz de fondo de la pantalla TFT se controla mediante una señal PWM.

c. Reproducción de audio

1) Tarjeta SD. El almacenamiento de los archivos a reproducir no puede realizarse dentro del microcontrolador, debido al tamaño de los archivos y que la memoria del microcontrolador posee otros fines. Los archivos serán almacenados en una tarjeta micro SD.

Para la comunicación con la tarjeta se utilizará la librería integrada del software de desarrollo MikroC Pro para ARM. La tarjeta se comunica vía SPI, debe indicarse el nombre del archivo. La librería está limitada a formato FAT16 o FAT, esto significa que la tarjeta puede ser de hasta un máximo de 2G de capacidad de almacenamiento. Conociendo los nombres de los archivos dentro de la tarjeta, se pedirá a la tarjeta que envié la información de estos archivos (formato MP3).

Los archivos son fragmentos de instrucciones: calle, dirección, tiempo, otros. Según la instrucción recibida, se llamarán a los archivos correspondientes para que combinando los distintos archivos, se reproduzca por audio la instrucción completa.

Se presenta el conjunto de palabras necesarias para reproducir una instrucción

1)	Dar vía	7)	Avenida	13)	Este
2)	Detener	8)	Arco	14)	Oeste
3)	A	9)	Diagonal	15)	Noreste
4)	Vía	10)	Ruta	16)	Sureste
5)	Boulevard	11)	Norte	17)	Noroeste
6)	Calle	12)	Sur	18)	Suroeste

Se crearán archivos MP3 con los audios de cada palabra numerada.

2) Comunicación con módulo reproductor de MP3. El módulo reproductor MP3 se comunica por SPI, existen dos tipos de tramas que se le pueden enviar: comandos o datos. Cando se quiere enviar información a reproducir se envían tipo datos. Cuando se quiere configurar alguna función se envía como comandos. La configuración predeterminada del módulo es:

- Reproducción a 288 kHz
- Comunicación tipo esclavo SPI
- SPI 8 bits
- SPI bit más significativo primero

Al recibir los archivos de la memoria micro SD, serán enviados lo más pronto posible a reproductor de audio por medio de una interrupción.

3) Cambio de volumen. El dispositivo tendrá la capacidad de poder cambiar el volumen de la reproducción del audio para evitar incomodidad al usuario. Existen comandos, que al ser enviados al módulo reproductor de audio MP3, configuran el volumen de la reproducción.

4) Cambio de audio automático. Debido a que las instrucciones son conformadas por una combinación de audios. El dispositivo debe tener la capacidad de determinar cuándo ha sido finalizada la reproducción de un archivo y es necesaria la reproducción de otro para poder reproducir el audio de una instrucción completa. Esto se realizará conociendo el tamaño de los archivos y llevando un conteo de los bits enviados a reproducir. Además, se debe conocer cuántos archivos completan la reproducción de una instrucción completa y cuales son.

d. Comunicación inalámbrica

1) Comunicación con el módulo. El módulo de comunicación inalámbrica se comunica a través de UART, 9600 bits, 1 stop bit, sin control de flujo. Los datos serán recibidos a través de una interrupción, byte por byte. Cada byte representará un carácter ASCII. Existirá un byte de inicio y un byte de fin y estos determinarán el inicio y fin de una trama de bytes que representan algo dentro del protocolo de comunicación. La interpretación de esta información se hará en el ciclo principal, como se determinó en la selección de arquitectura de software (máquina de estados finitos).

2) Características. El protocolo de comunicación asegura que los datos han sido recibidos, sea cualesquiera y que esos datos no están corruptos mediante un código CRC. Los detalles del protocolo de comunicación, su funcionamiento y partes están definidos por otro módulo del megaproyecto y no serán tratados en estas secciones.

4. Pruebas por módulos

Las pruebas cualitativas se realizaron a 4 usuarios y se les requirió calificar el resultado según el siguiente cuadro:

Cuadro 34. Resultados posibles a pruebas cualitativas.

Resultado a prueba cualitativa
Excelente
Muy bueno
Bueno
Regular
Malo

a. GPS. Luego de la implementación del módulo GPS3 Click, se validará su funcionamiento con cuatro pruebas. La información recibida por el GPS será enviada a una terminal USART en una computadora y ahí será interpretada para determinar se la prueba fue aprobada o no.

1) Recepción de datos. Consiste en verificar que el módulo GPS3 Click está enviando información y que el programa implementado la recibe. En esta etapa se muestra al usuario la información tal y como el dispositivo GPS3 Click la transmite. La información recibida será retransmitida a otra interfaz UART conectada a una computadora, donde será interpretada. Se evaluará durante 1 minuto la recepción de datos. Se repetirá el experimento 5 veces, empezando por el encendido del módulo.

2) GPS conectado a sistema de posicionamiento global. Consiste en recibir información del módulo GPS3 Click y proveniente del sistema GPS, es decir información válida. En esta etapa no se interpreta la información. La información recibida será retransmitida a otra interfaz UART conectada a una computadora, donde será interpretada. Se evaluará durante 1 minuto la recepción de datos. Se repetirá el experimento 5 veces, empezando por el encendido del módulo.

3) Extracción de latitud y longitud. Consiste en comunicarse con el módulo GPS3 Click, conectarse satisfactoriamente al sistema GPS, analizar la información proveniente y extraer los datos de latitud y longitud. Estos serán enviados a la terminal de la computadora a través de UART. Se evaluará durante 1 minuto la recepción de datos. Se repetirá el experimento 5 veces, empezando por el encendido del módulo.

4) Validación de resultados de latitud y longitud. Consiste en determinar si los datos obtenidos de latitud y longitud son los correctos respecto a la ubicación actual del dispositivo, para ello se

validarán los datos con Google Earth. Se evaluará durante 1 minuto la recepción de datos. Se repetirá el experimento 5 veces, empezando por el encendido del módulo. Los datos de latitud y longitud no deben cambiar significativamente al encontrase el dispositivo en la misma ubicación.

b. Interfaz gráfica

1) Tiempo de respuesta. Consiste en evaluar si la velocidad de respuesta de la interfaz gráfica es aceptable según las expectativas. Cualitativamente se ponderará si responde lo suficientemente rápido o se esperaban mejores resultados. El experimento se repetirá 5 veces a distintos usuarios esperando tiempos de respuesta constantes bajo distintas condiciones.

2) Apagado de pantalla tras inactividad. Consiste en evaluar la funcionalidad de la pantalla para apagarse después de cierto tiempo de inactividad y el regreso a operación normal al ser reactivada. El tiempo de regreso a operación será evaluado cualitativamente respecto a las expectativas. La funcionalidad del apagado tras inactividad será realizado con evaluación binara: si cumple, no cumple. Se repetirá el experimento 10 veces esperando tiempos de encendidos tras inactividad constantes bajo distintas condiciones, este tiempo se evaluará cuantitativamente.

3) Cambio de brillo configurable de pantalla. Consiste en evaluar la funcionalidad del cambio de brillo de pantalla. Se realizará una evaluación binara de funcionalidad: si cumple, no cumple. Se realizará una evaluación cualitativa respecto a cómo responde el sistema según las expectativas. El experimento se realizará 10 veces esperando tiempos de respuesta constantes, esto se realizará cuantitativamente.

c. Reproducción de audio

1) Calidad de reproducción de audio. Consiste en evaluar si el audio cumple con las expectativas. Cualitativamente se ponderará la calidad del audio. Se espera un audio que se reproduce de forma fluida, con un volumen máximo aceptable y sin saturación excesiva y si el audio sigue reproduciéndose con la misma calidad y si el cambio entre archivos es lo suficientemente rápido para considerar realizar una reproducción de instrucción completa de calidad. Se considerará la evaluación de múltiples usuarios. Se realizará el experimento 3 veces por usuario.

d. Comunicación inalámbrica

1) Confiabilidad en la entrega de datos. Consiste en asegurar que el protocolo de comunicación asegura la entrega de datos entre dispositivos. Se realizará una evaluación binaria: si cumple, no cumple. El experimento se repetirá 10 veces.

2) Integridad de datos. Consisten en verificar que el código CRC utilizado para detectar errores, es generado correctamente en el dispositivo y corresponde al obtenido por la información recibida. Se realizará una evaluación binaria: si cumple, no cumple. El experimento se repetirá 10 veces.

3) Recepción de una instrucción. Consiste en verificar que se implementaron correctamente todos los pasos que el protocolo de comunicación requiere para que la recepción de una instrucción se realice. Además, se evaluará la interpretación correcta de la instrucción recibida: calle, dirección, tiempo e indicador de congestionamiento. Se realizará una evaluación binaria: si cumple, no cumple. El experimento se repetirá 10 veces.

4) Envío de una notificación. Consiste en verificar que se implementaron correctamente todos los pasos que el protocolo de comunicación requiere para que la transmisión de una notificación se realice. Además, se verificará que una notificación puede ser generada a través del dispositivo con la información requerida: latitud, longitud y tipo de notificación (bloqueo total o parcial). Se realizará una evaluación binaria: si cumple, no cumple. El experimento se repetirá 10 veces.

5. Pruebas integradas. Luego de la etapa de implementación modular, se realizarán pruebas en la implementación integrada del sistema. Estas pruebas tienen como objetivo evaluar el desempeño de los módulos luego de la integración respecto a los requerimientos del sistema. Idealmente, el desempeño debe ser el mismo.

a. Tiempo de encendido. Consiste en la evaluación del tiempo en que tarda en inicializar todos los módulos e interfaz gráfica para pasar a un estado de operación y estar listo para interactuar con el usuario. La ponderación será cualitativa y cuantitativa y será realizada respecto el tiempo de encendido en la implementación modular e individualmente.

b. Recepción de instrucción

1) Reproducción de audio. Consiste en evaluar la calidad del sonido de las instrucciones como un todo y como un conjunto de distintos archivos siendo reproducidos consecutivamente. Se realizará la prueba a tres usuarios y se repetirá la prueba cuatro veces por usuario. La ponderación será cualitativa.

2) Reacción de interfaz. Consiste en evaluar como la interfaz gráfica reacciona a la interpretación de una instrucción recibida. Se realizará la prueba a tres usuario y se repetirá la prueba cuatro veces por usuario. La ponderación será cualitativa.

c. Envió de notificación

1) Interfaz gráfica. Consiste en evaluar como la interfaz gráfica reacciona al envío de una notificación, desde el menú principal. Se realizará la prueba a tres usuario y se repetirá la prueba cuatro veces por usuario. La ponderación será cualitativa.

d. General

1) Cumplimiento de los requerimientos. Se realizará una autoevaluación del cumplimiento de los requerimientos de sistema embebido obtenido de las entrevistas y el análisis y determinación de los mismos.

F. RESULTADOS

1. Pruebas por módulos

a. GPS

1) Recepción de datos

Ejemplo de los datos recibidos:

EV*G097T,TO$GPR0,.,,$.M,NP1,,*G,,1S0P,87X,TO$GPR0,.,,$.M,NP2,,*G,,1S0P,87X,TO$GPR0,.,,.M,NP2,,*G,,1S0P,87X,TO$GM,NP2,,*G,GPR0,.0,,,1S0P,87X,THGPR00.0,.M,NP1,,*G,,EV*G,87X,TO$GPR0,.,,$.M,NP2,,*G,,1S0P,87X,TO$GPR0,.,,$.,0NP2,,*G,,EV*G,07X,TO$GPR0,.0,.,0*G1,,*G,,EV*G007T,TO$GPR0,.,,$.M,NP2,,*G,,1S0P,87X,TO$GPR0,.,,$.M,NP2,,*G,,1S0P,87X,TO$GPR0,.0,.M,NP2,,*G,,1S0P,87X,TO$GM,NP2,,*G,GPR0,.,,,,1S0P,87X,THGPR00.0,.,0NP2,,*G,,EV*G007X,TO$GPR0,.,,$.M,NP2,,*G,,EV0P,87X,TO$GPR0,.,,$.M,NP2,,*G,,1S0P,87X,TO$GPR0,.0,.M,NP2,,*G,,1S0P,87X,TO$GM,NP2,,*G,,1S0P,87X,THGPR0000,.,0NP2,,*G,,EV*G007X,TO$GPR0,.0,.,0*G2,,*G,,EV*G007X,TO$GPR0,.,,.,0NP1,,*G,,EV*G,87X,TO$GPR0,.,,$.M,NP2,,*G,,EV0P,87X,TO$GPR0,.0,.M,NP1,,*G,,EV0P,87X,TO$GPR0,.,,$.,0NP1,,*G,,EV*G007X,TO$GPR0,.0,

El experimento no se realizó durante 1 minuto siempre, debido a que el módulo GPS3 Click pasaba a estado de conexión con el sistema GPS antes de transcurrir 1 minuto.

2) GPS conectado a sistema de posicionamiento global

Ejemplo de los datos recibidos:

GPGGA,1GPTXT,01,01,02,ANTSTATUS=SHORT*6D13.000,A,A*47,48,38,254,27*7FFGPRMC,202114.000,A,1435.1496,N,09034.GPVTG,0.00,T,,M,0.32,N,0.60,K,A*3A.7432,W,0.32,0.00,270915,,,A*74FGPGGA,202114.000,1435.1496,N,09034.7432,W,1,5,1.23,1GPGSA,A,3,05,29,12,25,131GPTXT,01,01,02,ANTSTATUS=SHORT*6D18.000,A,A*4D02,,,30*4523*7F5GPRMC,202119.000,A,1435.1497,N,09034.GPVTG,0.00,T,,M,0.38,N,0.71,K,A*30.7428,W,0.38,0.00,270915,,,A*795GPGGA,202119.000,1435.1497,N,09034.7428,W,1,5,1.23,1GPGSA,A,3,05,29,12,25,132GPTXT,01,01,02,ANTSTATUS=SHORT*6D23.000,A,A*4002,,,30*4422*7D8GPRMC,202124.000,A,1435.1502,N,09034.GPVTG,0.00,T,,M,0

Se obtuvieron los mismos resultados en las distintas pruebas.

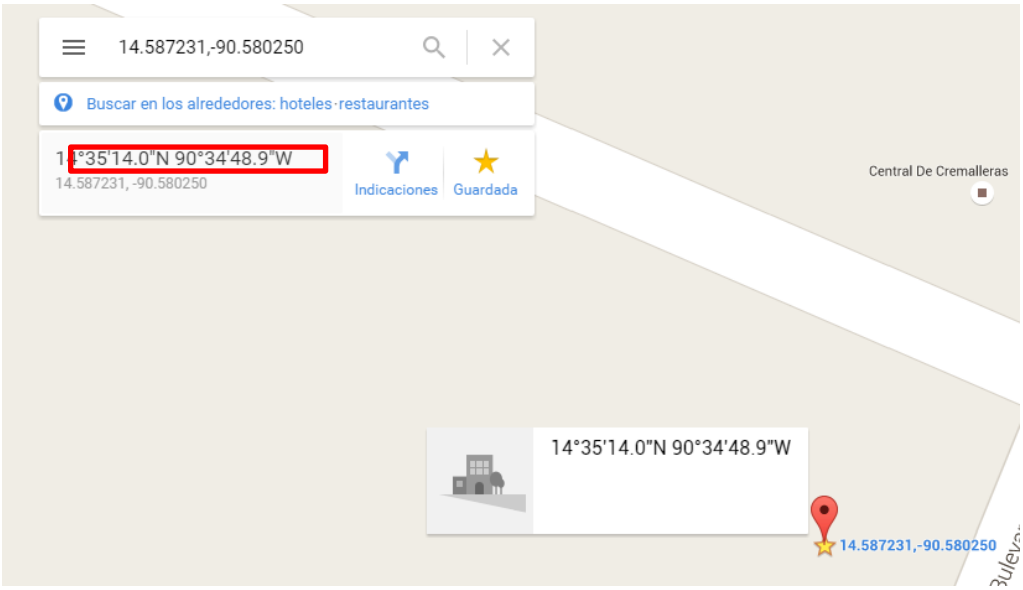
3) Extracción de latitud y longitud. Ejemplo de los datos recibidos en la interfaz USART de la computadora:

Inválido inválido inválido inválido inválido inválido inválido
Coordenadas: 1435.236 -9034.816
Coordenadas: 1435.236 -9034.816
Coordenadas: 1435.236 -9034.816
Coordenadas: 1435.236 -9034.816
Coordenadas: 1435.236 -9034.816
Coordenadas: 1435.236 -9034.816
Coordenadas: 1435.238 -9034.817
Coordenadas: 1435.238 -9034.817

Se obtuvieron los mismos resultados durante el tiempo del experimento y las distintas veces que se repitió el experimento. Los datos variaban únicamente en la última cifra significativa.

4) Validación de resultados de latitud y longitud. Los datos se validaron respecto a la ubicación según Google Maps y no Google Earth, debido a que permite una mejor aproximación hasta la ubicación deseada. Se presenta la ubicación obtenida durante la realización del experimento. La representación de los datos de interés se encuentra marcada en rojo. Existe cierto error debido a que la aplicación no permite el acercamiento suficiente para seleccionar la ubicación con mayor precisión.

Figura 148. Ubicación de la realización del experimento.



Se compararon con los datos obtenidos por el módulo GPS3 Click. Se obtuvo un promedio según una muestra de cincuentra datos obtenidos. Se convirtieron los datos al mismo formato (grados, minutos, segundos).

Cuadro 35. Comparación de resultados teóricos y experimentales del módulo GPS.

Valor GPS Google Maps	Valor GPS módulo (promedio)	Error (m)
14°35’14’’ 90°34’48’’	14°35'14.2" 90°34'49.0"	4.1

Figura 149. Fotografía de módulo GPS3 Click en funcionamiento (led rojo).

b. Interfaz gráfica

1) Tiempo de respuesta

Cuadro 36. Resultado a pruebas de velocidad en tiempo de respuesta.

Prueba	Resultado
Promedio de prueba cualitativa	Aceptable

2) Apagado de pantalla tras inactividad

Cuadro 37. Resultados a pruebas de apagado de pantalla tras inactividad.

Prueba	Resultado
Funcionalidad del apagado automático tras inactividad	Si cumple
Tiempo de apagado tras inactividad	20 s

Continuación Cuadro 36

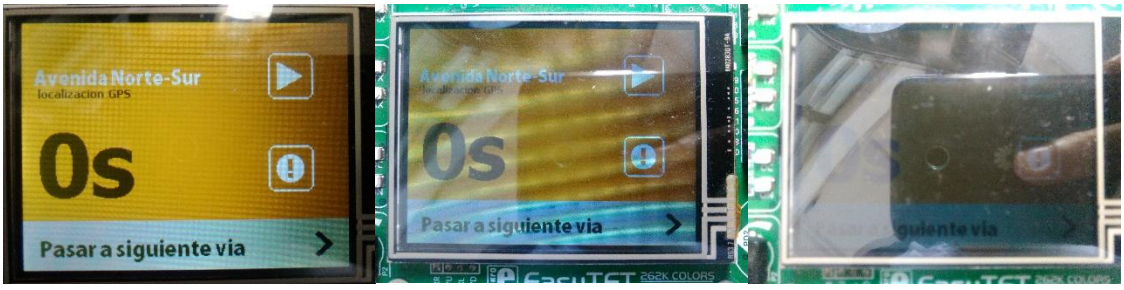
Prueba	Resultado
Tiempo de encendido tras inactividad	<500 ms
Consumo de corriente previo a apagado automático	130 mA
Consumo de corriente posterior a apagado automático	110 mA

3) Cambio de brillo configurable de pantalla

Cuadro 38. Resultado de pruebas a cambio de brillo configurable de pantalla.

Prueba	Resultado
Funcionalidad del brillo configurable de pantalla	Si cumple
Promedio de evaluación cualitativa	Excelente
Tiempo de cambio de brillo en pantalla	<100 ms
Consumo de corriente con brillo máximo	130 mA
Consumo de corriente con brillo medio	120 mA
Consumo de correinte con brillo minimo	110 mA

Figura 150. Fotografías de distintos brillos de pantalla. De izquierda a derecha, mayor brillo, menor brillo.



c. Reproducción de audio

1) Calidad de reproducción de audio

Cuadro 39. Resultado de pruebas de calidad de audio.

Prueba	Resultado
Promedio de la evaluación cualitativa de la calidad del audio	Buena
Promedio de la evaluación cualitativa del rango de volumen del audio	Excelente
Promedio de la evaluación cualitativa del cambio de archivos para reproducir una instrucción	Buena

d. Comunicación inalámbrica

1) Confiabilidad en la entrega de datos. Las pruebas de recepción de instrucciones y envío de notificaciones se realizaron 10 veces cada una.

Cuadro 40. Resultado de pruebas de confiabilidad en la entrega de datos.

Prueba	Resultado
Confiabilidad en entrega de datos	Sí cumple
Recepción de instrucciones	100%
Envío de notificaciones	100%

2) Integridad de datos. Las pruebas de recepción de instrucciones y envío de notificaciones se realizaron 10 veces cada una.

Cuadro 41. Resultado de pruebas de integridad de datos.

Prueba	Resultado
Integridad de datos	Sí cumple
Recepción de instrucciones	100%
Envío de notificaciones	100%

3) Recepción de una instrucción. Las pruebas de recepción de instrucciones se realizaron 10 veces.

Cuadro 42. Resultado de pruebas de recepción de instrucciones.

Prueba	Resultado
Recepción de instrucciones	Sí cumple
Recepción de instrucciones	100%
Interpretación de instrucciones	100%

4) Envío de una notificación. Las pruebas de envío de motivación se realizaron 10 veces.

Cuadro 43. Resultado de pruebas de envío de notificación.

Prueba	Resultado
Envío de notificación	Sí cumple
Envío de notificación	100%
Generación de notificación	100%

2. Pruebas integradas

a. Tiempo de encendido

Cuadro 44. Resultado de pruebas integradas de tiempo de encendido.

Prueba	Resultado
Promedio de la evaluación cualitativa del tiempo de encendido	Excelente
Tiempo de encendido	< 700 ms
Tiempo de encendido tras inactividad	< 500 ms

b. Recepción de instrucción

1) Reproducción de audio

Cuadro 45. Resultado de pruebas integradas de reproducción de audio.

Prueba	Resultado	Comentarios
Promedio de la evaluación cualitativa de la calidad del audio	Buena	La instrucción no fluye con naturalidad entre archivos de audio.
Promedio de la evaluación cualitativa del rango de volumen del audio	Excelente	
Promedio de la evaluación cualitativa del cambio de archivos para reproducir una instrucción	Buena	La instrucción no fluye con naturalidad entre archivos de audio.

2) Reacción de interfaz

Cuadro 46. Resultados de pruebas integradas de reacción de interfaz.

Prueba	Resultado	Comentarios
Promedio de la evaluación cualitativa de la reacción de la interfaz gráfica a la interpretación de una instrucción recibida.	Mala	La interfaz gráfica debería poderse seguir utilizando mientras se reproduce el audio.

c. Envío de notificación

Cuadro 47. Resultado de pruebas integradas de envío de notificación.

Prueba	Resultado
Promedio de la evaluación cualitativa a la interacción del usuario para enviar una notificación	Muy buena

3. Cumplimiento de los requerimientos. Se presenta la autoevaluación del cumplimiento de los requerimientos del sistema con un porcentaje de 0 a 100%.

Cuadro 48. Autoevaluación del cumplimiento de los requerimientos del sistema embebido.

Requerimiento	Se cumple con el requerimiento	Notas
Debe poder ser utilizado como herramienta de trabajo	80%	La pantalla TFT puede dañarse con relativa facilidad.
Debe poseer una interfaz gráfica fácil de entender	100%	
Debe comunicarse inalámbricamente	100%	
Debe permitir la interacción entre agentes	100%	
Debe ayudar en la toma de decisiones	100%	
La interacción entre el usuario y la máquina debe ser fluida	100%	
Debe asegurar la confiabilidad de los datos transferidos en la comunicación inalámbrica	100%	
La comunicación inalámbrica entre dispositivos debe ser lo más autónoma posible	100%	
El usuario debe poder notificar incidentes en cualquier momento	100%	
El usuario debe poder reproducir de nuevo la instrucción recibida	100%	
Debe poseerse un dispositivo de salida tipo pantalla para poseer una interfaz fácil de entender y con la cual interactuar	100%	
El sistema debe considerar la existencia de varios dispositivos del mismo tipo, operando simultáneamente	100%	
El dispositivo debe consumir una potencia aceptable para ser utilizado con una batería recargable	80%	Podrían implementarse otras técnicas para reducir el consumo.
El dispositivo debe ser lo suficiente pequeño para ser considerado como portátil	70%	Se deberá realizar una prueba de campo.
El dispositivo debe ser lo suficientemente robusto para ser utilizado como herramienta de trabajo.	70%	La pantalla TFT podría dañarse con relativa facilidad.
Total	93.33%	

4. Características del sistema desarrollado

Cuadro 49. Características generales del Sistema desarrollado.

Voltaje de alimentación	9 V
Consumo de corriente	110 mA – 150 mA
Memoria de programa	151 kB
Memoria de datos	3.7 kB
Protocolos de comunicación utilizados	UART, SPI
Tiempo de encendido	< 700 ms
Velocidad de operación	72 MHz
Almacenamiento externo	Micro SD (máx. 2 GB)
Limitaciones GPS	Operación en campo abierto
Limitaciones de reproducción de audio	Detiene la utilización de la interfaz gráfica
Interfaz gráfica	Pantalla TFT 230x240 pixeles (pantalla de 2.83'') y 262,000 colores
Configuración de brillo	Tres distintos niveles de brillo para la interfaz gráfica
Ahorro de energía	Apaga interfaz gráfica tras inactividad
Protocolo de comunicación	Asegura la confiabilidad y detección de errores
Reproducción de audio	Capacidad para subir y bajar volumen

Figura 151. Fotografía de panel de desarrollo con todo el sistema implementado.

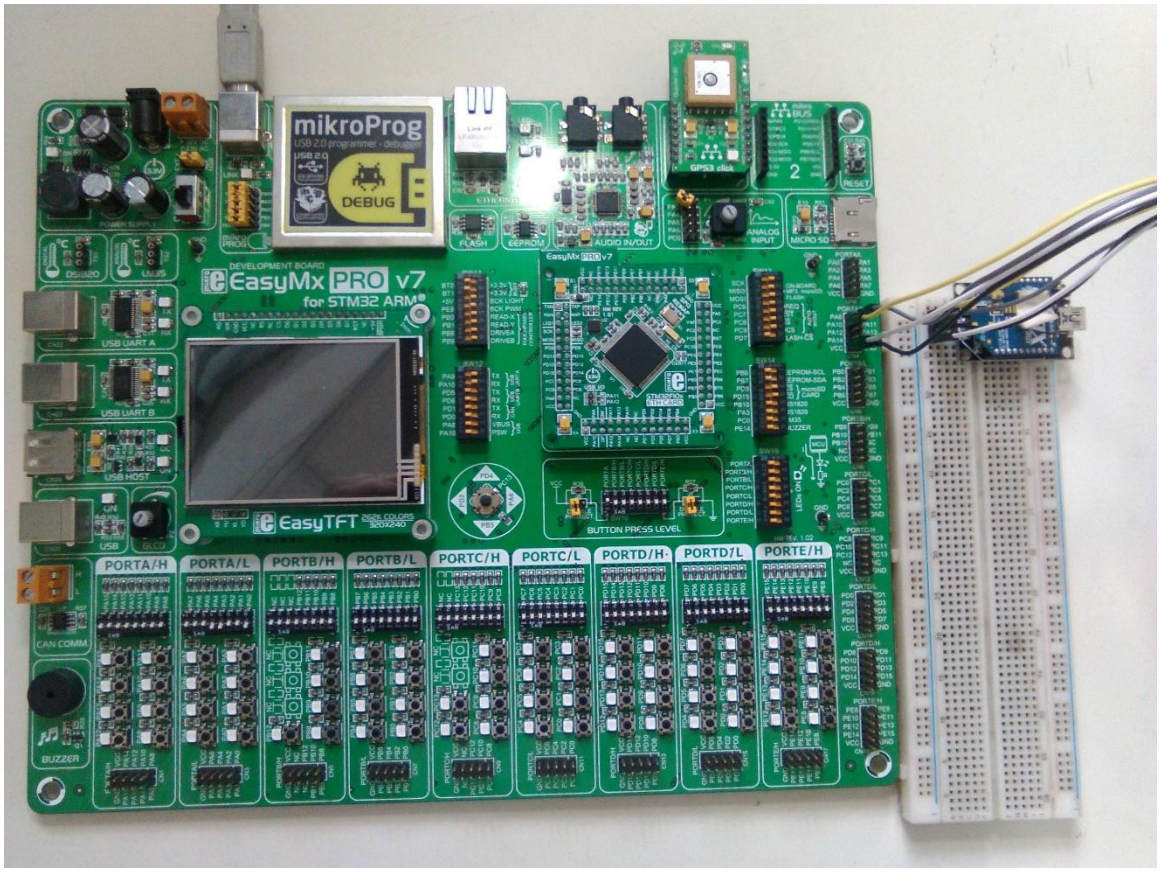
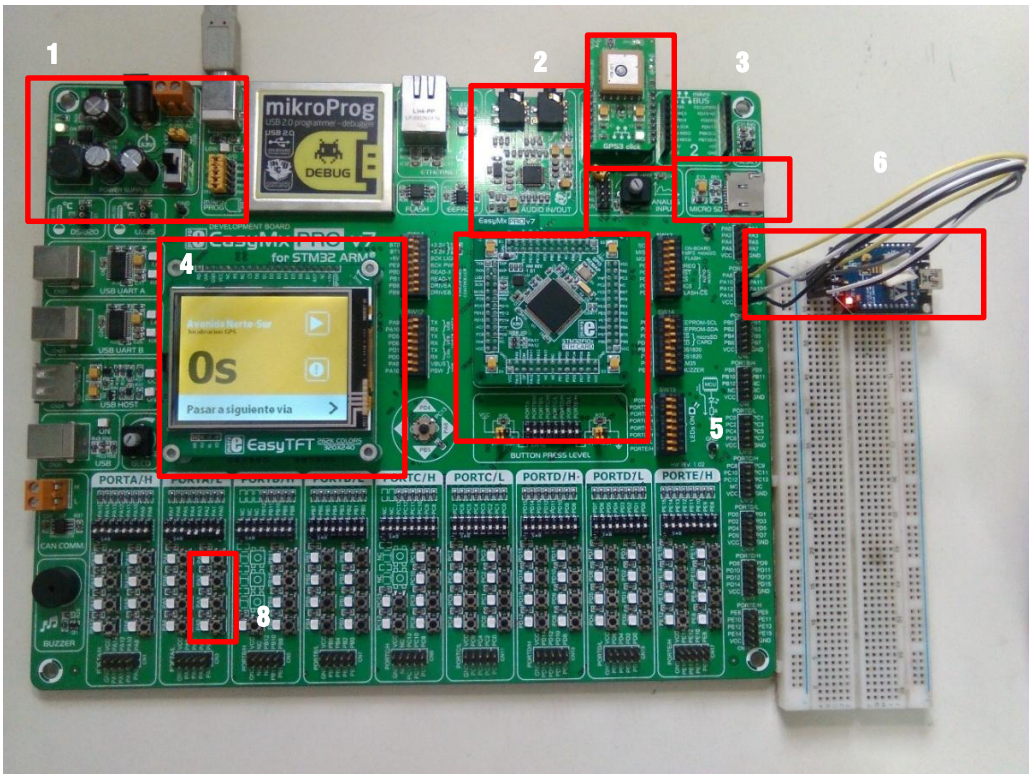


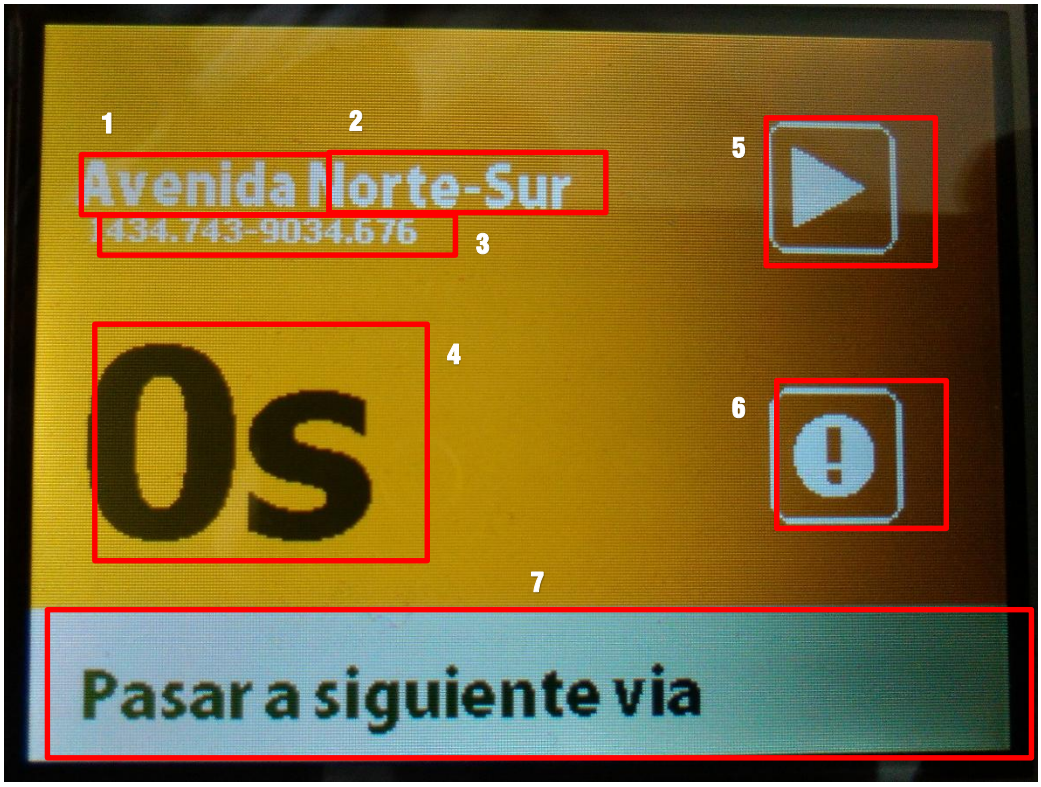
Figura 152. Fotografía del panel de desarrollo encendido y todo el sistema implementado.



Los números de la figura anterior corresponden a:

1. Fuente de alimentación
2. Reproductor de MP3
3. GPS
4. Pantalla TFT
5. Microcontrolador
6. Micro SD
7. XBee
8. Botones (subir volumen, bajar volumen, cambiar brillo).

Figura 153. Fotografía de interfaz gráfica principal.



Los números de la figura anterior corresponden a:

1. Calle a la cual debe darse vía.
2. Dirección a la cual debe darse vía de la calle del punto 1.
3. Coordenadas de la ubicación.
4. Tiempo restante para dar vía.
5. Volver a reproducir instrucción.
6. Notificar incidente.
7. Pedir nueva instrucción.

Figura 154. Fotografía de interfaz gráfica de menú de notificaciones.

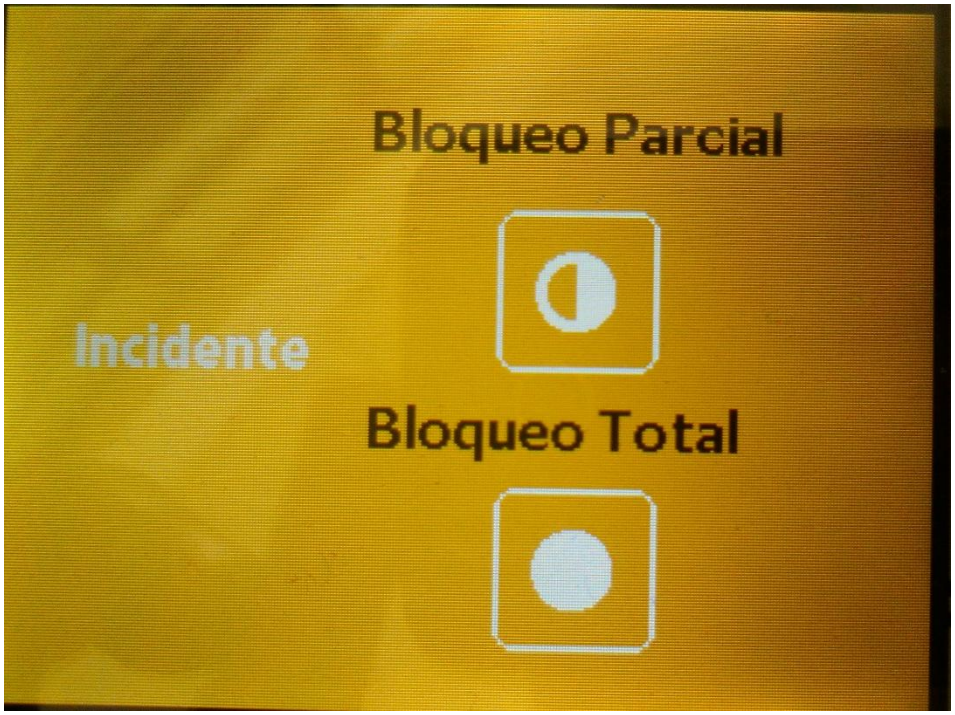


Figura 155. Memoria de datos utilizada en microcontrolador, considerar únicamente la memoria estática.

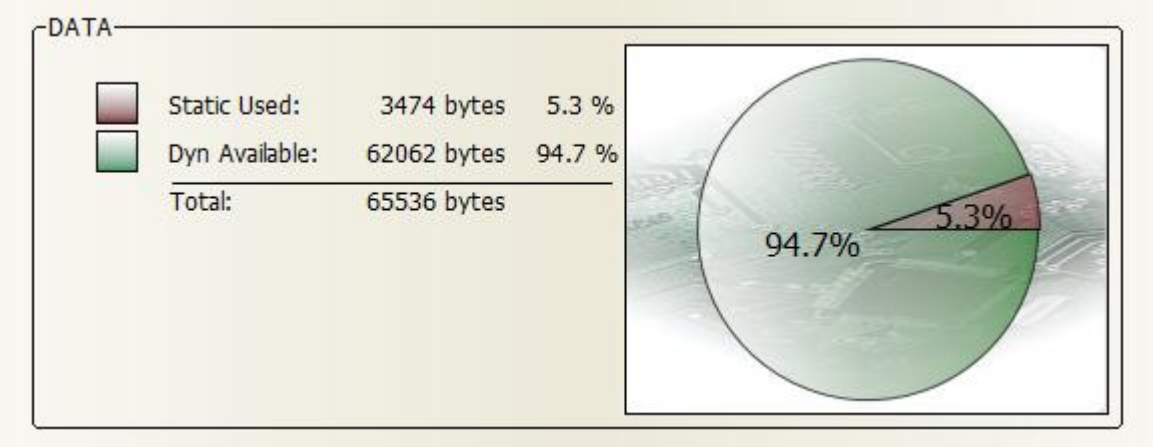


Figura 156. Memoria de datos utilizada en microcontrolador.

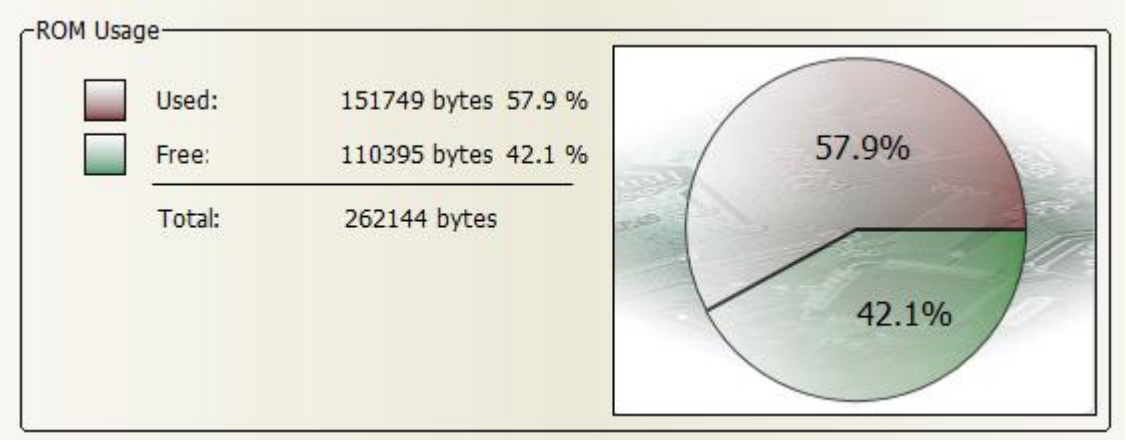


Figura 157. Tamaño (en bytes) de funciones 1/2, únicamente se muestran las funciones con mayor tamaño.

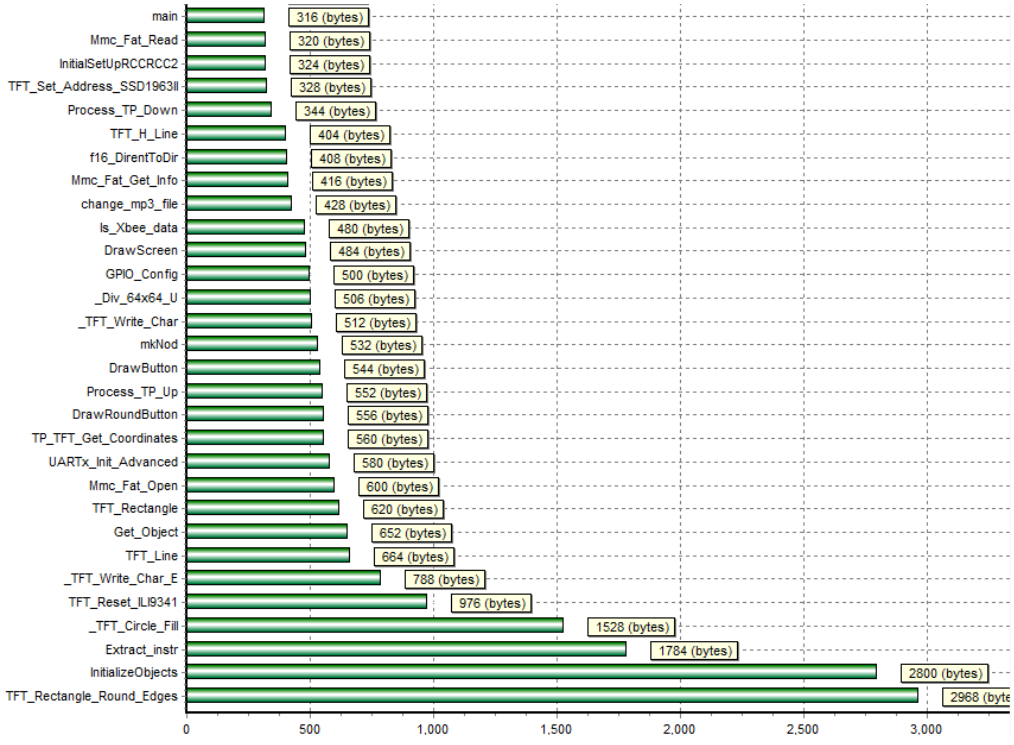


Figura 158. Tamaño (en bytes) de funciones parte 2/2, únicamente se muestran las funciones con mayor tamaño.

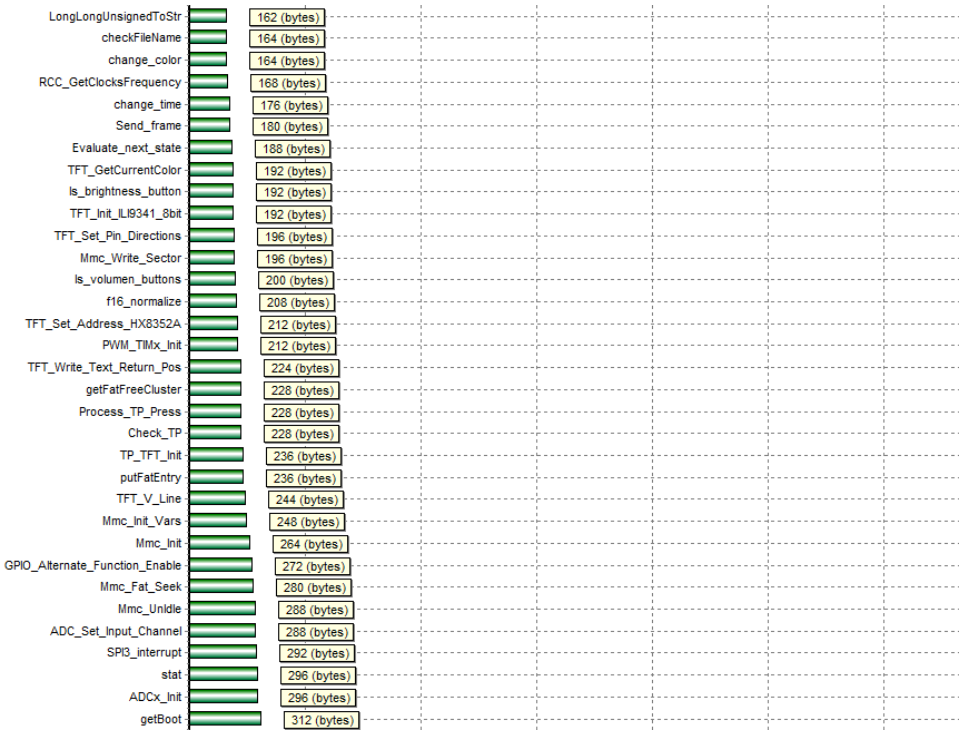


Figura 159. Palabras de configuración, microcontrolador utilizado, frecuencia de operación y tamaño y tipo de datos.

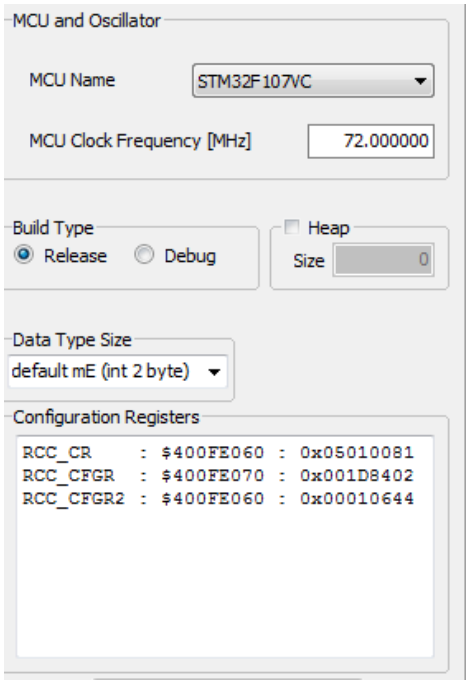


Figura 160. Especificaciones de las palabras de configuración del microcontrolador, parte 1/3.

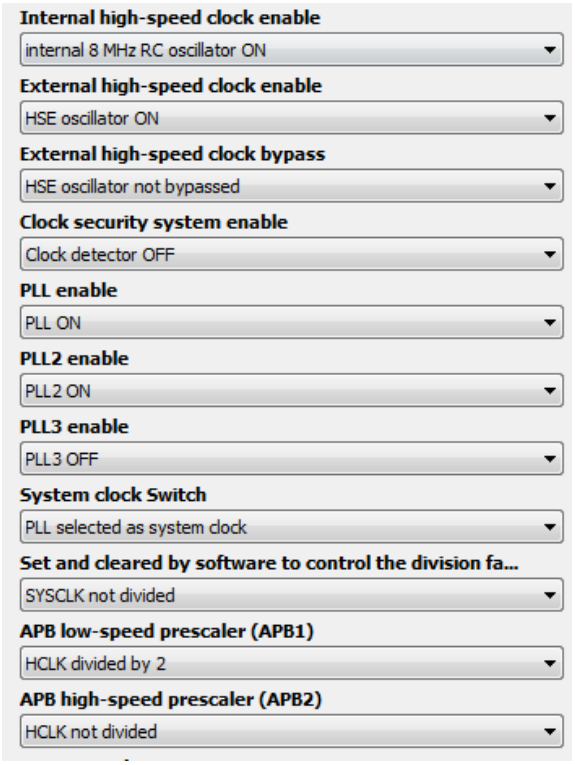


Figura 161. Especificaciones de las palabras de configuración del microcontrolador, parte 2/3.

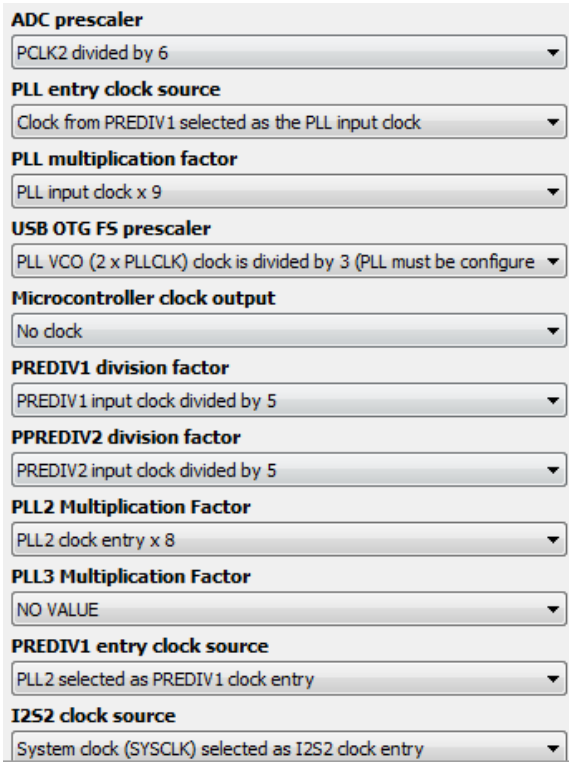
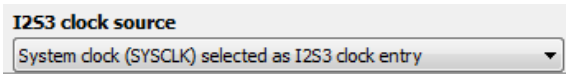


Figura 162. Especificaciones de las palabras de configuración del microcontrolador, parte 3/3.



G. ANÁLISIS DE RESULTADOS

1. GPS. Las pruebas realizadas al módulo GPS3 Click tenían el objetivo de determinar su comportamiento y determinar cómo realizar la obtención de la cadena de caracteres que representa la latitud y longitud obtenida del sistema GPS. Pese a que la información obtenida es variada, existen iniciadores y terminadores de cadenas que determinan la recepción de comandos completos. Estos sirvieron para identificar el comando GPRMC. Este comando posee dentro de sus campos la latitud y longitud obtenida desde el sistema de posicionamiento global.

Luego de realizar la extracción de la latitud y longitud satisfactoriamente. Se validó la información con respecto a la brindada por Google Maps, como se observa en la Figura 148. La comparación entre los datos obtenidos por el módulo y los datos obtenidos por Google Maps se observan en el Cuadro 34. El error es de 4.1 m. Este error es aceptable para la aplicación y requerimientos del sistema. Existen varios factores que aportan a este error. Entre ellos la misma precisión del sistema GPS que está limitada en los mejores casos a un error de 2 m (Letham, 2001:13). Además, la cantidad de satélites utilizados para determinar la posición es un factor que también afecta al error.

Existen tres tipos de cambios que podrían resultar en mejoras al sistema. La primera propuesta sería cambiar el algoritmo utilizado para extraer la latitud y longitud. Pese a que los resultados fueron satisfactorios, la carga computacional podría ser mejorada utilizando otro algoritmo que no espera a poseer todo el comando para determinar si es un comando GPRMC. Dadas las capacidades del microcontroladores, probablemente esta mejora resulte en únicamente reducción de código. La segunda propuesta consiste en mejorar la precisión del GPS aceptando los datos de latitud y longitud, únicamente cuando el módulo está utilizando más de 4 satélites (el mínimo necesario) para obtener la ubicación. Este cambio debe ser evaluado en el campo, dependiendo las condiciones del entorno puede no ser posible que el GPS se conecte a más de 4 satélites y el sistema solo dejaría de funcionar como se espera. La última propuesta es eliminar el LED que indica que el módulo se encuentra conectado al sistema GPS. Este indicador es útil en etapa de desarrollo, pero representa un consumo de energía extra e innecesaria para el usuario final.

2. Interfaz gráfica. La interfaz gráfica representa el mayor consumo de recursos de memoria en el sistema. Esto puede observarse en la Figura 157, ocho de diez de las funciones más grandes tamaño de almacenamiento son de la interfaz gráfica (pantalla TFT). La dificultad de cumplir con todos los requerimientos utilizando una interfaz que requiere tantos recursos se resuelve seleccionando correctamente la arquitectura de software y hardware para cada módulo (Cuadro 30 y Cuadro 31).

El resultado de las pruebas se considera satisfactorio. El tiempo de respuesta es aceptable (ver Cuadro 35), este tiempo se ve limitado a la tecnología utilizada, la tecnología TFT posee una alta latencia, aproximadamente de 300 ms. El apagado de pantalla tras inactividad tiene como objetivo reducir el consumo de energía, sin embargo debe reactivarse de manera satisfactoria para el usuario, esto quiere decir en un tiempo prudente. El tiempo de encendido tras inactividad varía debido a la arquitectura de software seleccionado, pero las pruebas realizadas aseguran un tiempo menor a 500 ms, tiempo que se consideró aceptable, Cuadro 36. El cambio de brillo configurable de pantalla tiene como objetivo permitir al usuario utilizar el brillo con el cual se siente cómodo. El tiempo en el cual la pantalla cambia de brillo también es variable debido a la arquitectura de software utilizada, sin embargo las pruebas cualitativas muestran que el usuario se siente satisfecho con este funcionamiento (ver Cuadro 37).

Pese a los resultados satisfactores, existe una propuesta que podría resultar en una mejora para el sistema, consiste en crear librerías propias ya que se utilizó el software VisualTFT que crea parte del código automáticamente. Esto podría resultar en código más eficiente, sin embargo aumentaría significativamente el tiempo necesario para el desarrollo del proyecto.

3. Reproducción de audio. Este módulo no requiere tantos recursos de almacenamiento en el sistema, pero es el que más tiempo de ejecución requiere. Esto se debe a que los datos a reproducir deben solicitarse a la memoria micro SD y luego de ser recibidos deben ser enviados al reproductor de audio, todo esto lo suficientemente rápido para que el audio se reproduzca con fluidez. Se seleccionó una arquitectura de software de interrupciones (ver Cuadro 31). La herramienta de desarrollo de software no posee una herramienta para determinar que tantas veces se accede a la interrupción a cargo de esta tarea. Los resultados se consideran satisfactorios, sin embargo para algunos usuarios la calidad del audio se ve reducida debido a que el cambio entre archivos no es lo suficientemente rápido. Este aspecto no puede mejorarse con el hardware utilizado (módulo MP3 click) y tendría que evaluarse otras alternativas. Los resultados se muestran en el Cuadro 38.

4. Comunicación inalámbrica. Las pruebas del módulo de comunicación estaban basados en asegurar la integridad de los datos, la confiabilidad en la entrega de los datos, la generación de una notificación a transmitir y la interpretación de una instrucción recibida. Los detalles del protocolo de comunicación corresponden a otro módulo del megaproyecto, por lo tanto solo se evalúan los resultados obtenidos en la comunicación con el dispositivo portátil según los requerimientos del sistema. Los resultados cumplen en un 100% con los requerimientos. Se considera que este módulo opera perfectamente utilizando una arquitectura de software de máquina de estados finitos implementada con la instrucción switch-case en lenguaje C. Los resultados pueden observarse en el Cuadro 39 al Cuadro 42.

5. Pruebas integradas. Al integrar todos los módulos, idealmente se espera un desempeño igual que al estar por separados. Sin embargo, el microcontrolador posee más tareas que atender y restricciones de tiempo con las cuales cumplir. Si la velocidad de operación e instrucciones por ciclo lo permiten, la selección de arquitectura de software, arquitectura de hardware, modularización, jerarquía y estandarización de interfaces, deberían resultar en una integración con un desempeño total muy parecido al modular.

Los resultados de estas pruebas se dividieron en 3: tiempo de encendido, recepción de instrucción y envío de notificación. El resultado del tiempo de encendido se observa en el Cuadro 43. Se observa que para el usuario, cualitativamente, el tiempo le parece excelente. En las pruebas cuantitativas, el tiempo de encendido y encendido tras inactividad no pueden asegurarse sino únicamente establecer un máximo. Esto se debe la arquitectura de hardware seleccionada (ver Cuadro 31), la arquitectura Round robin tiene como desventaja esta característica. Sin embargo, si el usuario lo nota imperceptible o cumple con sus expectativas el requerimiento ha sido cumplido.

Respecto a la recepción de una instrucción, la primera prueba consiste en la reproducción del audio. El audio es reproducido según la instrucción recibida. Una instrucción consiste en un conjunto de archivos de audio reproducidos uno atrás de otro. Los resultados se observan en el Cuadro 44, se observa que los comentarios van dirigidos a la naturalidad con la que la instrucción debería escucharse. Sin embargo, se consideran los resultados satisfactorios. La calidad del audio buena y el rango de volumen excelente.

El último aspecto a evaluar dentro de la recepción de una instrucción fue la reacción de la interfaz (ver Cuadro 45). Se evaluó como la interfaz gráfica reacciona al recibir una instrucción e interpretarla. Se evaluó con un resultado de mala. Cuando esto sucede, la interfaz debe variar los valores correspondientes a la calle, dirección, tiempo de dar vía y color del fondo de la pantalla principal (ver Figura 153 y su descripción), al mismo tiempo debe reproducir la instrucción recibida a través de audio. El principal problema es la cantidad de tareas que debe realizar al mismo tiempo. La reproducción de audio demanda gran cantidad de recursos computacionales y de tiempo, como se explicó con anterioridad. Al integrar todos los módulos, la arquitectura seleccionada, los recursos del microcontrolador y tareas a realizar no son suficientes para desempeñar las tareas sin poner en riesgo la integridad de las instrucciones. Esto quiere decir que la generación constante de interrupciones del módulo MP3 Click, para reproducir el audio con fluidez puede hacer que el sistema no se comunique correctamente con los demás módulos, debido a que, por ejemplo, la comunicación SPI o UART con otro módulo queda interrumpida y la información se corrompe. Este aspecto se detectó en las primeras pruebas de integración y se decidió solucionar deteniendo el funcionamiento de todas las interacciones con otros módulos, mientras se reproduce la instrucción. Sin embargo, el usuario no respondió satisfactoriamente a esta solución. Los cambios que podrían solucionar el problema son profundos y requieren cambios drásticos, por lo tanto se dejan como sugerencias para futuras mejoras. La primera opción sería implementar la comunicación entre micro SD y

MP3 a través del microcontrolador pero con un acceso directo a memoria (DMA), esta acción no interrumpe al procesador; la segunda opción puede ser, cambiar la arquitectura de software a sistema operativo de tiempo real (RTOS) y poner restricciones de tiempo para el módulo MP3.

Las pruebas integradas para el envío de notificación evalúan cualitativamente las expectativas del usuario con respecto al producto desarrollado. El Cuadro 46 muestra que el resultado es satisfactorio. Se considera que la creación de iconos más intuitivos podría ayudar a obtener mejores resultados.

6. Cumplimiento de los requerimientos. En el Cuadro 47 se autoevalúa el cumplimiento de los requerimientos del sistema obtenidos en la primera etapa de diseño e implementación. La ponderación total obtenida tras la autoevaluación es de 93.33%, la mayoría de requerimientos se cumplieron. Se considera exitosa la implementación del sistema.

Sin embargo, se considera que la interfaz gráfica podría llegar a ser un punto débil debido a que se requiere robustez para ser utilizado como herramienta de trabajo. Además, se cuestiona la portabilidad del producto desarrollado.

Para obtener un resultado real de los requerimientos que no se alcanzaron se necesita de retroalimentación de un usuario real, bajo condiciones no controladas, en el campo. Para llevar al producto a esa prueba es necesario la fabricación de un prototipo, por lo tanto, este requerimiento se encuentra fuera de alcance para los resultados de esta etapa del megaproyecto. Las limitaciones de las herramientas de desarrollo de la universidad para la fabricación de PCB dificultó la realización de un prototipo por el tiempo que requiere su diseño, fabricación y montaje, herramientas necesarias y disponibilidad de tiempo de la máquina. La otra opción es diseñar una PCB y fabricarla en el extranjero bajo reglas de diseño más permisivas, sin embargo posee las desventajas del tiempo y costos de envío significativamente mayores al de fabricarlas dentro de las instalaciones de la universidad.

Finalmente, se pone en consideración el consumo de corriente. A pesar, que existen baterías que disponen de hasta 1200 mAh y 9 V (resultando en 4 horas de descarga a máximo consumo de corriente), el consumo de corriente se considera elevado, 150 mA máximo (ver Cuadro 47). Sin embargo, debe considerarse que no se implementaron técnicas para reducir la corriente utilizada por el circuito y que al implementarlas podría reducirse considerablemente este dato. Si fuera necesario, se podría eliminar el GPS y basar la notificación del policía de tránsito respecto a cuál modulo de comunicaciones se está conectando y la localización del este, ya que su posición es fija. Y por último, podría considerarse cambiar el tipo de interfaz gráfica a una menos amigable, pero que consume menos corriente. Las soluciones propuestas son las más sencillas de implementar, pero existen otras como poner en bajo consumo energético al microcontrolador mientras no se utiliza.

7. Costos. Los costos se encuentran en el Anexo B. En esta sección se separaron los costos de desarrollo de los costos de la creación del sistema embebido. En los anexos se muestran los costos del desarrollo del proyecto. En los anexos se muestran los costos para recrear el sistema desarrollado. Pese a que los costos para la recreación del sistema embebido son altos, debe considerarse que para crear un producto final, los costos se reducen significativamente debido a que se utilizan componentes directamente de los proveedores y no tarjetas (módulos). Esto reducirá el costo significativamente, y el volumen del producto final.

H. CONCLUSIONES

Se diseñó e implementó un dispositivo de asistencia portátil para un policía de tránsito cumpliendo con un 90.0% de los requerimientos funcionales, no funcionales, de arquitectura y físicos.

Las debilidades principales del sistema desarrollado se encuentran en la robustez física de la interfaz gráfica (pantalla TFT) para ser utilizada como herramienta de trabajo y en el tamaño (volumen) del dispositivo para ser considerado como portátil.

Se diseñó e implementó una interfaz gráfica para la interacción entre el agende de policía de tránsito y el sistema de medición y análisis de tránsito vehicular.

Se logró la comunicación inalámbrica del dispositivo de asistencia portátil con el resto del sistema de medición de análisis de tránsito vehicular.

El dispositivo permite la ubicación del agente de policía de tránsito con el dispositivo de asistencia portátil, en las vías de tránsito al aire libre, por medio de GPS.

El dispositivo reproduce por audio las instrucciones sugeridas para el agente de policía de tránsito para la gestión de tránsito vehicular. La reproducción del audio limita la operabilidad del sistema.

El dispositivo permite la notificación de incidentes de tránsito a través del dispositivo de asistencia portátil.

I. RECOMENDACIONES

Se recomienda evaluar la posibilidad de utilizar un sistema operativo en tiempo real (RTOS) para cumplir con el requerimiento de reproducción de audio sin reducir la operabilidad del sistema. Si se lleva a cabo, debe considerarse cambiar de compilador debido a que el utilizado no posee dicha característica. Se recomienda utilizar Keil debido a la amplia documentación que posee en el tema.

Se recomienda realizar las pruebas de campo para determinar la portabilidad y robustez del dispositivo y determinar si es necesario cambiar la interfaz gráfica. Se deberá evaluar si otra interfaz reduce o no significativamente la calidad de interacción que la pantalla TFT brinda, por el color, tamaño y pantalla táctil. El cambio de pantalla reduce significativamente el tamaño mínimo del dispositivo. Si aun así no es lo suficientemente pequeño, deberá considerarse realizar un prototipo sin tarjetas de módulos, sino adquiriendo todos los componentes de proveedores y de preferencia de montaje superficial.

Se recomiendo utilizar técnicas de software para reducir el consumo de energía. Apagar el GPS si no es utilizado, poner en modo de ahorro de energía al microcontrolador sino está siendo utilizado. Diseñar el circuito para que únicamente utilice 3.3 V y eliminar el regulador de voltaje de 5 V.

Finalmente, se recomienda seguir utilizando la misma herramienta de desarrollo de hardware debido al tiempo y esfuerzos que redujo en el desarrollo.

XI. CONCLUSIONES

Respecto a las estrategias utilizadas por los algoritmos en las pruebas realizadas, no es posible determinar si alguna presenta mejor rendimiento debido a que los dos mejores algoritmos encontrados utilizan estrategias diferentes. Sin embargo la estategia de tiempos dinámicos presentó mejores resultados en la liberación de vehículos en el mapa.

En cuanto al bajo rendimiento de los algoritmos, con la información encontrada no se puede determinar una causa específica. Sin embargo se logró identificar en que puntos presentan deficiencias cada uno de los algoritmos de inteligencia artificial para poder contar con una revisión de su implementación.

Por otro lado, no se recomienda utilizar el algoritmo que presentó mejores resultados debido a su factor de aleatoriedad. El candidato a implementarse en caso es el algoritmo genético implementado en el *Módulo de diseño e implementación de un algoritmo genético para la optimización de ciclos de semáforo en intersecciones de una cuadrícula*.

El programa del sistema de control permite conectar hasta 255 dispositivos portátiles, con los cuales debe establecerse una conexión a través de un intercambio de mensajes específicos para dar seguridad. Dependiendo del tipo de instrucción que la unidad terminal envíe, el módulo de control envía una instrucción según es solicitado o guarda la notificación recibida. La sesión se cierra al finalizar la transacción.

El dispositivo de asistencia portátil para un policía de tránsito permite la recepción de instrucciones para la gestión del tránsito vehicular, la localización geográfica del agente y la notificación de incidentes a través de una interfaz gráfica, una pantalla táctil y la reproducción de audio.

XII. RECOMENDACIONES

El simulador de TCA utilizado para las pruebas demostró cumplir con su objetivo, sin embargo algunos puntos pueden ser mejorados para contar con la disponibilidad de realizar otro tipo de pruebas. Uno de estos puntos es la velocidad máxima permitida que no puede ser modificada durante la ejecución y contar con uan velocidad máxima baja podría resultar perjudicial para los algoritmos de inteligencia artificial. El segundo punto es la capacidad de modificar la tasa con la que se generan vehículos, para realizar la prueba de densidad vehicular fue necesario ralizar simulaciones independientes eliminando la posibilidad de evaluar el comportamiento al modificar las condiciones del mapa durante la simulación.

En cuanto a los algoritmos de inteligencia artificial se recomienda realizar una revisión para mejorar los puntos débiles identificados. Para el algoritmo genético se debe realizar un entrenamiento más riguroso utilizando los parámetros correspondientes para cada escenario y en un equipo con mejor poder de procesamiento para que el entramiento sea viable. Con el algoritmo multiagente con aprendizaje por refuerzo, se recomienda revisar la sección de toma de decisiones y verificar que las mismas tengan efecto dentro del simulador para evitar que llegue a un punto con todos los vehículos detenidos.

Realizar pruebas de campo con los dispositivos de asistencia portátil y de comunicación para determinar limitaciones y mejoras.

XIII. BIBLIOGRAFÍA

Acurio, J. (s.f.). *Comunicaciones digitales (CxD)*. Recuperado el 26 de Septiembre de 2015, de http://es.slideshare.net/Jaime_hernan/cd-1-introduccion-y-conceptos-basicose

Arduino. (s.f.). *Compare board specs*. Recuperado el 24 de Abril de 2015, de https://www.arduino.cc/en/Products.Compare

Arnouse Digital Devices Corp. (s.f.). *BioDigital PC*. Recuperado el 24 de Abril de 2015, de http://www.arnousedigitaldevices.com/products/

Balaji, P. G., German, X., y Srinivasan, D. (2010). *Urban traffic signal control using reinforcement learning agents*. IET Intelligent Transport Systems, 4(3): pp. 177–188.

Barnett, Richard; L. O'Cull y S. Cox. *2004 Embedded C Programming and the Microchip PIC, Volume 1*. Canada: Cengage Learning. 497.Beagleboard. (s.f.). *Beagleboard: BeagleBoneBlack*. Recuperado el 24 de Abril de 2015, de http://elinux.org/Beagleboard:BeagleBoneBlack

Bates, Martin. 2004. *PIC Microcontrollers: An Introduction to Microcontrollers*. 2ª ed. Oxford: Elsevier. 372.

Benjaafar, S., Dooley K. y Setyawan W. *Cellular Automata for Traffic Flow Modeling*. Center for Transportation Studies, University of Minnesota, 1997

Berczuk S. y B. Appleton, *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, Addison-Wesley Longman Publishing Co., 2002, iSBN 0201741172.

Bies, L. (s.f.). *On-line CRC calculation and free library*. Recuperado el 23 de Agosto de 2015, de http://www.lammertbies.nl/comm/info/crc-calculation.html

Bird, D., & Hardwood, M. (2003). *Network+ training guide*. Estados Unidos: Que Certification.

Braun R., F. Weichenmeier (2005). *Automatische Offline-Optimierung der lichtsignaltechnischen Koordinierung des mIV im stadtischen Netz unter Verwendung genetischer Algorithmen*. Straßenverkehrstechnik, vol. 5, pp. 232-238.

Breese, J., D. Heckerman y C. Kadie, *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*, Microsoft Research, Microsoft Corporation, 1998.

Brockfeld E. *et. al. Optimizing Traffic Lights in a Cellular Automaton Model for City Traffic*. Institut für Theoretische Physik, Universität zu Köln, Alemania, 2008.

Byford, J. (s.f.). *Building a basic communication network using XBee DigiMesh*. Recuperado el 26 de Marzo de 2015, de http://www.egr.msu.edu/classes/ece480/capstone/spring13/group02/documents/Application_Note_XBee_Communication.pdf

Carlson, B., Crilly, P., & Rutledge, J. (2002). *Communication Systems: An introduction to signals and noise in electrical communication* (Cuarta ed.). Nueva York: McGraw-Hill.

Carrano E., *et. al. A preliminary comparison of tree encoding schemes for evolutionary algorithms*. Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on 7-10 Oct 2007. On pages: 1969-1974.

Catsoulis, John. 2005. *Designing Embedded Hardware*. 2ª ed. Estados Unidos de América: O'Reilly. 540.

CCM Benchmark Group. (s.f.). *Estándar GPRS (servicio general de paquetes de radio)*. Recuperado el 23 de Agosto de 2015, de http://es.ccm.net/contents/680-estandar-gprs-servicio-general-de-paquetes-de-radio

CCM Benchmark Group. (s.f.). *Introducción - Bases de datos*. Recuperado el 22 de Agosto de 2015, de http://es.ccm.net/contents/66-introduccion-bases-de-datos

CCM Benchmark Group. (s.f.). *Verificación de errores*. Recuperado el 23 de Agosto de 2015, de http://es.ccm.net/contents/59-verificacion-de-errores

Chin, Y. K., Bolong, N., Kiring, A., Yang, S. S. y Teo, K. T. K. (2011). *Q-learning based traffic optimization in management of signal timing plan*. International Journal of Simulation: Systems, Science and Technology, 12(3): pp. 29–35. Retrieved from http://ijssst.info/Vol-12/No-3/paper5.pdf

Chowdhury, D., L. Santen y A. Schadschneider. *Statistical Physics of Vehicular Traffic and Some Related Systems*. Physics Reports, 329(4-6), 2000 pp. 199-329. doi:10.1016/S0370-1573(99)00117-9.

Cisco. (s.f.). *Ethernet technologies*. Recuperado el 13 de Febrero de 2015, de http://docwiki.cisco.com/wiki/Ethernet_Technologies

Computer Networking. (s.f.). *10BaseT 10BaseF 10Base2 5-4-3 rule 10Base5 100BaseFX 100BaseT4 100BaseTX*. Recuperado el 13 de Febrero de 2015, de http://computernetworkingnotes.com/network-technologies/10base-ethernet.html

Cook, J. A.; J. S. Freudenberg. 2008. *Embedded Software Architecture*. University of Michigan. 14.

Cook, G. (s.f.). *Catalogue of parametrised CRC algorithms with 16 bits*. Recuperado el 23 de Agosto de 2015, de http://reveng.sourceforge.net/crc-catalogue/16.htm

Deb, K. *Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems*. Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology Kanpur, India, 1999.

Debian Project. (s.f.). *Welcome to Raspbian*. Recuperado el 1 de Octubre de 2015, de https://www.raspbian.org/

Delorme, M. *An introduction to cellular automata*, in: M. Delorme, J. Mazoyer (Eds.), Cellular Automata – a parallel model, Kluwer Academic Publishers Group, 1998, iSBN 0792354931.

Digi International. (s.f.). *Flexible wireless design*. Recuperado el 19 de Agosto de 2015, de http://www.digi.com/lp/xbee/

Digi International. (s.f.). *Maximizing Range*. Recuperado el 13 de Abril de 2015, de www.digi.com

Digi International. (s.f.). *Received signal strength indicator (RSSI)*. Recuperado el 26 de Septiembre de 2015, de https://docs.digi.com/pages/viewpage.action?pageId=2626044

Digi International. (s.f.). *RF feature articles*. Recuperado el 12 de Agosto de 2015, de http://www.digi.com/technology/rf-articles/rf-basics

Digi International. (s.f.). *What is API (Application Programming Interface) mode and how does it work?* Recuperado el 1 de Octubre de 2015, de http://knowledge.digi.com/articles/Knowledge_Base_Article/What-is-API-Application-Programming-Interface-Mode-and-how-does-it-work

Digi International. (s.f.). *XBee & XBee-PRO OEM RF module antenna considerations*. Recuperado el 1 de Octubre de 2015, de http://ftp1.digi.com/support/images/XST-AN019a_XBeeAntennas.pdf

Digi. 2015. *XBee/RF Solutions, Modules*. http://www.digi.com/products/xbee-rf-solutions/modules/xbee-series1-module. [24 de septiembre 2015]

Duvall P., S. Matyas y A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, Inc., 2007, iSBN 9780321336385.

Electric Imp. (s.f.). *UART explained*. Recuperado el 26 de Septiembre de 2015, de https://electricimp.com/docs/resources/uart/

Esser, J. y M. Schreckenberg. *Microscopic Simulation of Urban Traffic based on Cellular Automata*. Gerhard-Mercator-Universitat-Duisburg, Duisburg, Germany, 1997.

Fernandez E. (2015). *Parque vehicular habría crecido 154% en 10 años*. Diario de Centroamérica, 16 de febrero de 2015. Disponible en internet en <http://www.dca.gob.gt/index.php/nacional/item/25882-parque-vehicular-habr%C3%ADa-crecido-154-en-10-a%C3%B1os>

Flaviu, C. *Exception Handling and Software-Fault Tolerance*. Computing Laboratory, University of Newcastle, 1999.

Fortin F., F. de Rainville, *et. al*. 2012. *DEAP: Evolutionary Algorithms Made Easy*. Journal of Machine Learning Research. Volume 3 pages 2171-2175.

Foy, M. *et. al*. *Signal Timing Determination Using Genetic Algorithms*. Transportation Research. Record 1365, National Research Council, Washington, D.C., 1992, pp. 108-115.

France, J. y Ghorbani, A. (2003). *A multiagent system for optimizing urban traffic*. IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003. (pp. 411-414)

Garlan, David; M. Shaw. 1994. *An Introduction to Software Architecture*. Pittsburgh: Carnegie Mellon University. 42.

Gershenson C. y D Rosenblueth. *Modeling self-organizing traffic lights with elementary cellular autómata*. Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, México, 2009.

Ghanbari, A., E. Hadavandi y S. Abbasian-Naghneh. *Comparison of Artificial Intelligence based Techniques for Short Term Load Forecasting*. IEEE Computer Society, 2010.

GitHub, Inc. (s.f.). *GitHub*. Recuperado el 1 de Octubre de 2015, de https://github.com/

Graepel, T., Herbrich, R. y Gold, J. (2004). *Learning to Fight*. Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education (pp. 193–200). Retrieved from http://www.herbrich.me/papers/graehergol04.pdf

Gregoire, P.-L., Desjardins, C., Laumonier, J. y Chaib-draa, B. (2007). *Urban Traffic Control Based on Learning Agents*. 2007 IEEE Intelligent Transportation Systems Conference, 916–921.

Hohpe G., B. Woolf, *et. al*. *Enterprise Integration Patterns Designing, Building, and Deploying Messaging Solutions*, Pearson Education, Inc., 2011, iSBN 0321200683.

Hong, Y. *et. al*. *Estimation of optimal green time simulation using fuzzy neural network*. Fuzzy Systems Conference Proceedings, Volumen 2, 1999.

Innova Technologies. (s.f.). *Telemetría*. Recuperado el 23 de Agosto de 2015, de http://www.radiocomunicaciones.net/telemetria.html

IT Law Wiki. (s.f.). *Point-to-multipoint topology*. Recuperado el 1 de Octubre de 2015, de http://itlaw.wikia.com/wiki/Point-to-multipoint_topology

Jadhao, M. y Kulkarni, M. (2012). *Reinforcement Learning Based for Traffic Signal Monitoring and Management*. International Journal of Engineering Research and Technology, 1(4): pp. 1–4. Retrieved from http://www.ijert.org/view-pdf/176/reinforcement-learning-based-for-traffic-signal-monitoring-and-management-

Kamal, Raj. 2008. *Embedded Systems* 2E. 2ª ed. Najar: McGraw-Hill. 348.

Keller, R. y J. Tessier. (1997). *Layla: A Pattern-based Framework for Network Management Interfaces*, Departement d'informatique et de recherche opérationnelle Université de Montréal.

Kleinjohann, Bernd. 2013. *Architecture and Design of Distribute Embedded Systems*. Alemania: Springer. 236.

Kuhne, T. (1997), *The Translator Pattern - External Functionality with Homomorphic Mappings*, Department of Computer Science, Darmstadt University of Technology.

Lagoudakis, M. G. (2011). *Value Function Approximation*. En C. Sammut y G. I. Webb (Eds.), Encyclopedia of Machine Learning (pp. 1011-1021). Springer.

Letham, Lawrence. 2001. *GPS fácil. Uso de sistema de posicionamiento global*. Barcelona: Editorial Paidotribo. 284.

Li, K., Z. Zhang y J. Kou. *Breeding Software Test Data with Genetic-Particle Swarm Mixed Algorithm*, Journal of Computers, Volumen 5, No. 2, Febrero 2010.

Libelium. (s.f.). *Wireless sensor network with Waspmote and Meshlium*. Recuperado el 17 de Abril de 2015, de http://www.libelium.com/downloads/documentation/wsn-waspmote_and_meshlium_eng.pdf

Linares, M. (s.f.). *Sistema GPRS*. Recuperado el 23 de Agosto de 2015, de http://www.uv.es/~montanan/redes/trabajos/GPRS.doc

López, J. R. (s.f.). *El modelo relacional*. Recuperado el 1 de Octubre de 2015, de http://docencia.lbd.udc.es/bdd/teoria/tema2/2.3.1.-ElModeloRelacional.pdf

Maerivoet S. y B. De Moor. *Cellular Automata Models of Road Traffic*. Physics Reports, vol. 419, nr. 1, pages 1-64.

Maerivoet, Sven y Bart De Moor. *Transportation Planning and Traffic Flow Models*. Katholieke Universiteit Leuven, Leuven, Belgium, 2008.

Mannion, P., Duggan, J., y Howley, E. (2015). *Parallel Reinforcement Learning for Traffic Signal Control*. Procedia Computer Science, 52(0): pp. 956–961. Retrieved from http://www.sciencedirect.com/science/article/pii/S1877050915009722

Marsland, S. (2014). *Machine Learning An Algorithmic Perspective*. 2nd ed. CRC Press.

MCI Electronics. (s.f.). *¿Qué es XBee?* Recuperado el 22 de Agosto de 2015, de http://xbee.cl/que-es-xbee/

Medvidovic, Nenad; S. Malek y M. Mikic-Rakic. 2003. *Software Architectures and Embedded Systems*. Los Angeles: University of Southern California. 10.

Mikroelectronika. Store. http://www.mikroe.com/store/. [24 de septiembre 2015]

Mitchel, M. *An Introduction to Genetic Algorithms*. MIT Press. 209 páginas.

Nagel K. y M. Schreckenberg. *A cellular automaton model for freeway traffic*. Journal de Physique I, EDP Sciences, 1992, 2 (12), pp.2221-2229

Noergaard, Tammy. 2012. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. 2ª ed. Waltham: Newnes. 672.

Oracle Corporation. (s.f.). *What is MySQL*. Recuperado el 22 de Agosto de 2015, de https://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html

Oracle. (s.f.). *MySQL*. Recuperado el 26 de Septiembre de 2015, de http://www.mysql.com/

Paolo, Maurizio Di. 2014. *Embedded Systems Design for High-Speed Data Acquisition and Control*. New York: Springer. 155.

Parallella. (s.f.). *Parallella-1.x reference manual*. Recuperado el 24 de Abril de 2015, de http://www.parallella.org/docs/parallella_manual.pdf

Paramio, C. (s.f.). *GitHub, el servicio donde alojar tus repositorios Git (como el nuestro)*. Recuperado el 22 de Agosto de 2015, de http://www.genbetadev.com/sistemas-de-control-de-versiones/conociendo-github-el-servicio-donde-alojar-tus-repositorios-git-como-el-nuestro

Panait, L. y Luke, S. (2005). *Cooperative Multi-Agent Learning: The State of the Art*. Autonomous Agents and Multi-Agent Systems, 11, (pp. 387–434).

Parsons, J. J., & Oja, D. (2011). *New perspectives on computer concepts*. Boston: Cengage Learning.

Pham, T., Brys, T., y Taylor, M. (2013). *Learning Coordinated Traffic Light Control*. Proceedings of the Adaptive and Learning Agents Workshop. Retrieved from http://www.eecs.wsu.edu/ taylorm/Publications/ALA13-Pham.pdf

Poole, I. (s.f.). *GPRS general packet radio service tutorial*. Recuperado el 26 de Marzo de 2015, de http://www.radio-electronics.com/info/cellulartelecomms/gprs/gprs_tutorial.php

Poole, I. (s.f.). *GSM: global system for mobile communications tutorial*. Recuperado el 26 de Marzo de 2015, de http://www.radio-electronics.com/info/cellulartelecomms/gsm_technical/gsm_introduction.php

Poulsen, L. (s.f.). *How far and how fast*. Recuperado el 13 de Febrero de 2015, de http://www.afar.net/tutorials/how-far/

Rajeswaran, S. y S.Rajasekaran. *A Study Of Vehicular Traffic Flow Modeling Based On Modified Cellular Automata*. IOSR Journal of Mathematics, 2013, 4 (5), pp. 32-38

Rankin, D. B. (s.f.). *Network topologies*. Recuperado el 1 de Octubre de 2015, de http://www.unc.edu/~dbr/inls182/Old%20Network%20Topologies.htm

Raspberry Pi Foundation. (s.f.). *GPIO: Raspberry Pi models A and B*. Recuperado el 1 de Octubre de 2015, de https://www.raspberrypi.org/documentation/usage/gpio/

Raspberry Pi Foundation. (s.f.). *Raspberry Pi 2 model B*. Recuperado el 24 de Abril de 2015, de https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

Raspberry Pi Foundation. (s.f.). *What is a Raspberry Pi?* Recuperado el 12 de Agosto de 2015, de https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

Rickert, M., K. Nagel, M. Schreckenberg y A. Latour. *Two Lane Traffic Simulations using Cellular Automata*. Center for Parallel Computing, Universität zu Köln, 50923 Köln, Alemania, 2008.

Rodríguez M. (2015). *Prevén complicaciones en la ciudad por aumento del parque vehicular*. Diario La Hora, 03 de enero de 2015. Disponible en internet en <http://lahora.gt/preven-complicaciones-en-la-ciudad-por-aumento-del-parque-vehicular/>

Rosenblueth, D. y C. Gershenson. *A Model of City Traffic Based on Elementary Cellular Automata*. Universidad Nacional Autonoma de México, Ciudad Universitaria. México D.F., México.

Russell, S. J. y Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. 3rd ed. New Jersey: Prentice Hall.

Sanchez-Medina J, M. Galán y E. Rubio. (2004) *Genetic algorithms and cellular automata: a new architecture for traffic light cycles optimization*. Evolutionary Computation (CEC), 2004 IEEE Congress, On page(s): 1668 - 1674 Vol 2.

Sanchez-Medina J, M. Galán y E. Rubio. (2005) *Stochastic Vs. Deterministic Traffic Simulator. Comparative Study for Its Use within a Traffic Light Cycles Optimization Architecture*. Conference: Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach: First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005, Las Palmas, Canary Islands, Spain, June 15-18, 2005, Proceedings, Part II.

Sanchez-Medina J, M. Galán y E. Rubio. (2010) *Traffic Signal Optimization in 'La Almozara' District in Saragossa Under Congestion Conditions, Using Genetic Algorithms, Traffic Microsimulation, and Cluster Computing*. IEEE Transactions on Intelligent Transportation Systems (Impact Factor: 2.47). 04/2010; 11(1):132 - 141. DOI: 10.1109/TITS.2009.2034383

Sanchez-Medina J, M. Galán y E. Rubio. (2015) *Genetic Algorithms and Cellular Automata for Traffic Light Cycles Optimization. Scalability Study*. Centro de Innovación para la Sociedad de la Información (C.I.C.E.I.), Universidad de Las Palmas de Gran Canaria.

Sanner M. *Python: A Programming Language for Software Integration and Development*. The Scripps Research Institute, 1999.

Schwartz, H. M. (2014). *Multi-Agent Machine Learning: A Reinforcement Approach*. John Wiley & Sons, Inc.

Science aid. (s.f.). *Communication systems*. Recuperado el 26 de Septiembre de 2015, de http://scienceaid.co.uk/physics/waves/communication.html

Sears, Andrew; J. Jacko. 2007. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. 2ª Ed. New York: CRC Press. 1384.

Seeed Studio. (s.f.). *Seeeduino-Stalker v3*. Recuperado el 24 de Abril de 2015, de http://www.seeedstudio.com/wiki/Seeeduino-Stalker_v3

Seeed Studio. (s.f.). *Zigbee networking with XBee series 2 and Seeed's products*. Recuperado el 26 de Marzo de 2015, de http://www.seeedstudio.com/wiki/Zigbee_Networking_with_XBee_Series_2_and_Seeed's_Products

Seiffertt, J. y Wunsch, D. (2012). *Reinforcement Learning Control with Time-Dependent Agent Dynamics*. En F. L. Lewis y D. Liu (Eds.), Reinforcement Learning and Approximate Dynamic Programming for Feedback Control (pp. 203-220). Wiley-IEEE Press.

Sen, S. y Weiss, G. (1999). *Learning in Multiagent Systems*. En G. Weiss (Ed.), Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence (pp. 259-298). MIT Press.

Sherstov, A. A. y Stone, P. (2005). *Function approximation via tile coding: Automating parameter choice*. Abstraction, Reformulation and Approximation. 3607, pp. 194-205. Springer.

Simon, David E. 2005. *An Embedded Software Primer*. Patparganj: Pearson. 225.

Smith S. *System Software Integration: An Expansive View*. University of Texas at Austin, Estados Unidos, 2009.

Sparkfun. (s.f.). *XBee buying guide*. Recuperado el 13 de Febrero de 2015, de https://www.sparkfun.com/pages/xbee_guide

Stevanovic, A., P. T. Martin, J. Stevanovic (2007). *VISGAOST: VISSIM-based Genetic Algorithm Optimization of Signal Timings*. In Proceedings 86th Transportation Research Board Meeting.

Sun, D., R. F. Benekohal, S. T. Waller (2003). *Multiobjective traffic signal timing optimization using nondominated sorting genetic algorithm*. In Proceedings IEEE Intelligent Vehicles Symposium, pp. 198-203.

Superintendencia de Telecomunicaciones. (s.f.). *Ley general de telecomunicaciones*. Recuperado el 30 de Septiembre de 2015, de http://www.itu.int/ITU-D/treg/Legislation/Guatemala/leygen.pdf

Sutton, R. S. y Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge Univ Press.

Tahilyani, S., Darbari, M. y Shukla, P. K. (2013). *Soft Computing Approaches in Traffic Control Systems: A Review*. AASRI Procedia, 4, pp. 206–211.

Telefónica. (s.f.). *GPRS*. Recuperado el 26 de Septiembre de 2015, de http://www.telefonica.com.sv/Empresas/svas_gprs.html

The Fan Club. (s.f.). *How to setup a USB 3G Modem on Raspberry PI using usb_modeswitch and wvdial*. Recuperado el 28 de Octubre de 2015, de https://www.thefanclub.co.za/how-to/how-setup-usb-3g-modem-raspberry-pi-using-usbmodeswitch-and-wvdial

Tisch School of the Arts. (s.f.). *Synchronous serial communication: the basics*. Recuperado el 1 de Octubre de 2015, de https://itp.nyu.edu/physcomp/lessons/serial-communication/synchronous-serial-communication-the-basics/

Tomasi, Wayne. 2003. *Sistemas de comunicaciones electrónicas*. 4ª ed. México: Pearson Educación. 948.

Ton, D. (s.f.). *Radio frequency (RF) data communications*. Recuperado el 12 de Agosto de 2015, de https://courses.cs.washington.edu/courses/cse477/99au/tutorials/RF/rf.ppt

Torres, M. (s.f.). *Diagrama a bloques de un sistema de comunicación*. Recuperado el 26 de Septiembre de 2015, de http://marcotorresluna.blogspot.com/2015/01/diagrama-bloques-de-un-sistema-de.html

Traylor, R. (s.f.). *USART and asynchronous communication*. Recuperado el 12 de Agosto de 2015, de http://web.engr.oregonstate.edu/~traylor/ece473/lectures/uart.pdf

Turky A., M. Ahmad, M. Yusoff. *The Use of Genetic Algorithm for Traffic Light and Pedestrian Crossing Control*. IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.2, February 2009.

U.S. Department of transportation, Federal Highway Administration. (s.f.). *Chapter 2. Fundamentals of telecommunications. Page 1 of 3*. Recuperado el 4 de Febrero de 2015, de http://ops.fhwa.dot.gov/publications/telecomm_handbook/chapter2_01.htm

U.S. Department of transportation, Federal Highway Administration. (s.f.). *Chapter 2. Fundamentals of telecommunications. Page 2 of 3*. Recuperado el 4 de Febrero de 2015, de http://ops.fhwa.dot.gov/publications/telecomm_handbook/chapter2_02.htm

Vahid, Frank; T. Givargis. 2009. *Embedded System Design: A Unified Hardware/Software Approach*. Riverside: university of California. 103.

van Hasselt, H. (2012). *Reinforcement Learning in Continuous State and Action Spaces*. En M. Wiering y M. van Otterlo (Eds.), Reinforcement Learning: State-of-the-Art (pp. 207-251). Springer.

White, Elecia. 2012. *Making Embedded Systems*. Estados unidos: O'Reilly Media: 445.

Wiering, M. (2000). *Multi-agent reinforcement learning for traffic light control*. Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000). (pp. 1151–1158).

Wiewiora, E., Cottrell, G., y Elkan, C. (2003). *Principled Methods for Advising Reinforcement Learning Agents*. Proceedings of the 20th International Conference on Machine Learning (ICML), (1999), 792–799. Retrieved from http://www.aaai.org/Papers/ICML/2003/ICML03-103.pdf

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. 2nd ed. Wiley.

Yadav, A. y Shrivastava, S. (2010). *Evaluation of reinforcement learning techniques*. Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia, 88–92. Retrieved from http://dl.acm.org/citation.cfm?id=1963578

Yannick, G., O. Gagnon *et. al*. 2014. *Once you SCOOP, no need to fork*. Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment. ACM.

Yun I., B. Park (2005). *Stochastic optimization method for coordinated actuated signal systems*. Technical Report UVACTS-15-0-102, Center for Transp. Studies, Univ. of Virginia.

Zhiyong, *et. al*. (2006) *Immunity genetic algorithms based adaptive control method for urban traffic network signal*. Control Theory & Applications, 2006, 23(1): pp. 119-125.

Zurell, Kirk. 2000. *C Programming for Embedded Systems*. Lawrence: R&D Books. 191.

XIV. ANEXOS

A CÓDIGO FUENTE DE SIMULADOR

```
"""
10/2015
Luis Valdeavellano - 11218
Universidad del Valle de Guatemala

Implementation of a Traffic Cellular Automata simulator
"""

import random

class Cell:
    id = 0
    car = None
    viewer_address = None
    rule = None
    rule_class = None
    p = None
    topology = None

    def __init__(self):
        self.id = Cell.id
        Cell.id += 1

    def __repr__(self):
        return "<Cell: %s (%s)>" % (self.id, '.' if self.car is None
            else self.car.speed)

    def apply_rules(self):
```

```
28          self.car = self.p.car
29
30  class ProvisionalCell:
31      car = None
32      recipient = False
33
34      def __init__(self, cell):
35          self.car = cell.car
36
37  class ProvisionalCar:
38      speed = 0
39      cell = None
40
41      def __init__(self, car):
42          self.speed = car.speed
43          self.cell = car.cell
44
45  class Rule:
46      cell = None
47      car = None
48      front_gap = 0
49      front_cells = []
50
51      def __init__(self, cell):
52          self.cell = cell
53          self.cell.p = ProvisionalCell(self.cell)
54          if cell.car is not None:
55              self.car = cell.car
56              self.car.p = ProvisionalCar(self.car)
57          self.populate()
58
59      def populate(self):
60          if self.car is None:
61              return
62          self.front_gap = 0
63          self.front_cells = self.cell.get_front_cells(self.car.v_max,
```

```
            self.car.route)
64          for cell in self.front_cells:
65              if cell.car is not None:
66                  break
67              self.front_gap += 1
68
69      def pre_setting(self):
70          pass
71
72      def calculate(self):
73          if self.car is None:
74              return
75          self.nasch_rules()
76
77      def nasch_rules(self):
78          # rule 1 (acceleration):
79          self.car.p.speed = min(self.car.p.speed + 1, self.car.v_max)
80
81          # rule 2 (collide avoidance)
82          self.car.p.speed = min(self.car.p.speed, self.front_gap)
83
84          # rule 3 (stochastic deceleration)
85          if random.random() < self.car.decelerate_rate:
86              self.car.p.speed = max(0, self.car.p.speed - 1)
87
88          # move car
89          if self.car.p.speed > 0:
90              self.car.p.cell = self.front_cells[self.car.p.speed - 1]
91              self.car.p.cell.p.car = self.car
92              self.car.p.cell.p.recipient = True
93              if not self.cell.p.recipient:
94                  self.cell.p.car = None
95
96      def apply_(self):
97          self.cell.apply_rules()
98          if self.car is not None:
```

```
99              self.car.apply_rules()
100
101 class StreetRule(Rule):
102
103     def pre_setting(self):
104         super().pre_setting()
105         self.calculate_changing_lane_rates()
106         self.change_lane_rules()
107
108     def calculate_changing_lane_rates(self):
109         if self.car is None:
110             return
111         base = self.car.base_lane_changing_rate
112         if self.car.route is None:
113             self.car.right_change_rate = 0.5
114             self.car.lane_changing_rate = base
115             return
116
117         dif = self.cell.lane - self.car.route.entrance_lane
118         if dif != 0:
119             if self.cell.cells_to_end <= 1:
120                 self.car.waits_for_lane_change += 1
121                 if self.car.waits_for_lane_change >=
                        self.car.changing_route_max_wait:
122                     self.car.route =
                            random.choice(self.cell.connection.routes)
123                     self.car.waits_for_lane_change = 0
124 #                   print('deadlock avoidance')
125
126             self.car.right_change_rate = 1 if dif < 0 else 0
127             self.car.lane_changing_rate = (self.cell.cell /
                    self.cell.street.length) * (1 - base) + base
128         else:
129             self.car.right_change_rate = 0.5
130             self.car.lane_changing_rate = (self.cell.cells_to_end /
                    self.cell.street.length) * base
```

```
131             self.car.waits_for_lane_change = 0
132
133
134     def change_lane_rules(self):
135         if self.car is None or random.random() >
                self.car.lane_changing_rate:
136             return
137         side = 'right' if random.random() < self.car.right_change_rate
                else 'left'
138         dest_cell = getattr(self.cell, '%s_cell' % side)
139         if dest_cell is not None and dest_cell.car is None:
140             if not dest_cell.p.recipient:
141                 self.car.p.cell = dest_cell
142                 dest_cell.p.car = self.car
143                 dest_cell.p.recipient = True
144                 if not self.cell.p.recipient:
145                     self.cell.p.car = None
146
147 class IntersectionRule(Rule):
148     pass
149
150 class StreetCell(Cell):
151     rule_class = StreetRule
152     street = None
153     lane = None
154     cell = None
155     cells_to_end = None
156     front_cell = None
157     right_cell = None
158     left_cell = None
159
160     def get_front_cells(self, n, route=None):
161         cells = self.street.cells[self.lane][self.cell + 1 :]
162         dif = n - len(cells)
163         if dif > 0 and cells[-1].connection is not None:
164             cells += cells[-1].get_front_cells(dif, route)
```

```
165             return cells[:n]
166
167
168     class IntersectionCell(Cell):
169         rule_class = IntersectionRule
170         routes = None
171         intersection = None
172
173         def get_front_cells(self, n, route=None):
174             if route is None:
175                 return []
176             cells = route.cells[route.cells.index(self) + 1 :]
177             dif = n - len(cells)
178             if dif > 0 and cells[-1].connection is not None:
179                 cells += cells[-1].get_front_cells(dif, route)
180             return cells[:n]
181
182
183     class Car:
184         id = 0
185         cell = None
186         speed = 0
187         route = None
188         v_max = 3
189         p = None
190
191         decelerate_rate = 0.3
192         # Que tan agresivo es el carro para cambiar de carriles
193         base_lane_changing_rate = 0.2
194         # Que tan agresivo es en este momento el carro para cambiar de
                carril
195         lane_changing_rate = 0.2
196         # Que tan probable es que cambie a la derecha. Complemento a la
                izquierda
197         right_change_rate = 0.5
198         # Cuantas iteraciones esperaria al final de una calle para cambiar
```

```
        de carril
199     # antes de cambiar de ruta
200     changing_route_max_wait = 10
201     # Cuantas iteraciones lleva esperando al final de una calle
            tratando de cambiar de carril
202     waits_for_lane_change = 0
203
204     def __init__(self):
205         self.id = Car.id
206         Car.id += 1
207
208     def __repr__(self):
209         return "<Car: %s (%s)>" % (self.id, self.speed)
210
211     def apply_rules(self):
212         if isinstance(self.p.cell, StreetCell):
213             if not isinstance(self.cell, StreetCell) or (
214                 self.cell.street != self.p.cell.street
215             ):
216                     self.p.cell.street.car_entry(self)
217         self.speed = self.p.speed
218         self.cell = self.p.cell
219
220
221 class EntranceRule(Rule):
222     generate = False
223     is_street = False
224
225     def populate(self):
226         self.generate = False
227         if self.cell.connection is not None:
228             return
229         if self.car is not None:
230             return
231         if random.random() <= self.cell.rate:
232             self.generate = True
```

```
233             if isinstance(self.cell, StreetCell):
234                 self.is_street = True
235 
236         def apply_(self):
237             if self.generate:
238                 car = Car()
239                 car.cell = self.cell
240                 car.speed = self.cell.speed
241                 self.cell.car = car
242                 if self.is_street:
243                     self.cell.street.car_entry(car)
244                 self.cell.topology.cars.append(car)
245 
246     class ExitRule(Rule):
247         consume = False
248 
249         def populate(self):
250             self.consume = False
251             if self.cell.connection is not None:
252                 return
253             if self.car is None:
254                 return
255             if random.random() <= self.cell.rate:
256                 self.consume = True
257 
258         def apply_(self):
259             if self.consume:
260                 self.cell.topology.cars.remove(self.car)
261                 self.cell.car = None
262                 self.car.cell = None
263 
264     class EndpointCell(Cell):
265         rate = 0
266         connection = None
267 
268     class EndpointEntranceCell(EndpointCell):
```

```
269     endpoint_rule_class = EntranceRule
270     speed = 1
271
272 class EndpointExitCell(EndpointCell):
273     endpoint_rule_class = ExitRule
274
275     def get_front_cells(self, n, route=None):
276         if self.connection is None:
277             return []
278         elif isinstance(self.connection, StreetCell):
279             return [self.connection] +
                self.connection.get_front_cells(n - 1, route)
280         elif isinstance(self.connection, IntersectionCell):
281             if (route in
                self.connection.intersection.semaphore.get_active_light().routes
282                 and route in self.connection.routes):
283                 return [self.connection] +
                    self.connection.get_front_cells(n - 1, route)
284             else:
285                 return []
286         else:
287             raise NotImplementedError
288
289
290 class StreetEntranceCell(EndpointEntranceCell, StreetCell):
291     pass
292
293 class StreetExitCell(EndpointExitCell, StreetCell):
294     pass
295
296 class IntersectionEntranceCell(EndpointEntranceCell, IntersectionCell):
297     pass
298
299 class IntersectionExitCell(EndpointExitCell, IntersectionCell):
300     pass
301
```

```
302 class Automaton:
303     topology = None
304     generation = 0
305     cycle = 40
306 
307     def get_cycle_time(self):
308         return self.generation % self.cycle
309 
310     def sync_update(self, func):
311         # Instantiation & population of rules
312         for cell in self.topology.cells:
313             cell.rule = cell.rule_class(cell)
314 
315         # Calculation of new values
316         for cell in self.topology.cells:
317             func(cell)
318 
319         # Syncronous application of new values
320         for cell in self.topology.cells:
321             cell.rule.apply_()
322 
323     def update(self):
324         # Generation and consumption of cars
325         for cell in self.topology.endpoint_cells:
326             rule = cell.endpoint_rule_class(cell)
327             rule.apply_()
328 
329         # Configuration changes on map
330         self.sync_update(lambda x: x.rule.pre_setting())
331 
332         # Cars movement rules application
333         self.sync_update(lambda x: x.rule.calculate())
334 
335         # Update of semaphores states
336         for semaphore in self.topology.semaphores:
337             semaphore.update()
```

```
338
339             self.generation += 1
340
341
342     class Street:
343         id = 0
344         cells = []
345         exit_routes = []
346         length = None
347         lanes = None
348
349         def __init__(self):
350             self.id = Street.id
351             Street.id += 1
352             self.cells = []
353             self.exit_routes = []
354
355         def __repr__(self):
356             return "<Street: %s>" % self.id
357
358         def car_entry(self, car):
359             if len(self.exit_routes) > 0:
360                 car.route = random.choice(self.exit_routes)
361             else:
362                 car.route = None
363
364
365     class Route:
366         id = 0
367         cells = []
368         entrance_lane = None
369
370         def __init__(self):
371             self.id = Route.id
372             Route.id += 1
373             self.cells = []
```

```
374
375     def __repr__(self):
376         return "<Route: %s>" % (self.id)
377
378
379 class Intersection:
380     id = 0
381     cells = []
382     routes = []
383
384     in_streets = []
385     out_streets = []
386     neighbors = []
387
388     semaphore = None
389
390     def __init__(self):
391         self.id = Intersection.id
392         Intersection.id += 1
393         self.cells = []
394         self.routes = []
395
396         self.in_streets = []
397         self.out_streets = []
398         self.neighbors = []
399
400     def __repr__(self):
401         return "<Intersection: %s>" % (self.id)
402
403     def get_valid_route(self, cell):
404         for route in self.routes:
405             if cell in route.cells:
406                 return route
407         raise KeyError
408
409
```

```
410 class Semaphore:
411     id = 0
412     lights = []
413     counter = 0
414     active = 0
415     schedule = None
416     topology = None
417 
418     def __init__(self):
419         self.id = Semaphore.id
420         Semaphore.id += 1
421         self.lights = []
422         self.schedule = {}
423 
424     def __repr__(self):
425         return "<Semaphore: %s>" % (self.id)
426 
427     def set_schedule(self, schedule):
428         prev = None
429         self.active = 0
430         cycle_time = 0
431         if len(self.schedule) > 0:
432             self.get_active_light().free = False
433             cycle_time = self.topology.automaton.get_cycle_time()
434         self.schedule = {}
435         for period_start in sorted(schedule):
436             light = schedule[period_start]
437             self.schedule[period_start] = {
438                 'light': light,
439                 'change': 0,
440             }
441             if period_start <= cycle_time:
442                 self.active = period_start
443             if prev is not None:
444                 self.schedule[prev]['change'] = period_start
445             prev = period_start
```

```
446             self.get_active_light().free = True
447
448
449         def get_schedule(self):
450             schedule = dict()
451             for k, v in self.schedule.items():
452                 schedule[k] = v['light']
453             return schedule
454
455         def update(self):
456             time = self.topology.automaton.get_cycle_time()
457             change = self.schedule[self.active]['change']
458             change = change if change != 0 else
                    self.topology.automaton.cycle
459             if time == 0:
460                 self.get_active_light().free = False
461                 self.active = 0
462                 self.get_active_light().free = True
463             elif time >= change:
464                 self.get_active_light().free = False
465                 self.active = self.schedule[self.active]['change']
466                 self.get_active_light().free = True
467
468         def get_active_light(self):
469             return self.schedule[self.active]['light']
470
471
472 class Light:
473     id = 0
474     routes = []
475     viewer_address = None
476     semaphore = None
477     free = False
478
479     def __init__(self):
480         self.id = Light.id
```

```
481             Light.id += 1
482             self.routes = []
483 
484         def __repr__(self):
485             return "<Light: %s (%s)>" % (self.id, 1 if self.free else 0)
486 
487 
488 class Topology:
489     cells = []
490     endpoint_cells = []
491     lights = []
492     semaphores = []
493 
494     intersections = []
495     streets = []
496 
497     cars = []
498     automaton = None
499 
500     def __init__(self):
501         self.cells = []
502         self.endpoint_cells = []
503         self.lights = []
504         self.semaphores = []
505         self.cars = []
```

B CÓDIGO FUENTE DE ALGORITMO GENÉTICO

```
1 '''
2 Genetic Algorithm for the optimization of traffic lights cycles
3 Jorge Lainfiesta 11142
4 '''
5 
6 import random
7 import copy
8 
```

```
9  from deap import base
10 from deap import creator
11 from deap import tools
12
13 from service.tca_service import TCAService
14 from operator import itemgetter
15
16 def get_normalized_lights(traffic_lights):
17     '''
18     Recieves an array of intersection dicts
19     Returns a dict with the form {id: [normalized ids]}
20     '''
21     #Get the max amount of lights on any intersection
22     max_lights = max(len(_['lights']) for _ in traffic_lights)
23     #Build a normalized list of IDs for each light
24     intersections = {}
25     real_intersections = {}
26     for intersection in traffic_lights:
27         intersections[intersection['id']] = [l % max_lights for l in
               intersection['lights']]
28         real_intersections[intersection['id']] = [l for l in
               intersection['lights']]
29     return intersections, real_intersections
30
31 def build_rand_chromosome(individual, intersections, period, getrand):
32     '''
33     Receives a normalized intersection lights dict, a period and a
       random choice function
34     Returns an array with a chromosome form
35     '''
36     chromosome = []
37     for id, lights in intersections.items():
38         for _ in range(period):
39             chromosome.append(getrand(lights))
40     return individual(chromosome)
41
```

```
42 def decode_chromosome(period, normal_inters, real_inters, individual):
43     '''
44     Receives a period, the normalized intersections, the real
       intersections, and an individual
45     Returns a list of intersection descriptions according to the API
46     '''
47     api_inters = []
48     for inter_id, inter_lights in normal_inters.items():
49         api_inter = {"id": inter_id, "lights": real_inters[inter_id]}
50         light_past = -1
51         schedule = {}
52         for t in range(period):
53             light_t = individual[t]
54             if light_t != light_past:
55                 schedule[t] =
                       real_inters[inter_id][inter_lights.index(light_t)]
56                 light_past = light_t
57         api_inter['schedule'] = schedule
58         api_inters.append(api_inter)
59     return api_inters
60
61 def evaluate(simulator, period, normal_inters, real_inters,
       individual):
62     '''
63     Receives a simulator instance, an intersection map and an
       individual
64     Executes simulation and returns fitness values
65     '''
66     #Map normalized ids to real ids and calculate times
67     api_inters = decode_chromosome(period, normal_inters, real_inters,
           individual)
68     simulator.reset_statistics()
69     #Change traffic lights and configuration and run simulation
70     simulator.set_traffic_lights(api_inters)
71     simulator.fixed_time_start(period * 5)
72
```

```
73      return simulator.get_average_speed(), simulator.get_stopped_time()
74
75  def fill_toolbox(intersections, period, simulator):
76      '''
77  ____Returns a DEAP toolbox with required components
78  ____'''
79      normal_inters, real_inters = get_normalized_lights(intersections)
80      #Register toolbox components
81      toolbox = base.Toolbox()
82      #Population building components
83      toolbox.register("attr_light", random.choice)
84      toolbox.register("individual", build_rand_chromosome,
            creator.Individual, normal_inters, period, toolbox.attr_light)
85      toolbox.register("population", tools.initRepeat, list,
            toolbox.individual)
86      #Register operator
87      toolbox.register("selectBest", tools.selBest)
88      toolbox.register("selectRest", tools.selRoulette)
89      toolbox.register("mate", tools.cxUniform)
90      toolbox.register("mutate", tools.mutShuffleIndexes)
91      toolbox.register("decode", decode_chromosome, period,
            normal_inters, real_inters)
92      toolbox.register("evaluate", evaluate, simulator, period,
            normal_inters, real_inters)
93      #Register helper
94      toolbox.register("clone", copy.copy)
95
96      return toolbox
97
98  def find_solution(population=100, max_gen=10, period=10, seed=64):
99      '''
100 ____Recives configurations for the genetic algorithm: period, seed
101 ____Executes algorithm to find a result
102 ____Returns found solution
103 ____'''
104     #Get simulator data
```

```
105     simulator = TCAService()
106     intersections = simulator.get_traffic_lights()
107
108     #Register global creator classes
109
110     #Fitrness function should maximize average speed and minimize
            total stopped time
111     creator.create("FitnessMin", base.Fitness, weights=(1.0, -0.5))
112     #Individual basic definition
113     creator.create("Individual", list, fitness=creator.FitnessMin)
114
115     #Set random seed and fill toolbox
116     random.seed = seed
117     toolbox = fill_toolbox(intersections, period, simulator)
118
119     #Init population
120     population = toolbox.population(n=population)
121
122     #Evaluate the entire population fitness
123     fitnesses = list(map(toolbox.evaluate, population))
124     for ind, fit in zip(population, fitnesses):
125         ind.fitness.values = fit
126     #Operator probability
127     CXPB, MUTPB = 0.5, 0.2
128
129     #Iterate generations
130     g = 0
131
132     fitness_records = []
133     while g < max_gen:
134         print("g: %s population: %s" % (g, len(population)))
135         # Select the next generation individuals
136         best_num = int(len(population) * 0.20)
137         offspring = toolbox.selectBest(population, best_num)
138         # Clone the selected individuals
139         offspring = list(map(toolbox.clone, offspring))
```

```
140         offspring = [toolbox.clone(child) for child in offspring]
141         # Select the next generation individuals
142         offspring2 = toolbox.selectRest(population, (len(population) -
                best_num))
143         # Clone the selected individuals
144         offspring2 = list(map(toolbox.clone, offspring2))
145
146         # Apply crossover on the offspring
147         for child1, child2 in zip(offspring2[::2], offspring2[1::2]):
148             if random.random() < CXPB:
149                 toolbox.mate(child1, child2, 0.2)
150                 del child1.fitness.values
151                 del child2.fitness.values
152
153         # Apply mutation on the offspring
154         for mutant in offspring2:
155             if random.random() < MUTPB:
156                 toolbox.mutate(mutant, 0.1)
157                 del mutant.fitness.values
158
159         # Evaluate the individuals with an invalid fitness
160         invalid_ind = [ind for ind in offspring2 if not
                ind.fitness.valid]
161         fitnesses = list(map(toolbox.evaluate, invalid_ind))
162         for ind, fit in zip(invalid_ind, fitnesses):
163             ind.fitness.values = fit
164
165         offspring.extend(offspring2)
166
167         all_fitness = [ind.fitness.values for ind in offspring]
168
169         #Calculate fitness statistics
170         avg_fitness_speed = sum(f[0] for f in all_fitness) /
                len(all_fitness)
171         max_fitness_speed = max(all_fitness, key=itemgetter(0))[0]
172         avg_fitness_stop = sum(f[1] for f in all_fitness) /
```

```
            len(all_fitness)
173         max_fitness_stop = min(all_fitness, key=itemgetter(1))[1]
174
175         fitness_records.append((avg_fitness_speed, max_fitness_speed,
                avg_fitness_stop, max_fitness_stop))
176
177
178         # The population is entirely replaced by the offspring
179         population[:]= offspring
180         g += 1
181     #Select the best one
182     best = toolbox.selectBest(population, 1)[0]
183     print(toolbox.decode(best))
184     print(best.fitness.values)
185     f = open(('roulete-fit-%s-%s-%s.csv' %(len(population), max_gen,
            period)), 'w')
186     for record in fitness_records:
187         f.write(str(record)[1:-1] + "\n")
188     f.close()
189
190 if __name__ == '__main__':
191     #Execution example using cProfiler
192     import cProfile
193     cProfile.run('find_solution(population=%s, max_gen=%s, period=%s)'
            %(200, 100, 30))
```

C CÓDIGO FUENTE DE LOS TIPOS DE AGENTE

```
1 import random
2 import math
3 from service.tca_service import TCAService
4
5 class ITA(object):
6   def __init__(self,service,cycle=5):
7     self.service = service
8     self.dynamic = None
```

```
9       self.newSchedule = None
10      self.timeInterval = cycle
11      self.intersections = {}
12      for i in self.service.get_intersections():
13        self.intersections[i['id']] = i
14      traffic_lights = self.service.get_traffic_lights()
15      for key, value in self.intersections.items():
16        for j in traffic_lights:
17          if key==j['id']:
18            self.intersections[key]['schedule'] = j['schedule']
19            self.intersections[key]['green_light_id'] = j['schedule'][0]
20            self.intersections[key]['lights'] = j['lights']
21
22    def update(self):
23      self.dynamic = {}
24      self.newSchedule = []
25      for avg in self.service.dynamic_time_update(self.timeInterval):
26        street = {}
27        street['avg_speed'] = avg['average_speed']
28        street['cars_number'] = avg['cars_number']
29        street['green_light'] = avg['green_light']
30        self.dynamic[avg['id']] = street
31
32    def updateSchedule(self):
33      self.service.set_traffic_lights(self.newSchedule)
34
35    def getSpeed(self,agentID):
36      intersection = self.intersections[agentID]
37      speed = {}
38      for i in intersection['in_streets']:
39        speed[i] = self.dynamic[i]
40      return speed
41
42    def schedule(self,agentID):
43      intersection = self.intersections[agentID]
44      schedule = {}
```

```
45      for i in intersection['lights']:
46        if(i != self.intersections[agentID]['green_light_id']):
47          self.intersections[agentID]['green_light_id'] = i
48          schedule[0] = self.intersections[agentID]['green_light_id']
49          break
50      self.newSchedule.append({'id':agentID,'schedule':schedule})
51      return 1
52
53    def neighbours(self,intersection):
54      return self.intersections[intersection]['out_streets']
55
56    def getTimeInterval(self):
57      return self.timeInterval
58
59    def total_cars(self):
60      total = 0
61      for key, value in self.dynamic.items():
62        total+=value['cars_number']
63      return total
64
65    def getLightStreet(self,agentID):
66      intersection = self.intersections[agentID]
67      lights = {}
68      for i in intersection['in_streets']:
69        if self.dynamic[i]['green_light'] == 0:
70          lights['red'] = i
71        else:
72          lights['green'] = i
73      return lights
```

```
1  import random
2  import math
3  from service.tca_service import TCAService
4
5  class CTA(object):
6    def __init__(self,ITA):
```

```
7       self.ITA = ITA
8
9     #Comapare with neighbours if there is a congestion to stop vehicles
          from keep going
10    def validate(self,agentID,redStreet):
11      neighbours = self.ITA.neighbours(agentID)
12      density = []
13      cars = []
14      for i in neighbours:
15        density.append(self.ITA.dynamic[i]['cars_number']/(self.ITA.dynamic[i]['avg_spe
16        cars.append(self.ITA.dynamic[i]['cars_number'])
17
18      # The value of 100 is arbitrary. Can be changed find better results
19      if max(density)<100:
20        return 1
21      else:
22        if max(cars)<self.ITA.dynamic[redStreet]['cars_number']:
23          return 1
24        else:
25          return 0
```

```
1  import random
2  import math
3  from service.tca_service import TCAService
4  from tca_ma.Sarsa import Sarsa
5
6  class LTA(object):
7    def __init__(self,id,ITA,CTA,tilings=10):
8        self.id = id
9        self.actions=["change","stay"]
10       # epsilon=0.1, alpha=0.1, gamma=0.9
11       self.algorithm = Sarsa(self.actions,0.1, 0.1,0.9,tilings)
12       self.lastAction = None
13       self.ITA = ITA
14       self.CTA = CTA
15       self.streetLights = None
```

```
16          self.greenSpeed = 0.0
17          self.redSpeed = 0.0
18          self.ratio = 1.0
19          self.lastChange = 0
20          self.secondLast = 0
21
22      def update(self):
23          self.lastChange+= self.ITA.getTimeInterval()
24          self.secondLast+= self.ITA.getTimeInterval()
25          speed = self.ITA.getSpeed(self.id)
26          self.streetLights = self.ITA.getLightStreet(self.id)
27          self.greenSpeed = speed[self.streetLights['green']]['avg_speed']
                #green
28          self.redSpeed = speed[self.streetLights['red']]['avg_speed'] #red
29          #Reward = negative of the sum of time since last two changes and
                the number of vehicles
30          reward = -(speed[self.streetLights['red']]['cars_number'] +
                self.lastChange)
31          if self.greenSpeed> self.redSpeed:
32            if(self.redSpeed == 0):
33              self.ratio  = 5
34            else:
35              self.ratio = math.log(self.greenSpeed/self.redSpeed)
36          else:
37            if(self.greenSpeed == 0):
38              self.ratio  = -5
39            else:
40              self.ratio = -math.log(self.redSpeed/self.greenSpeed)
41          state =
                self.algorithm.activeTile([self.lastChange,self.secondLast,self.ratio])
42          action = self.algorithm.chooseAction(state)
43
44          if action=="change":
45            #CTA will decide if it's better not changing the lights
46            if self.CTA.validate(self.id,self.streetLights['red']) == 1:
47              self.secondLast = self.lastChange
```

```
48              self.lastChange = 0
49              temp = self.streetLights['green']
50              self.streetLights['green'] = self.streetLights['red']
51              self.streetLights['red'] = temp
52              self.ITA.schedule(self.id)
53
54          if self.lastAction is not None:
55            self.algorithm.learn(self.lastState, self.lastAction, reward,
                  state, action)
56          self.lastState = state
57          self.lastAction = action
```

D CÓDIGO FUENTE DEL ALGORITMO SARSA

```
1  import random
2  import math
3
4  class Sarsa(object):
5      def __init__(self, actions, epsilon=0.1, alpha=0.2,
           gamma=0.9,tilings=1):
6          self.q = {"change":{},"stay":{}}
7          self.epsilon = epsilon
8          self.alpha = alpha
9          self.gamma = gamma
10         self.actions = actions
11
12         """
13 ␣␣␣␣␣␣␣␣specifications␣used␣for␣the␣experiments:
14 ␣␣␣␣␣␣␣␣-negative:␣if␣the␣variable␣can␣have␣negative␣values
15 ␣␣␣␣␣␣␣␣-start:␣first␣value␣of␣the␣variable␣in␣the␣tiling
16 ␣␣␣␣␣␣␣␣-width:␣width␣of␣the␣tiles
17 ␣␣␣␣␣␣␣␣-size:␣number␣of␣tiles
18 ␣␣␣␣␣␣␣␣-res:␣resolution␣of␣the␣tiling
19 ␣␣␣␣␣␣␣␣"""
20         self.specs = [
21           {"negative":False,"start":0,"width":5.0,"size":30,"res":
```

```
            0.5},
22          {"negative":False,"start":0,"width":10.0,"size":30,"res":
                1.0},
23          {"negative":True,"start":-5,"width":0.5,"size":30,"res":
                0.05}
24      ]
25
26  def getQvalue(self, state, action,):
27      return self.q[action].get(state, 0.0)
28
29  def Qlearning(self, state, action, reward, value):
30      oldv = self.q[action].get(state, None)
31      if oldv is None:
32          self.q[action][state] = reward
33      else:
34          self.q[action][state] = oldv + self.alpha * (value - oldv)
35
36  def chooseAction(self, state):
37      if random.random() < self.epsilon:
38          action = random.choice(self.actions)
39      else:
40          q = []
41          for a in self.actions:
42            q.append(self.getQvalue(state,a))
43          maxQ = max(q)
44          count = q.count(maxQ)
45          if count > 1:
46              best = [i for i in range(len(self.actions)) if q[i] ==
                    maxQ]
47              i = random.choice(best)
48          else:
49              i = q.index(maxQ)
50
51          action = self.actions[i]
52      return action
53
```

```
54      def learn(self, state1, action1, reward, state2, action2):
55          qnext = self.getQvalue(state2, action2)
56          self.Qlearning(state1, action1, reward, reward + self.gamma *
                qnext)
57
58      #Discretize state
59      def activeTile(self,state):
60        tiles = []
61        coord = []
62        dim = 0
63        for i in self.specs:
64          length = i["width"]*i["size"]
65          cell = (state[dim]-i["start"])/i["width"]
66
67          coord.append(str(math.floor(cell)))
68        coord = ",".join(coord)
69        return coord
70
71      #Active tiles for tile coding
72      def activeTiles(self,state,tilings=1):
73        tiles = []
74        for tiling in range(tilings):
75          coord = []
76          dim = 0
77          for i in self.specs:
78            length = i["width"]*i["size"]
79            cell = length/state[dim]
80            if i["negative"]:
81              cell = cell + i["res"]*tiling if tiling<tilings/2 else
                    cell - i["res"]*tiling
82            else:
83              cell = cell - i["res"]*tiling
84            coord.append(math.floor(cell))
85            dim+=1
86          coord = ",".join(coord)
87          tiles.append(coord)
```

```
88          return tiles
89
90      #Tile coding
91      def weightSum(self,tiles):
92          totalWeight = 0
93          for key in tiles:
94              if self.matrix.has_key(key):
95                  totalWeight+=self.matrix[key]
96              else:
97                  self.matrix[key]=0
98          return totalWeight
```

E CÓDIGO FUENTE DE LA API

```
1
2  from tca_ng.models import Automaton, Semaphore, Light
3  from tca_ng.example_maps import totito_map, grid_2lane_map
4  import random
5
6
7  class TCAService(object):
8      """
9  ____Traffic Cellular Automaton Service - Application Program Interface
       for TCA simulator
10
11 ____Strategies:
12 ________-Fixed traffic lights time
13 ________-Dynamic traffic lights  time
14
15 ____Metrics:
16 ________-Average cars speed
17 ________-Total stopped time
18 ____"""
19
20      def __init__(self, map=1, rate=0.2, seed=0):
21          """
```

```
22 ␣␣␣␣␣␣␣␣TCAService␣␣␣init__
23 ␣␣␣␣␣␣␣␣:return:
24 ␣␣␣␣␣␣␣␣"""
25             # Automaton
26             self._automaton = Automaton()
27 
28             Semaphore.id = 0
29             Light.id = 0
30 
31             if map == 1:
32                 self._automaton.topology = totito_map(10, rate)
33             elif map == 2:
34                 self._automaton.topology = grid_2lane_map(5, 2, 1, rate)
35             elif map == 3:
36                 self._automaton.topology = grid_2lane_map(5, 2, 2, rate)
37             self._automaton.topology.automaton = self._automaton
38 
39             self.map = map
40             self.rate = rate
41             self.seed = seed
42 
43             # Class attributes
44             self.traffic_lights = []
45             self.average_speed = None
46             self.step_average_speed = []
47             # self.average_distance = 0
48             self.stopped_time = 0
49             self.step_stopped_time = []
50             self.average_stopped_time = 0
51             # self.stopped_time_per_car
52             self.average_cars_number = None
53             self.step_car_number = []
54             self._cycle_count = 0
55             self.intersections = []
56             self.streets = []
57             self.iteration = 0
```

```
58
59          self.printable_statistics = []
60
61          # Build data from map
62          self._build_traffic_lights()
63          self._build_intersections()
64
65      def fixed_time_start(self, cycle_count=60, all_data=False):
66          """
67  ␣␣␣␣␣␣␣␣Simulation␣start␣using␣a␣fixed␣time␣strategy
68  ␣␣␣␣␣␣␣␣:param␣cycle_count:␣Number␣of␣cycles␣to␣be␣simulated
69  ␣␣␣␣␣␣␣␣:param␣all_data:␣If␣True␣print␣data␣to␣file
70  ␣␣␣␣␣␣␣␣:return:␣True␣if␣simulation␣was␣successfully␣completed
71  ␣␣␣␣␣␣␣␣"""
72
73          # Set simulation values
74          self._cycle_count = cycle_count
75
76          try:
77
78              # Simulate and get data from TCA simulator
79              for i in range(cycle_count):
80                  self._automaton.update()
81                  self.iteration += 1
82                  self._update_data()
83
84              # Process obtained data
85              self._process_data()
86
87              # Print obtained data to file
88              if all_data:
89                  self._print_data()
90
91          except Exception as e:
92              print('\nERROR:␣Simulator␣raised␣an␣exception!')
93              print('Exception␣message:␣{}␣\n'.format(e))
```

```
94              return False
95
96          # Success
97          return True
98
99      def dynamic_time_update(self, cycle_count=5, all_data=True):
100         """
101 ␣␣␣␣␣␣␣␣Simulation␣update␣using␣dynamic␣time␣strategy
102 ␣␣␣␣␣␣␣␣:param␣cycle_count:␣Number␣of␣updates␣to␣be␣simulated
103 ␣␣␣␣␣␣␣␣:param␣all_data:␣If␣True␣print␣data␣to␣file
104 ␣␣␣␣␣␣␣␣:return:␣List␣containing␣streets␣information␣in␣this␣format
105
106 ␣␣␣␣␣␣␣␣␣␣␣␣[
107 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣{'id':␣0,␣'cars_number':␣9,␣'average_speed':␣2.5,␣
        'green_light':␣0},
108 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣{'id':␣1,␣'cars_number':␣11,␣'average_speed':␣3.1,␣
        'green_light':␣1},
109 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣{'id':␣2,␣'cars_number':␣9,␣'average_speed':␣2.3,␣
        'green_light':␣None}
110 ␣␣␣␣␣␣␣␣␣␣␣␣]
111
112 ␣␣␣␣␣␣␣␣"""
113
114         try:
115
116             # Simulate and get data from TCA simulator
117             for i in range(cycle_count):
118                 self._automaton.update()
119                 self.iteration += 1
120                 self._update_data()
121
122                 # Print obtained data to file
123                 if all_data:
124                     self._print_data()
125
126             # Process obtained data
```

```
127             self._process_data()
128
129         except Exception as e:
130             print('\nERROR: Simulator raised an exception!')
131             print('Exception message: {} \n'.format(e))
132             return None
133
134         # Success, build and return data
135         self._build_streets()
136         return self.streets
137
138     def random_fixed_time_start(self, cycle_count=60, all_data=False):
139
140         # Set simulation values
141         self._cycle_count = cycle_count
142
143         # Generate random schedule and set it
144         schedule = []
145
146         for traffic_light in self.get_traffic_lights():
147             schedule_dict = dict()
148             schedule_dict['id'] = traffic_light['id']
149
150             light_dict = dict()
151             light_dict[0] = random.choice(traffic_light['lights'])
152             for light in traffic_light['lights']:
153
154                 start = random.randint(0, self.get_cycle_size())
155
156                 light_dict[start] = light
157
158             schedule_dict['schedule'] = light_dict
159
160             schedule.append(schedule_dict)
161
162         self.set_traffic_lights(schedule)
```

```
163
164         try:
165
166             # Simulate and get data from TCA simulator
167             for i in range(cycle_count):
168                 self._automaton.update()
169                 self.iteration += 1
170                 self._update_data()
171
172             # Process obtained data
173             self._process_data()
174
175             # Print obtained data to file
176             if all_data:
177                 self._print_data()
178
179         except Exception as e:
180             print('\nERROR: Simulator raised an exception!')
181             print('Exception message: {} \n'.format(e))
182             return False
183
184         # Success
185         return True
186
187     def random_dynamic_time_start(self, cycle_count=60,
        variation_time=5, rate=0.8, all_data=False):
188
189         # Set simulation values
190         self._cycle_count = cycle_count
191
192         try:
193
194             # Simulate and get data from TCA simulator
195             for i in range(cycle_count):
196                 self._automaton.update()
197                 self.iteration += 1
```

```
198                 self._update_data()
199
200                 # Verify if a change is needed
201                 if random.random() < rate:
202
203                     # Generate random schedule and set it
204                     schedule = []
205
206                     for traffic_light in self.get_traffic_lights():
207
208                         if random.random() < rate:
209                             schedule_dict = dict()
210                             schedule_dict['id'] = traffic_light['id']
211
212                             light_dict = dict()
213                             light_dict[0] =
                                    random.choice(traffic_light['lights'])
214                             for light in traffic_light['lights']:
215
216                                 delta =
                                        random.randint(-variation_time,
                                        variation_time)
217                                 start = 0
218
219                                 for k, v in
                                        traffic_light['schedule'].items():
220                                     if v == traffic_light:
221                                         start = k
222
223                                 start += delta
224                                 start = min(start,
                                        self.get_cycle_size())
225                                 start = max(start, 0)
226
227                                 light_dict[start] = light
228
```

```
229                                 schedule_dict['schedule'] = light_dict
230
231                                 schedule.append(schedule_dict)
232
233                         self.set_traffic_lights(schedule)
234
235                 # Process obtained data
236                 self._process_data()
237
238                 # Print obtained data to file
239                 if all_data:
240                     self._print_data()
241
242             except NotImplementedError:
243                 pass
244             # except Exception as e:
245             #     print('\nERROR: Simulator raised an exception!')
246             #     print('Exception message: {} \n'.format(e))
247             #     return False
248
249             # Success
250             return True
251
252         def reset_statistics(self):
253             """
254         ________Reset_statistics_to_zero_or_empty
255         ________:return:
256         ________"""
257             Semaphore.id = 0
258             Light.id = 0
259
260             self._automaton = Automaton()
261             if self.map == 1:
262                 self._automaton.topology = totito_map(10, self.rate)
263             elif self.map == 2:
264                 self._automaton.topology = grid_2lane_map(5, 2, 2,
```

```
                self.rate)
265         elif self.map == 3:
266             self._automaton.topology = grid_2lane_map(5, 2, 2,
                    self.rate)
267         self._automaton.topology.automaton = self._automaton
268 
269         self.average_speed = 0
270         self.step_average_speed = []
271         self.stopped_time = 0
272         self.step_stopped_time = []
273         self.average_stopped_time = 0
274         self.average_cars_number = 0
275         self.step_car_number = []
276         self._cycle_count = 0
277         self.iteration = 0
278 
279         # Build data from map
280         self._build_traffic_lights()
281         self._build_intersections()
282 
283     def get_actual_iteration(self):
284         """
285 ␣␣␣␣␣␣␣␣Get␣actual␣iteration␣number
286 ␣␣␣␣␣␣␣␣:return:␣iteration␣number
287 ␣␣␣␣␣␣␣␣"""
288 
289         return self.iteration
290 
291     def get_max_speed(self):
292         """
293 ␣␣␣␣␣␣␣␣Get␣max␣speed␣car␣speed␣in␣simulator
294 ␣␣␣␣␣␣␣␣:return:␣max␣speed
295 ␣␣␣␣␣␣␣␣"""
296 
297         # TODO get real value
298         return 3
```

```
299
300     def get_cycle_size(self):
301         """
302 ␣␣␣␣␣␣␣␣Get␣automaton␣cycle␣size
303 ␣␣␣␣␣␣␣␣:return:␣cycle␣size
304 ␣␣␣␣␣␣␣␣"""
305
306         return self._automaton.cycle
307
308     def set_cycle_size(self, cycle_size):
309         """
310 ␣␣␣␣␣␣␣␣Set␣automaton␣cycle␣size
311 ␣␣␣␣␣␣␣␣:param␣cycle_size:␣Cycle␣size
312 ␣␣␣␣␣␣␣␣:return:␣True␣if␣changed␣correctly
313 ␣␣␣␣␣␣␣␣"""
314
315         try:
316             self._automaton.cycle = cycle_size
317         except Exception as e:
318             print('\nERROR:␣Cycle␣size␣could␣not␣be␣changed!')
319             print('Exception␣message:␣{}␣\n'.format(e))
320             return False
321
322         return True
323
324     def get_intersections(self):
325         """
326 ␣␣␣␣␣␣␣␣Get␣intersections␣in␣this␣format:
327
328 ␣␣␣␣␣␣␣␣␣␣␣␣[
329 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣{'id':␣0,␣'traffic_light':␣0,␣'in_streets':␣[0,␣1],␣
    'out_streets':␣[2,␣3]},
330 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣{'id':␣1,␣'traffic_light':␣1,␣'in_streets':␣[3,␣4],␣
    'out_streets':␣[5,␣6]},
331 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣{'id':␣2,␣'traffic_light':␣2,␣'in_streets':␣[6,␣2],␣
    'out_streets':␣[0,␣4]}
```

```
332             ]
333
334         :return: List containing dictionaries representing
                intersections
335         """
336
337             return self.intersections
338
339         def get_traffic_lights(self):
340             """
341         Get traffic lights in this format:
342
343             [
344                 {'id': 0, 'schedule': {0: 0, 5: 1}, 'lights': [0, 1]},
345                 {'id': 1, 'schedule': {0: 2, 4: 3}, 'lights': [2, 3]},
346                 {'id': 2, 'schedule': {2: 4, 6: 5}, 'lights': [4, 5]}
347             ]
348
349         :return: List containing dictionaries representing traffic
                lights
350         """
351             # Re build traffic lights
352             self._build_traffic_lights()
353
354             # Return dictionary containing traffic lights information
355             return self.traffic_lights
356
357         def set_traffic_lights(self, traffic_light_schedule):
358             """
359         Set traffic lights schedule:
360
361             [
362                 {'id': 0, 'schedule': {0: 0, 5: 1}},
363                 {'id': 1, 'schedule': {0: 2, 4: 3}},
364                 {'id': 2, 'schedule': {2: 4, 6: 5}}
365             ]
```

```
366 ␣␣␣␣␣␣␣␣:param␣traffic_light_schedule:␣Dictionary␣containing␣traffic␣
        lights␣schedule
367 ␣␣␣␣␣␣␣␣:return:␣True␣if␣traffic␣lights␣schedule␣changed␣correctly
368 ␣␣␣␣␣␣␣␣"""
369             try:
370                 for schedule in traffic_light_schedule:
371                     traffic_light =
                            self._search_traffic_light(schedule['id'])
372 
373                     # Build and set new schedule dictionary to change
                        format
374                     new_schedule = dict()
375                     for offset, light in schedule['schedule'].items():
376                         new_schedule[offset] = self._search_light(light)
377 
378                     traffic_light.set_schedule(new_schedule)
379 
380             except InvalidTrafficLightId as invalid_traffic_light_id:
381                 raise invalid_traffic_light_id
382             except InvalidLightId as invalid_light_id:
383                 raise invalid_light_id
384             except Exception as e:
385                 print('\nERROR:␣Incorrect␣dictionary␣for␣setting␣traffic␣
                        lights␣schedule!')
386                 print('Exception␣message:␣{}␣\n'.format(e))
387                 return False
388 
389             # Success
390             return True
391 
392         def get_average_speed(self):
393             """
394 ␣␣␣␣␣␣␣␣Get␣average␣speed␣of␣the␣simulation
395 
396 ␣␣␣␣␣␣␣␣Formula:
397 ␣␣␣␣␣␣␣␣␣␣␣␣(car_1_speed_1␣+␣car_2_speed_1␣+␣car_1_speed_2␣+␣
```

```
        car_2_speed_2) / number_of_cars / number_of_cycles
398
399 ␣␣␣␣␣␣␣␣:return:␣Average␣speed,␣None␣if␣not␣available
400 ␣␣␣␣␣␣␣␣"""
401             return self.average_speed
402
403         def get_average_cars_number(self):
404             """
405 ␣␣␣␣␣␣␣␣Get␣average␣cars␣number
406
407 ␣␣␣␣␣␣␣␣Formula:
408 ␣␣␣␣␣␣␣␣␣␣␣␣(cars_number_iteration_1␣+␣cars_number_iteration_2␣+␣
        cars_number_iteration_number_of_cycles)␣/␣number_of_cycles
409 ␣␣␣␣␣␣␣␣:return:␣Average␣cars␣number,␣None␣if␣not␣available
410 ␣␣␣␣␣␣␣␣"""
411
412             return self.average_cars_number
413
414         def get_average_stopped_time(self):
415             """
416 ␣␣␣␣␣␣␣␣Get␣average␣stopped␣time
417
418 ␣␣␣␣␣␣␣␣Formula:
419 ␣␣␣␣␣␣␣␣␣␣␣␣(stopped_time_iteration_1␣+␣stopped_time_iteration_2␣+␣
        stopped_time_iteration_number_of_cycles)␣/␣number_of_cycles
420 ␣␣␣␣␣␣␣␣:return:␣Average␣stopped␣time,␣None␣if␣not␣available
421 ␣␣␣␣␣␣␣␣"""
422
423             return self.average_stopped_time
424
425         def get_average_distance(self):
426             raise NotImplementedError
427
428         def get_stopped_time(self):
429             """
430 ␣␣␣␣␣␣␣␣Get␣stopped␣time␣of␣cars␣in␣simulation
```

```
431
432         Formula:
433             If car speed == 0, stopped time += 1
434
435         :return: Stopped time
436         """
437         return self.stopped_time
438
439     def _search_traffic_light(self, traffic_light_id):
440         """
441         Search for a traffic light into semaphore list
442         :param id: id of the traffic light
443         :return: traffic light object, None if doesn't exists
444         """
445         for traffic_light in self._automaton.topology.semaphores:
446             if traffic_light.id == traffic_light_id:
447                 return traffic_light
448
449         # Raise exception if id doesn't exists
450         raise InvalidTrafficLightId(traffic_light_id)
451
452     def _search_light(self, light_id):
453         """
454         Search for a light into light list
455         :param light_id: id of the light
456         :return: light object, None if doesn't exists
457         """
458
459         for light in self._automaton.topology.lights:
460             if light.id == light_id:
461                 return light
462
463         # Raise exception if id doesn't exists
464         raise InvalidLightId(light_id)
465
466     def _update_data(self):
```

```
467             """
468         Get data from simulator for metrics
469         :return:
470         """
471
472             # Stopped time counter
473             cycle_stopped_time = 0
474
475             # Update average speed (cumulative speed of all cars / number
                    of cars)
476             # Update stopped time, increment 1 when speed equals 0
477             cumulative_speed = 0
478             for car in self._automaton.topology.cars:
479                 if car.speed == 0:
480                     self.stopped_time += 1
481                     cycle_stopped_time += 1
482                 cumulative_speed += car.speed
483
484             if len(self._automaton.topology.cars) > 0:
485                 self.step_average_speed.append(cumulative_speed /
                        len(self._automaton.topology.cars))
486             else:
487                 self.step_average_speed.append(0)
488
489             # Update stopped time
490             self.step_stopped_time.append(cycle_stopped_time)
491
492             # Update cars number
493             self.step_car_number.append(len(self._automaton.topology.cars))
494
495             # Update printable statistics
496             self.printable_statistics.append((self.iteration,
                    self.step_average_speed[-1], self.step_stopped_time[-1],
                    self.step_car_number[-1]))
497
498         def _process_data(self):
```

```
499             """
500         Process final data
501         :return:
502         """
503 
504             # Process average speed
505             self.average_speed = sum(self.step_average_speed) /
                    float(len(self.step_average_speed))
506 
507             # Process average cars number
508             self.average_cars_number = int(round(sum(self.step_car_number)
                    / float(len(self.step_car_number)), 0))
509 
510             # Process average stopped time per simulation
511             self.average_stopped_time = sum(self.step_stopped_time) /
                    float(len(self.step_stopped_time))
512 
513         def _print_data(self):
514 
515             file = open('statistics-{}-{}.csv'.format(self.map,
                    self.rate), 'w')
516 
517             for record in self.printable_statistics:
518                 file.write(str(record)[1:-1] + '\n')
519             file.close()
520 
521         def _build_streets(self):
522             """
523         Build list containing dictionaries representing streets with
            metrics information
524         :return:
525         """
526 
527             # Clean streets list
528             self.streets = []
529             free_routes = []
```

```
530
531         # Get free routes
532         for light in self._automaton.topology.lights:
533
534             # If light is green add all routes
535             if light.free:
536                 for route in light.routes:
537                     free_routes.append(route)
538
539         # Iterate all streets in simulator map
540         for street in self._automaton.topology.streets:
541
542             # Create new dictionary and add street id
543             street_dict = dict()
544             street_dict['id'] = street.id
545
546             # Data values
547             cars_number = 0
548             total_speed = 0
549
550             # Iterate cells in streets and get data
551             for lane in street.cells:
552                 for cell in lane:
553                     if cell.car is not None:
554                         cars_number += 1
555                         total_speed += cell.car.speed
556
557             # Add data to dictionary
558             if cars_number > 0:
559                 street_dict['cars_number'] = cars_number
560                 street_dict['average_speed'] =
                        round(total_speed/cars_number, 2)
561             else:
562                 street_dict['cars_number'] = 0
563                 street_dict['average_speed'] = 0
564
```

```
565             # Verify if light is green
566             green = 1
567 
568             if not street.exit_routes:
569                 green = None
570             else:
571                 for route in street.exit_routes:
572                     if route not in free_routes:
573                         green = 0
574 
575             street_dict['green_light'] = green
576 
577             # Add intersection to intersection list
578             self.streets.append(street_dict)
579 
580     def _build_intersections(self):
581         """
582 ␣␣␣␣␣␣␣␣Build␣list␣containing␣dictionaries␣representing␣intersections
583 ␣␣␣␣␣␣␣␣:return:
584 ␣␣␣␣␣␣␣␣"""
585 
586         # Clean intersection list
587         self.intersections = []
588 
589         # Iterate all intersections in simulator map
590         for intersection in self._automaton.topology.intersections:
591 
592             # Create new dictionary and add intersection id
593             intersection_dict = dict()
594             intersection_dict['id'] = intersection.id
595 
596             # Add traffic light to intersection dictionary
597             intersection_dict['traffic_light'] =
                    intersection.semaphore.id
598 
599             # Build in streets and add it to dictionary
```

```
600             in_streets = []
601             for street in intersection.in_streets:
602                 in_streets.append(street.id)
603             intersection_dict['in_streets'] = in_streets
604 
605             # Build out streets and add it to dictionary
606             out_streets = []
607             for street in intersection.out_streets:
608                 out_streets.append(street.id)
609             intersection_dict['out_streets'] = out_streets
610 
611             # Build neighbors
612             neighbors = []
613             for neighbor in intersection.neighbors:
614                 neighbors.append(neighbor.id)
615             intersection_dict['neighbors'] = neighbors
616 
617             # Add intersection to intersection list
618             self.intersections.append(intersection_dict)
619 
620     def _build_traffic_lights(self):
621         """
622 ________Build list containing dictionaries representing traffic lights
623 ________:return:
624 ________"""
625 
626         # Clean traffic lights list
627         self.traffic_lights = []
628 
629         # Iterate all traffic lights in simulator map
630         for traffic_light in self._automaton.topology.semaphores:
631 
632             # Create new dictionary and add traffic_light id
633             traffic_light_dict = dict()
634             traffic_light_dict['id'] = traffic_light.id
635 
```

```
636             # Build lights list and add it to dictionary
637             lights = []
638             for light in traffic_light.lights:
639                 lights.append(light.id)
640             traffic_light_dict['lights'] = lights
641 
642             # Build schedule and add it to dictionary
643             schedule = dict()
644             for start, value in traffic_light.schedule.items():
645                 schedule[start] = value['light'].id
646             # Add traffic_light schedule
647             traffic_light_dict['schedule'] = schedule
648 
649             # Add traffic_light dictionary to traffic lights list
650             self.traffic_lights.append(traffic_light_dict)
651 
652 
653 class InvalidTrafficLightId(Exception):
654     """
655     Custom exception to raise when a traffic light id is not found.
656     """
657     def __init__(self, value):
658         """
659         Exception __init__
660         :param value: Value that is incorrect
661         :return:
662         """
663         self.value = value
664 
665     def __str__(self):
666         """
667         Exception string representation
668         :return: Message with error representation
669         """
670         return 'ERROR: The traffic light id ({}) does not exists in
                simulator!'.format(repr(self.value))
```

```
671
672
673 class InvalidLightId(Exception):
674     """
675 ␣␣␣␣Custom␣exception␣to␣raise␣when␣a␣light␣id␣is␣not␣found.
676 ␣␣␣␣"""
677
678     def __init__(self, value):
679         """
680 ␣␣␣␣␣␣␣␣Exception␣␣␣init__
681 ␣␣␣␣␣␣␣␣:param␣value:␣Value␣that␣is␣incorrect
682 ␣␣␣␣␣␣␣␣:return:
683 ␣␣␣␣␣␣␣␣"""
684         self.value = value
685
686     def __str__(self):
687         """
688 ␣␣␣␣␣␣␣␣Exception␣string␣representation
689 ␣␣␣␣␣␣␣␣:return:␣Message␣with␣error␣representation
690 ␣␣␣␣␣␣␣␣"""
691         return 'Error:␣The␣light␣id␣({})␣does␣not␣exists␣in␣
            simulator!'.format(repr(self.value))
```

F GLOSARIO

Application Program Interface (API): En programación de computadoras, una interfaz de aplicación de software contiene un conjunto de herramientas que pueden utilizarse para acceder a los servicios que provee una aplicación. Una API define funcionalidades independientes de su implementación por lo que puede cambiar la definición o la implementación sin comprometer la interfaz.

Excepción: En términos de lenguaje de programación, una excepción es el aviso de un problema ocurrido en la ejecución de un programa. Las excepciones suceden regularmente cuando un tipo de dato utilizado no es el correcto o existe alguna instrucción dentro del código que no es posible ejecutar debido a las condiciones del programa en ese instante.

Lenguaje interpretado: Este tipo de lenguajes de programación se caracterizan porque las instruccio-

nes se ejecutan directamente del código sin haber pasado por un proceso de compilación a instrucciones de máquina. Cada sentencia del programa se traduce a un equivalente ya compilado previamente, este proceso se realiza en tiempo real durante la ejecución del programa.