UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de una red de comunicación wifi e interfaz gráfica para una mesa de pruebas de robótica de enjambre de,

José Andrés Castañeda Forno

Trabajo de graduación presentado por José Andrés Castañeda Forno
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2021

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de una red de comunicación wifi e interfaz gráfica para una mesa de pruebas de robótica de enjambre

Trabajo de graduación presentado por José Andrés Castañeda Forno
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2021

Vo.Bo.:

(f) _____

MAEB. Pablo Daniel Mazariegos de la Cerda

Tribunal Examinador:

(f) _____

MAEB. Pablo Daniel Mazariegos de la Cerda

(f) _____

MSc. Miguel Enrique Zea Arenales

(f) _____

Ing. Kurt Emmanuel Kellner Juarez

Fecha de aprobación: Guatemala, 13 de Enero de 2021.

Prefacio

Quiero agradecer a toda mi familia, quienes siempre me han acompañado y apoyado durante todo este camino. Principalmente agradezco a mis padres, Gaby y Mario, ya que son mi mayor ejemplo en la vida y siempre me han exigido dar lo mejor de mi para cumplir todas mis metas. Sin el esfuerzo de todos y cada uno de ellos no estaría donde me encuentro.

De igual forma agradezco a mi asesor, Pablo Mazariegos y Miguel Zea, por el apoyo y quienes a pesar de las circunstancias me exigieron a trabajar con calidad y excelencia. Por último les deseo lo mejor a mis compañeros de carrera y les agradezco toda su ayuda y buenos momentos.

Índice

Lista de figuras

Lista de cuadros

Resumen

En este documento se plantea el desarrollo del software de una mesa de pruebas de robótica de enjambre de bajo costo, por medio de la cual se pueden evaluar los algoritmos de robótica de enjambre para ser validados con agentes físicos en entornos reales. Se describe el desarrollo de la aplicación basada en multithreading desarrollada en python para la interacción entre el usuario y la plataforma, al igual que el despliegue de la información en tiempo real. También se detalla el software implementado en los agentes y como es que este interactúa con el software de la plataforma para poder realizar las simulaciones y cargar el algoritmo de control de una forma eficiente, permitiéndole al usuario obtener la información necesaria para evaluar una gran variedad de algoritmos swarm.

También se detalla la forma en que se implementa la red de comunicación dentro del sistema, como esta es diseñada por el usuario y creada por los agentes al momento de iniciar la simulación. Explicando la flexibilidad que presenta esta red para poder ser implementada en diferentes topologías de comunicación, las cuales pueden ser diseñadas por el usuario desde la aplicación. Se compara el nuevo protocolo de comunicación MQTT con el protocolo usando en la versión anterior del Robotat y cómo es que este presenta un mejor rendimiento al momento de la simulación.

Por último, se evalúan y presentan los resultados obtenidos por medio de un proceso implementado en la aplicación para que la plataforma se autoevalúe y le presente al usuario la cantidad de envíos por segundo que esta puede realizar a cada uno de los agentes. Permitiéndole así al usuario tener más información sobre las condiciones en las cuales se está realizando la simulación.

Abstract

This document proposes the development of the software for a low-cost swarm robotics test table, by means of which swarm robotics algorithms can be evaluated and validated with physical agents and real environments. The development of the multithreading-based application developed in python for the interaction between the user and the platform is described, as well as the display of information in real time. The software implemented in the agents is also detailed and how it interacts with the platform software in order to carry out simulations and load a controller in an efficient way, allowing the user to obtain the necessary information to evaluate a great variety of algorithms.

The way in which the communication network is implemented within the system is also detailed, as it is designed by the user and created by the agents when starting the simulation. Explaining the flexibility that this network presents to be able to be implemented in different communication topologies, which can be designed by the user from the application. The new MQTT communication protocol is compared with the protocol used in the previous version of Robotat and how it presents a better performance at the time of simulation.

Finally, the results obtained are evaluated and presented through a process implemented in the application where the platform can self-evaluate and present the user with the number of payloads send per second that the it can make to each of the topics. This allowing the user to have more information about the conditions in which the simulation is being carried out.

CAPÍTULO 1

Introducción

Esta investigación se centra en el campo de la robótica de enjambre, la cual actualmente está teniendo un gran impacto tanto en el campo industrial como en el académico. Enfocándose en el desarrollo del software necesario para poder implementar una mesa de pruebas de bajo costo, por medio de la cual se puedan evaluar y validar dichos algoritmos de robótica de enjambre. Para llevar a cabo el desarrollo del software de esta plataforma, este se dividió en dos áreas principales. La aplicación, en la cual se realizó el análisis, procesamiento de datos e interfaz gráfica; y, el software que necesitan los agentes para poder ejecutar el algoritmo de control y comunicarse con la aplicación al momento de obtener la información de su pose actual. Tanto el software de la aplicación, como el de los agentes estructuro de forma conjunta para obtener una ejecución fluida y coordinada durante la simulación.

Para llevar a cabo esta investigación realizaron tres pasos generales en cada una de las etapas en las que se dividió el proyecto: investigación, desarrollo y evaluación. Esto con el fin de evaluar el resultado de cada iteración que se realizó en cada área de trabajo y ver posibles mejoras para implementar en la siguiente iteración, logrando así obtener un mejor producto final. En la fase de investigación se analizaron y evaluaron todas las posibles tecnologías que se podían implementar en las diferentes partes del proyecto, tales como el microcontrolador, el lenguaje de programación, librerías, protocolos de comunica con, etc. Luego de definir estas herramientas se procedió a desarrollar el software necesario en cada parte del proyecto para poder cumplir con los objetivos de dichas partes. Además, se evaluaron los resultados obtenidos por separado y en conjunto del sistema general para obtener mejoras que se implementarían en la siguiente iteración, en la cual se repitió nuevamente este proceso hasta que se obtuvo el producto final que logro cumplir con todos los requerimientos del sistema.

Por último, se realizó una evaluación del rendimiento del sistema general involucrando tanto a los agentes, como a la aplicación. Por medio de la cual logramos observar el rendimiento del sistema en sus diferentes configuraciones de red, para los algoritmos de robótica de enjambre.

CAPÍTULO 2

Antecedentes

2.1. Robotarium by Geórgica Tech

Este es un proyecto desarrollado por Georgia Institute of Technology desde principios del 2016, con el fin de facilitar el estudio de robótica de enjambre. A lo largo de este proyecto se han realizado varias publicaciones dentro de las cuales podemos destacar el artículo *Robotarium: A remotely accessible swarm robotics research testbed* . El cual plantea el diseño y desarrollo de una plataforma de pruebas de bajo costo, que pueda ser accesada de forma remota, con el objetivo de facilitarle a estudiantes e investigadores en cualquier parte del mundo el poder probar simulaciones enfocadas a robótica de enjambre. [1]

Este artículo resalta varias características importantes tomadas en cuenta durante el desarrollo del proyecto:

- Integrar medidas de seguridad internas para proteger al robotarium y sus agentes.
- Recolección de información del Robotarium de una forma sencilla y eficiente.
- Realizar un sistema escalable en términos de la cantidad de robots conectados.
- Minimizar el costo individual de los robots.
- Realizar una interfaz pública por medio de la cual el usuario final pueda trabajar a distancia.
- Lograr una conexión eficiente Nube (acceso remoto) - Servidor (Procesamiento de datos) - Robots (Ejecución).

Actualmente el Robotarim cuenta con una plataforma de acceso libre, por medio de la cual se pueden tanto simular, como poner a prueba los algoritmos Swarm en los robots de la plataforma.

2.2. Kilobots de Hardvard

En el 2010 Hardvard público el artículo *Kilobot: A Robotic Module for Demonstrating Behaviors in a LargeScale (2^{10} Units) Collective*. Por medio del cual se presenta un sistema de comportamiento colectivo, escalable y de bajo costo. Este sistema tenía como base a los Kilobots, pequeños agentes que individualmente podían realizar tareas muy simples, pero que podían trabajar en conjunto para poder cumplir objetivos más complejos.[2]

Este proyecto tenía como objetivo principal, el poder desarrollar una plataforma de pruebas de robótica swarm escalable a una gran cantidad de agentes (1024) y de esta forma poder realizar pruebas de algoritmos swarm complejas. Para poder cumplir con este objetivo los desarrolladores se centraron en el diseño los kilobot, enfocándose en las áreas de comunicación, alimentación, costo y programación. Esto atendiendo a la necesidad que el usuario tenga que interactuar lo menos posible de forma individual con cada kilobot y que estos sean altamente replicables. Para lo cual tomaron un cuenta los siguientes aspectos:

- Locomoción por medio de motores de vibración (reducción de costo).
- Comunicación de baja potencia por medio infrarrojo (eficiencia).
- Encendido y apagado por medio de comandos infrarrojos (eficiencia).
- Programación masiva por medio infrarrojo (eficiencia).
- Sensores de bajo costo y potencia: Distancia y luz (reducción de costo).
- Diseño de carga de batería altamente escalable.
- Calculo de información por medio de agentes vecinos.

Este proyecto demuestra la alta escalabilidad con la que cuentan las plataformas swarm. De igual forma nos permite observar la estructura de una red de comunicación de agentes de corta distancia que permite obtener la información necesaria para poder observar simulaciones complejas.

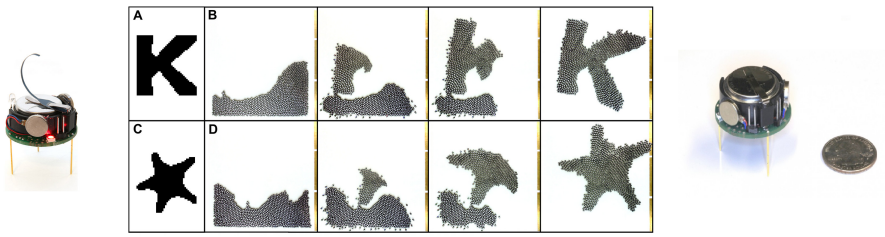

Figura 1: Kilobots Harvard [2].

2.3. Zooids - Swarm User Interface

Este proyecto desarrollado por varias universidades de Francia y Stanford en el año 2016, las cuales publicaron el artículo *Zooids: Building Blocks for Swarm User Interfaces*. Nos presentan una interfaz swarm diseñada para interactuar con el usuario de forma directa. Dicha plataforma constaba de un conjunto de micro robots de ruedas de 2.6cm de diámetro, una estación central de comunicación, un proyector de luz para poder realizar control óptico y un framework para poder desarrollar aplicaciones de control. Este proyecto tomaba en cuenta el factor económico al momento de llevarlo a cabo, ya que cada robot tenía un costo aproximado de 50 dólares americanos, el cual podía ser reducido a 20 dolares al realizar una producción en masa [3].

Cada robot cuenta con :

- Dos motores DC
- Sensores de contacto
- Batería LiPo de 100 mAh
- Microcontrolador ARM TM32F051C8 de 48 MHz
- Comunicación inalámbrica - módulo RF Nordic nRF24L01+.

Dentro de los aspectos más importantes a resaltar en este proyecto es la forma de control al momento de llevar a cabo la interacción con el usuario. Ya que desarrollaron tanto algoritmos de control para cada robot de forma individual como control por gestos para poder controlar a todo el "*enjambre*" completo. Lo cual es una gran ventaja al momento de trabajar en la robótica swarm ya, que los robots se comunican entre sí para poder llevar a cabo las tareas deseadas por el usuario.

De igual forma desarrollaron el control de velocidad y movimiento de forma muy eficiente, ya que controlan y varían tanto la posición, orientación y velocidad de los robots para poder realizar las trayectorias planteadas por el controlador de una forma suave y coordinada para poder evitar la colisión entre los mismos robots.

Figura 2: Zooids [3].

2.4. Robotarium UVG

2.4.1. FASE I - 2017

En octubre de 2017, se presentó la tesis de licenciatura, *Reingeniería de Megaproyectos Fase I-Módulo de Swarm Robotics*, un proyecto desarrollado por los estudiantes de la Uiversidad del Valle de Guatemala. En donde se describe el diseño y construcción de un robot modular destinado a trabajar en enjambre. De igual forma se trabajó en la implementación de la comunicación WiFi por medio del módulo ESP8266.

Para llevar a cabo el proyecto crearon una librería en Matlab que permitiera un uso fácil de la comunicación para cualquier persona con conocimientos básicos de programación en Matlab. Esta librería también está hecha para funcionar con varios robots en la red y que así puedan ejecutar algoritmos de control.

2.4.2. Fase II - 2019

En el 2019 la Universidad del Valle realizó la publicación de un artículo titulado: *Orus and Bitbot: A low cost testbed and robots for Experimentation in Swarm Robotics* por Andre J. Rodas y Marlon J. Castillo. El cual tenía como objetivo el desarrollo de una mesa de pruebas de robótica de enjambre de bajo costo, por medio de la cual se puedan realizar pruebas físicas de algoritmos Swarm. [4]. Durante el desarrollo de esta fase se plantearon y analizaron las diferentes tecnologías disponibles para poder realizar el proyecto de la forma más eficiente posible. El proyecto fue trabajado en diferentes módulos:

- **Visión de computadora**: Procesamiento de imagen por medio de la librería *OpenCV* en C++ y control de pose por medio de un controlador LQR.
- **Red de comunicación inalámbrica**: Utilizando el módulo ESP8266-01 con protocolo TCP, topología de red en estrella e infraestructura como modo de operación
- **Hardware de los robots**: Motores de engranajes planetarios, Microcontrolaros PIC16F887, modulo ESP8266-01 y Battery LiPo 3.3V-700mAh.

Durante el desarrollo de este proyecto se presentaron varios problemas, los cuales fueron planteados como recomendaciones para futuras iteraciones del proyecto. Dentro de las cuales podemos destacar la saturación del servidor al momento de realizar el control de posición para más de cinco agentes, para esto se recomendó realizar el procesamiento de datos por medio de *threads* en el lenguaje de programación Python, ya que este también cuenta con la librería *OpenCV* para realizar el análisis gráfico.

De igual forma se recomendó el desarrollo de un sistema de programación y carga de batería general para todos los robots, esto con el objetivo de poder realizar el sistema lo más escalable posible y que el usuario no tenga que interactuar individualmente con cada robot para utilizar el sistema .

CAPÍTULO 3

Justificación

Actualmente en el área de investigación robótica existen una gran cantidad de trabajos y publicaciones enfocadas al desarrollo de algoritmos y métodos de control para grandes grupos de robots, llamada robótica de enjambre. Estos algoritmos están diseñados para poder controlar de forma organizada a un par o incluso cientos de agentes de forma simultánea. Lastimosamente estas investigaciones generalmente solo llegan a ser validadas por medio de entornos de simulación computarizados, por motivos de costo, tiempo y complejidad en su desarrollo. Para atender a esta problemática actualmente existen plataformas de prueba, principalmente robotariums, las cuales están diseñadas para poder ejecutar dichos algoritmos en mesas de pruebas con robots especializados para realizar estas tareas.

La cantidad de sistemas existentes y el alcance con el que cuentan estos es reducido, ya que no existen muchas de estas plataformas abiertas al público. Hoy en día la universidad del Valle de Guatemala no cuenta con una plataforma para la experimentación y prueba de algoritmos enfocados a robótica de enjambre. El desarrollo de estos ambientes de experimentación controlados permitiría aumentar las investigaciones en dicha área, tanto dentro como fuera de la universidad, ya que actualmente la robótica de enjambre cuenta con varias aplicaciones como: Búsqueda y rescate luego de desastres naturales, Modelado de comportamientos animales, nanorobotica, etc.

Por lo mencionado anteriormente, en este trabajo se presenta el diseño e implementación de una aplicación desarrollada en Python, basa en multithreading capaz de realizar una rutina de control de posición para los robots, el despliegue de una interfaz gráfica en tiempo real para la interpretación de datos por parte del usuario y el desarrollo de una red de comunicación inalámbrica por medio de WIFI para el envió de información a los agentes. Parte fundamental de este módulo es que sea altamente escalable y eficiente para poder realizar pruebas con múltiples agentes de forma simultánea. Para el desarrollo de esta fase se tomaron en cuenta las recomendaciones de las fases anteriores de este proyecto. Por lo que la aplicación será desarrollada en el lenguaje de programación python utilizando multithreading para realizar el procesamiento de datos de una forma más eficiente, lo cual nos permitirá incrementar la cantidad de robots con lo que se trabaja de forma simultánea.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Diseñar e implementar una red de comunicación bidireccional por medio de wifi que permita a la plataforma robotat programar, controlar y diagnosticar a los bitbots.

4.2. Objetivos específicos

- Implementar y desarrollar un sistema de carga de programa de forma inalámbrica y simultanea general para todos los robots mediante un bootloader interno en el microcontrolador.
- Desarrollo e implementación de una aplicación basada en multithreading capaz de realizar la rutina de control de los robots, desplegar una interfaz gráfica en tiempo real y enviar información a los bitbots basada en el lenguaje de programación python
- Definir la cantidad de agentes máximos para poder tener un rendimiento ideal en la conexión del sistema durante su funcionamiento (10 envíos de datos por segundo)

CAPÍTULO 5

Alcance

Este trabajo se centra únicamente en el desarrollo de tres áreas del Robot. La primera, se centra en el diseño e implementación de una red WLAN por medio de WIFI, la cual se utilizará como canal de comunicación entre la aplicación y los agentes del Robotarium. La segunda, es el desarrollo de una GUI (Graphical User Interface), por medio de la cual el usuario podrá interactuar con la platafroma y observar los resultados de los algoritmos swarm. Por último, la implementación de un proceso que permita la carga automática de un nuevo archivo principal a cada uno de los agentes de forma eficiente. Estas tres áreas se ven complementadas con el trabajo de Jose Molina, encargado de desarrollar el controlador de posición y desarrollar las capacidades de sensado por medio de visión de computadora y odometría.

Debido a las condiciones de trabajo en las cuales se desarrollo esta investigación, dada la pandemia global por el COVID19 en el 2020, este trabajo únicamente se centra en el desarrollo de las tres áreas mencionadas anteriormente con 10 agentes, ya que no se cuenta con todo los recursos necesarios para poder realizar las pruebas a mayor escala y evaluar el sistema completo. De igual forma, estas áreas serán evaluadas y presentadas de forma aislada al resto de los trabajos mencionados anteriormente.

CAPÍTULO 6

Marco teórico

6.1. Robótica de enjambre

La robótica de enjambre es un área de la robótica enfocada al estudio y desarrollo de sistemas inspirados en el comportamiento organizado de grupos de animales sociales. Por medio del cual, a través de tareas simples y la interacción entre cada uno de los agentes se pueda obtener un comportamiento coordinado de parte de un gran número de agentes para poder cumplir un objetivo complejo. Como podemos observar en las Figura 3, los agentes se coordinan para poder realizar una formación más compleja.

Figura 3: Robótica de enjambre - Agrupación de agentes [5].

Dentro de las principales áreas de aplicaciones para estos sistemas están:

- Observación del comportamiento de organismos celulares.
- Enfoque educativo - Investigación de algoritmos Swarm.
- Búsqueda y rescate - Minimizando el riesgo humano.
- Militares - Observación y análisis de terreno.

6.2. Comunicación WIFI

WIFI es el acrónimo de Wireless Fidelity, un tipo de interconexión de red inalámbrica de alta velocidad y corta distancia (WLAN). Por medio de la cual, los dispositivos habilitados con wifi pueden conectarse entre sí o a internet a través de un punto de acceso de red inalámbrica (Gateways). Esta tecnología actualmente está teniendo un gran impacto en el desarrollo e implementación de nuevas tecnologías, principalmente en el área de IOT/IOE, ya que nos permite la interconexión de una alta cantidad de dispositivos entre si. [6]

6.2.1. MODELO OSI

Es un modelo desarrollado a finales de la década de los años setenta con el objetivo de normalizar la conexión en red llamado *Open System Interconnection reference model*. Este modelo divide en siete capas el proceso de transición de información entre dos dispositivos, donde cada capa esta encargada de ejecutar una parte específica del proceso general de forma jerárquica (Capas). [7]

Las capas de modelo describen el proceso de transmisión de los datos dentro de una red, en la siguiente figura podemos observar la estructura y capas de este modelo. Durante el envío de datos, estos pasan de forma descendente por el modelo OSI pero al momento de recibirlos, estos pasan de forma ascendente.

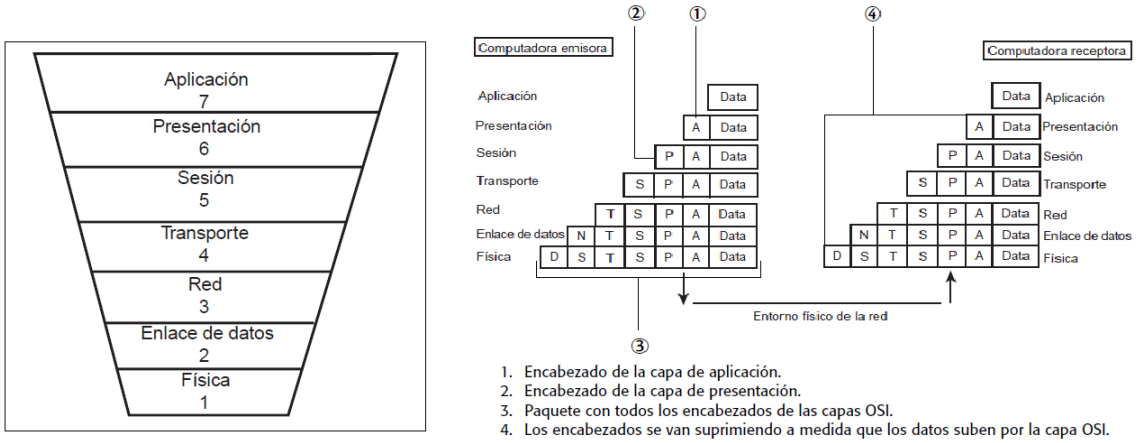

Figura 4: Estructura de capas del modelo OSI [7].

6.2.2. Las capas OSI

Capa de aplicación

Esta capa proporciona la interfaz de servicios que soportan las aplicaciones del usuario y ofrecer acceso general a la red. Entre los servicios de intercambio de información que gestiona la capa de aplicación se encuentra la web, servicios de correo electrónico y aplicaciones de cliente servidor.

Capa de presentación

La capa de presentación es considerada como el traductor de este modelo, ya que es el encargado de tomar los paquetes enviados por la capa de aplicación y colocarlos en un formato genérico, el cual puedan leer todas las computadoras. Durante este proceso también se encarga de cifrar y comprimir los datos para reducir el tamaño del paquete.

Capa de sesión

Esta capa es la encargada de establecer el enlace de comunicación entre las dos computadoras, de igual forma gestiona la sesión que se establece entre ambos nodos. Cuando ya se encuentra establecida la sesión, esta capa se encarga de colocar puntos de control en la secuencia de datos, con el objetivo de restablecer la comunicación y envió de datos desde el punto de control al momento de una falla en la conexión y no enviar nuevamente todo el paquete de datos. Los protocolos de comunicación que operan en esta capa pueden proporcionar distintos tipos de enfoque para el envió de datos.

Capa de transporte

Esta capa es la encargada de controlar el flujo de datos entre los nodos establecidos en la comunicación. Durante este proceso los datos deben de ser entregados en la secuencia indicada y sin errores. Esta capa es la encarga de evaluar el tamaño de los paquetes con el fin de que estos cumplan con el tamaño requerido por las capas inferiores del conjunto de protocolos, dicho tamaño es indicado por la arquitectura de la red que se esté utilizando.

Capa de red

Es la encargada de determinar la ruta que deben de seguir los paquetes y entregarlos de forma eficiente. En esta parte las direcciones lógicas se convierten en direcciones físicas, las direcciones de hardware de la NIC, tarjeta de interfaz para red. Dentro de esta capa operan los routers y utilizan los protocolos de encaminamiento para poder determinar la ruta a seguir por los paquetes de datos.

Capa de enlace de datos

Al momento en que los datos llegan a esta capa se encuentran estructurados en tramas (unidades de datos), las cuales están definidas por la arquitectura de la red utilizada. Esta capa se encarga de desplazar los datos por el enlace físico de comunicación hasta el nodo receptor. Identifica cada computadora dentro de la red con su dirección de hardware asignada dentro de la tarjeta de interfaz para red.

Capa física

Dentro de esta capa las tramas procedentes de la capa de enlace de datos se convierten en una secuencia de bits que pueden ser transmitidos por el entorno físico de la red. Esta capa también determina los aspectos físicos de conexión en la red. En el nodo receptor, los datos son recibidos en la secuencia de bits enviada.

6.3. Protocolos de transporte de información

6.3.1. Protocolo TCP/IP

El protocolo de comunicación TCP / IP es la base de la comunicación en redes de área local y área extensa. Este protocolo se da en la capa de transporte del modelo OSI y está basado en la comunicación por medio de la dirección IP de cada uno de los host conectados a la red. Por medio de este protocolo se garantiza que los paquetes lleguen en secuencia y sin errores, al intercambiar la confirmación de la recepción de los datos y retransmitir los paquetes perdidos. Este tipo de comunicación se conoce como transmisión de punto a punto

6.3.2. Protocolo SOCKETS

Los sockets son un mecanismo de comunicación entre procesos/maquinas por medio de un canal de notificación directo que emite y/o recibe información, de otro proceso incluso estando en distintas máquinas. Este mecanismo se implementa principalmente en la comunicación entre diferentes maquinas o dispositivos, ya que dichos dispositivos no deben estar solicitando la información y verificando si fue enviada, ya que el socket les notificara cuando la información está disponible.

6.3.3. Protocolo MQTT

MQTT (Message Queue Telemetry Transport) es un protocolo de transporte de mensajes Cliente/Servidor basado en publicaciones y subscripciones a los denominados "tópicos". Cada vez que un mensaje es publicado será recibido por el resto de los dispositivos suscritos a un tópico del protocolo. Este protocolo es idóneo para aplicaciones de IOT en las cuales se envían pequeños paquetes de datos, por lo que no necesita un gran ancho de banda. El protocolo MQTT funciona principalmente sobre TCP/IP aunque también sobre otros protocolos de red con soporte bi-direccional y sin pérdidas de datos [8]. Sus principales características son:

- Uso de mensajes "broadcast" para subscripción y publicación de datos con independencia de la aplicación.
- Transporte de mensajes transparente y con un flujo de datos optimizado lo cual permite reducir el tráfico en la red.
- Posee un mecanismo de notificación de desconexiones inesperadas.

6.3.4. Topología de Red

La topología de una red es la disposición física en la cual se conecta una res de dispositivos. A continuación se presentan las topologías más comunes [7]:

- **Red de anillo:** Cada nodo está conectado al siguiente y la última se conecta con la primera. Cada estación tiene un receptor y un transmisor que hace la función de un repetidor, pasando la señal al siguiente nodo del anillo.
- **Red en estrella:** Todos los nodos están conectados directamente a un servidor, por medio del cual pasa todo el tráfico de datos.
- **Red en malla:** Cada nodo está conectado a uno o más entre los otros nodos. De esta manera se logra llevar el mensaje a un nodo por diferentes caminos.
- **Red de linea:** Un conjunto de nodos conectados en una línea. Cada nodo se conecta a sus dos nodos vecinos excepto el nodo final e inicial.
- **Red de árbol:** Esta red cuenta con un cable principal al que hay conectadas redes individuales en bus (colocadas en forma de árbol). Dentro de las cuales se comparte el mismo canal de comunicaciones.
- **Red de bus:** Todos los nodos se encuentran conectadas por un único canal (bus) de comunicación. Por lo que la comunicación es visible para todos.

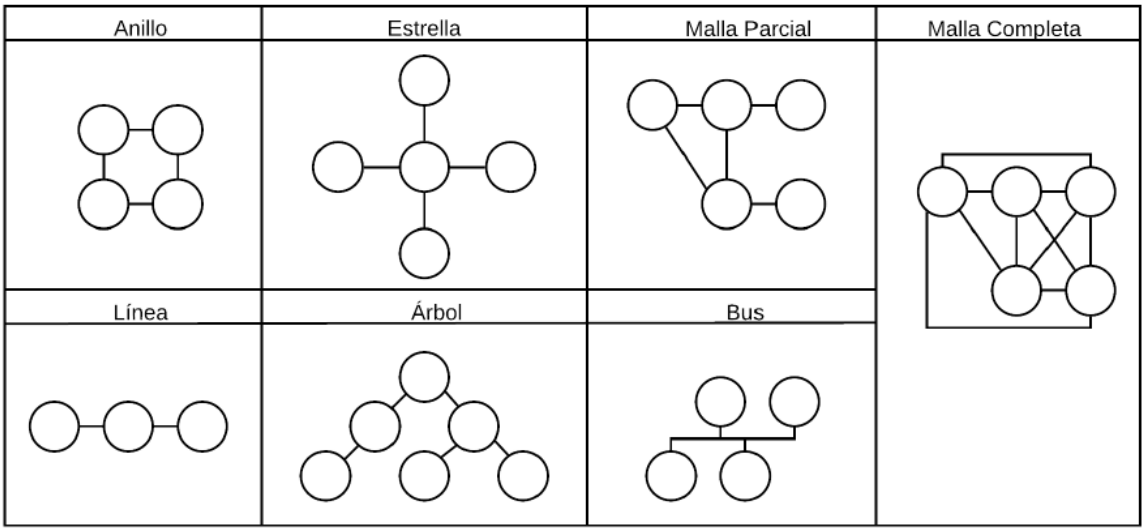

Figura 5: Topologías de red [9].

6.3.5. Modos de operación

El conjunto de estándares IEEE 802.11 definen dos modos para las redes inalámbricas descritos a continuación. Es importante resaltar que no siempre los modos se ven reflejados directamente en la topología. Por ejemplo, un enlace punto a punto puede ser del modo ad hoc. El modo puede ser visto como la configuración individual en la tarjeta inalámbrica de un modo más que como una característica de toda la infraestructura.

Ad hoc

El modo *ad hoc*, también conocido como punto a punto, es un método para que los clientes inalámbricos puedan establecer una conexión de forma directa entre sí. Para lograr esto a cada cliente inalámbrico en una red *ad hoc* se le debe de configurar su andador inalámbrico en *ad hoc* y usar los mismos SSID y numero de canal de la red. En una red *ad hoc* el rendimiento es menor al número de nodos.

Infrstructura

Este modo cuenta con un elemento de coordinación, un punto de acceso o estación base. El punto de acceso se conecta a una red y los clientes inalámbricos pueden acceder a la red fija atreves del punto de acceso para interconectar muchos puntos de acceso y clientes inalámbricos, todos deben configurarse con el mismo SSID. A diferencia del modo ad hoc, este modo nos permite la interconexión de un número mayor de clientes.[9]

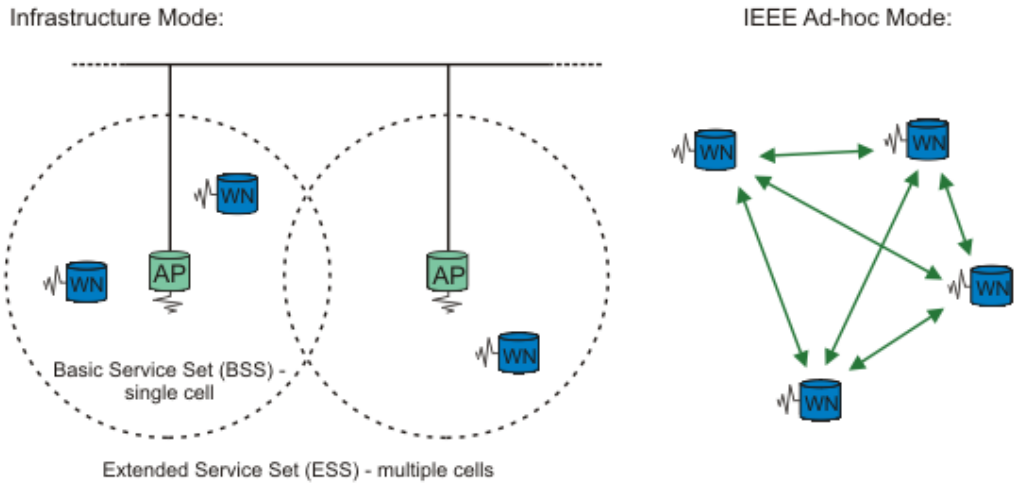

Figura 6: Comparación de los modos de comunicación [10].

6.3.6. Seguridad en redes inalámbricas

La conexión por medio de la tecnología WiFi, a pesar de ser inalámbrica cuenta con riesgos en la seguridad a la información de los usuarios y dispositivos que utilizan esta tecnología[11]. Por esto se utilizan varios métodos de seguridad dentro de las conexiones, a continuación, se presentan los principales:

- **WEP - Wired Equivalent Privacy:** Durante este proceso de seguridad un punto de acceso debe de autenticar a una estación antes de que esta se asocie al mismo. Lo que se autentica con WEP son las estaciones y no los usuarios. Esto lo realiza por medio de mecanismo de desafío/respuesta con una clave secreta (de 64 o 128 bits) compartida por la estación y el punto de acceso, de manera que se niega el acceso a todo aquel que no tenga la clave asignada.

  La principal debilidad de WEP es que la clave compartida que se usa para encriptar y desencriptar las tramas de datos es la misma que la que se usa para la autenticación. Otra debilidad importante es que siempre se utiliza la misma clave de encriptación en la red.

- **WAP - WiFi Protected Access:** Utiliza el método PSK (Pre-Shared Key). Solamente se necesita introducir una clave maestra o PSK en cada uno de los puntos de acceso y de las estaciones que conforman nuestra WLAN. Solamente podrán acceder al punto de acceso los dispositivos móvile cuya contraseña coincida con la del punto de acceso.

- **WAP2:** Una de las principales diferencias entre WPA y WPA2 es el uso del algoritmo AES; este es un tipo de cifrado por bloques, adoptado como estándar de cifrado por los EE.UU., el cual permite claves más largas y más seguras y, además, la implementación del CCMP (Modo de contador cifrado bloqueo de encadenamiento protocolo de código de autenticación de mensajes) que es un protocolo mejorado de encriptación.

6.3.7. Grafic User Interface - GUI

Una interfaz gráfica para el usuario (GUI) es un sistema de componentes visuales e interactivos implementados en software. Con el objetivo de desplegar información y permitirle al usuario interactuar de forma directa con el sistema. En una GUI se encuentran implementados los iconos, botones, cajas de texto, etc. Esto le permite al sistema ser más amigable con el usuario que el implementar una linea de comandos (CUI).

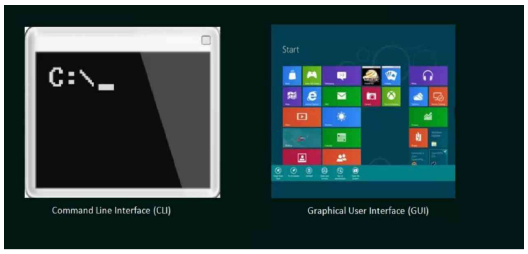

Figura 7: Comparación GUI vs CUI [12].

Para desarrollar una GUI pyton cuenta con *Tkinter* que es un binding de la biblioteca gráfica Tcl/Tk. Tkinter es una buena opcion para desarrollar GUI's de aplicaciones que tienen un enfoque de configuración y manejo de datos [13].

6.4. Sistemas embebidos

Un sistema embebido es una combinación de componentes de hardware y software capaces de ser programados para poder realizar una tarea específica o alguna función dentro de un sistema más complejo. La gran mayoría de estos sistemas cuentan con un microcontrolador, el cual esta encargado de llevar a cabo todos los procesos lógicos necesarios. De igual forma cuentan con una interfaz de usuario (UI), que no llega a ser tan compleja como la GUI ya que únicamente cuenta con bottones, leds, sensores, etc.

Como características principal este tipo de sistemas suelen ser de bajo costo, bajo consumo energético, bajo peso y baja cantidad de recursos de cómputo. Esto con el objetivo de poder realizar su tarea de la forma más eficiente y económicamente posible. Actualmente este tipo de sistemas están teniendo un gran impacto en nuestra sociedad ya que casi cualquier aparato electrónico cuenta con un sistema embebido.

Figura 8: Sistemas embebidos [14].

6.4.1. Microcontrolador

Un microcontrolador es un circuito integrado capaz de poder ser programado para poder realizar una tarea en específico. Como mencionamos anteriormente, los microcontroladores se utilizan principalmente en el desarrollo de sistemas embebidos ya que suelen ser de bajo costo y bajo consumo energético. De igual forma cuentan con varios módulos, protocolos y buses periféricos como se muestra en la Figura 8, que permiten poder realizar aplicaciones más complejas en menos espacio.

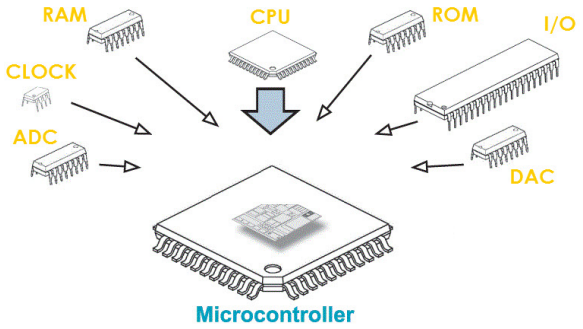

Figura 9: Estructura de un microcontrolador [15].

6.4.2. Módulo ESP8266

Es un chip con modulo WIFI desarrollado por Ai-thinker Team[16]. El cual cuenta con un microcontrolador Tensilica L106 de 32 bits de bajo consumo. Este módulo ha tenido un gran impacto en el área de IOT debido a sus siguientes características:

Hardware

- Voltaje de operación entre 3V y 3,6V.
- Velocidad de reloj 160MHz.
- Temperatura de operación -40ºC y 125ºC.
- Soporta los principales buses de comunicación (SPI, I2C, UART).

Conectividad

- Soporta IPv4 y los protocolos TCP/UDP/HTTP/FTP.
- Estandar IEEE802.11 b/g/n.
- Configuración como STA/AP/STA + AP.

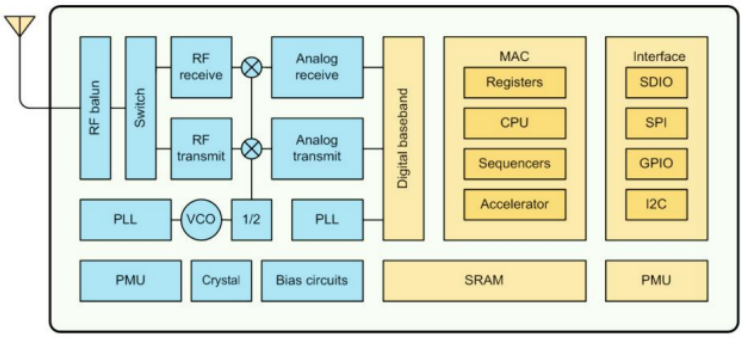

Figura 10: ESP8866-01 Diagrama de bloques [16].

Este módulo inicialmente es programado por comandos AT por medio del puerto UART, pero puede ser programado por medio de Arduino o Micropython (Python enfocado en microcontroladores) al momento de cargarles el respectivo firmware. Al realizar este cambio en el firmware del ESP8266 logramos obtener mayor flexibilidad en el uso de este, ya que dichas plataformas cuentan con una amplia variedad de librerías que permiten y facilitan el uso de todos los recursos del microcontrolador.

6.5. Micropython - Python for microcontrolers

Micropython es un firmware para microcontroladores ligero que implementa el lenguaje de programación **python3** incluyendo un pequeño set de librerías estándares optimizadas para poder ser implementadas en microcontroladores y en ambientes con recursos limitados [17].

Este firmware es compatible con python normal (para pc), lo cual facilita la comunicación entre Microcontrolador - Servidor, al momento de desarrollar sistemas con dicha estructura. De igual forma cuenta con varias librerías que nos permiten aprovechar todos los recursos del ESP8266 al momento de desarrollar este proyecto, dentro de las principales se encuentran:

- Machine, os → Controlamos hardware y sistema de archivos interno de micropython.
- Socket, network, json → Utilizar el módulo wifi y comunicarnos con otro dispositivo.
- Btree → Administrar una base de datos interna.

Para poder cargar dicho firmware en el microcontrolador ESP8266, es necesario utilizar la plataforma **esptool.py** de espressif, repositorio de github: `https://github.com/espressif/esptool`. Por medio de la cual logramos comunicarnos con el bootloader ROM del ESP8266 chips.

También es necesario utilizar la Adafruit micropython tool**(Ampy)** desarrollada por Adafruit, repositorio de `github:https://github.com/scientifichackers/ampy`. Es una herramienta implementada desde la línea de comandos, por medio de la cual podemos administrar los archivos en el microcontrolador desde la PC por medio del puerto serial [18].

6.6. Eclipse Mosquitto

Es un software de código abierto (EOL/EDL licensed) para implementar la comunicación del modelo publicador/subscriptor, por medio de un broquer que utiliza el protocolo de comunicación MQTT, en su version 5.0, 3.1.1 y 3.1. Gracias a su tamaño reducido, mosquitto, puede ser implementado en una gran cantidad de dispositivos, sin exigir tantos recursos para su funcionamiento. Micropython cuenta con el software necesario para poder implementar esta tecnología [19].

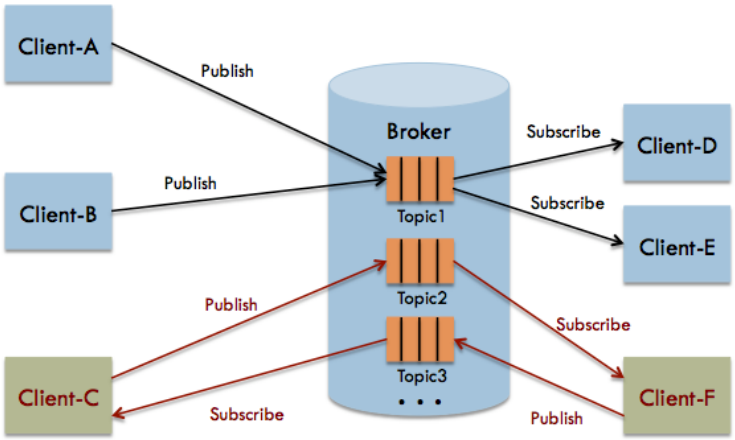

Figura 11: Estructura de un Broker [20].

6.6.1. Bootloader

El Bootloader es una pieza de código ejecutada al inicio del programa, encargada de verificar y ejecutar alguna actualización en el código principal en la memoria. La implementación de un bootloader en un sistema embebido se hace con el objetivo de poder reprogramar nuestro microcontrolador a distancia y de forma inalámbrica. Cabe resaltar que, este código es desarrollado para cumplir con las necesidades de nuestro sistema y está basado en la estructura de nuestro microcontrolador, tomando en cuenta la arquitectura, cantidad de memoria, protocolos de comunicación, etc.

6.6.2. Multithreading

Multithreading es un modelo de programación en paralelo mediante el cual se trabaja con varios hilos encargados ejecutar ciertas lineas de código. Estos hilos son ejecutados de forma alternante con el objetivo de mejorar el rendimiento en el tiempo de ejecución del programa. [21]

Este tipo de modelo se implementa con una arquitectura de memoria compartida como se muestra en la figura 12, por lo que todos los hilos cuentan con acceso al mismo espacio de memoria. Debido a esto es muy importante lograr la sincronización entre las diferentes threads para evitar problemas al momento de actualizar y leer valores en la memoria. Este tipo de modelo se implementa principalmente para poder afrontar problemas de actualización o manejo de datos dentro del programa, no mucho para el análisis de datos. Para lograr la sincronización de las diferentes threads en el sistema se implementan los diferentes métodos de sincronización, entre los que podemos encontrar:

- Sincronización por medio de Lock y RLock: Cuando una thread desea modificar un espacio de memoria debe adquirir la "*Llave*" antes de modificarla, luego de modificarla la thread libera la "*Llave*" para que alguien más pueda accesar a dicho espacio de memoria.
- Sincronización por medio de semáforos: El módulo cuenta con una variable interna que se decrementa cada solicitud de acceso a la memoria y se incrementa cada ves que una thread libera dicho espacio de memoria. Si el valor no es negativo le permite el acceso a la siguiente thread.
- Sincronización por medio de condiciones: Durante este método una thread espera un evento para acceder a la memoria y otra thread notifica cuando se da dicho evento.
- Sincronización por medio de eventos: Un objeto cuenta con bandera interna que comunica a las threads cuando el espacio está disponible para modificar.
- Comunicación entre threads por medio de Queue: Funcionan por medio de un Queue que funciona como un embudo"de acceso para los recursos de cada thread, por lo que nos permite obtener resultados más seguros y un diseño del código mucho más entendible.

Python actualmente cuenta con varias librerías que nos permiten implementar multithreading en nuestros programas. La librería más utilizada para este tipo de programación es **threading**, la cual nos permite definir los hilos de trabajo dentro del código y acceder a todas las clases y subclases de sincronización de datos descritas anteriormente.

Multiprocessing

Este modelo de programación está orientado a crear diferentes procesos de trabajo, cada proceso copia de la memoria de forma independiente y se ejecuta de forma paralela, ya que cada uno cuenta con su propio interprete. Dentro de cada proceso pueden declararse diferentes threads de trabajo, por lo que implementar multiprocesasing tiene un impacto en el consumo de memoria que utilizara el sistema. Este tipo de modelo está orientado a realizarse en tareas que conllevan el análisis de información de grandes cantidades de datos para poder realizarlos en tiempos significativamente menores. [21]

De igual forma, pyhon cuenta con una librería principal llamada **Multiprocessing** para poder eficientizar el tiempo de ejecución de nuestro código. Algo importante de mencionar es que al aplicar multiprocessing utilizan muchos recursos de la computadora, ya que cada procesamiento es como tener un programa nuevo ejecutándose al mismo tiempo y esto puede llegar a tener un impacto en el rendimiento de nuestro sistema, si nuestro hardware no tiene los recursos necesarios. a continuación, vemos una imagen que explica las diferencias mencionadas entre multiprocessing y multithreading.

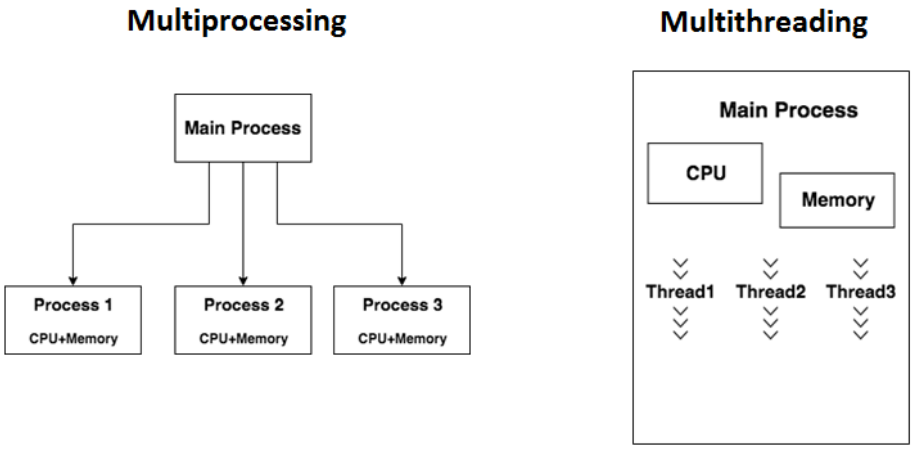

Figura 12: Multithreding vs Multiprocessing [22] .

CAPÍTULO 7

Metodología

Para llevar a cabo este proyecto se han definido cuatro áreas principales de trabajo, dentro de las cuales se encuentra tanto el desarrollo de software como de hardware de la plataforma de pruebas, siendo estas las siguientes:

- Diseño del hardware de los agentes.
- Desarrollo de software de control por visión de computadora.
- Desarrollo de software de comunicación wifi.
- Desarrollo de una interfaz gráfica para el usuario.

En la Figurar 11, se presenta el esquema de componentes y su configuración dentro del proyecto. De igual forma, podemos observar las separaciones del área de hardware (agentes), software y como estos se van a comunicar por medio de una red WIFI.

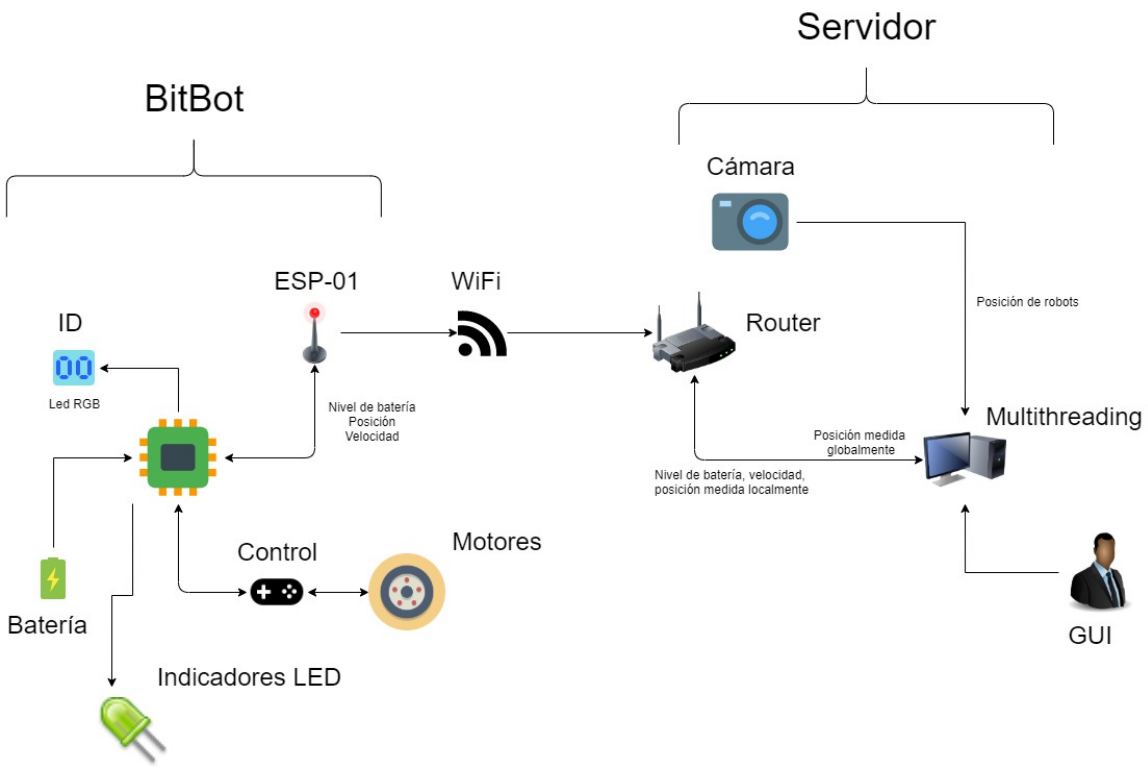

Figura 13: Esquema de trabajo del Robotat.

Diseño del hardware de los agentes El desarrollo del hardware en este proyecto se centrará en los agentes, ya que la mesa de pruebas realizada en la fase anterior se encuentra en buenas condiciones de trabajo. El diseño de los robots se realizará en dos placas, la superior contará con los módulos de comunicación WIFI y procesamiento de datos, mientras que la inferior contará con los módulos de potencia para las llantas. Dentro de los trabajos principales a realizar en esta área se encuentra la implementación de un sistema eficiente de carga y almacenamiento de energía. Al igual que un diseño PCB eficiente, el cual funcione como estructura del agente, como el mostrado en la Figura 14.

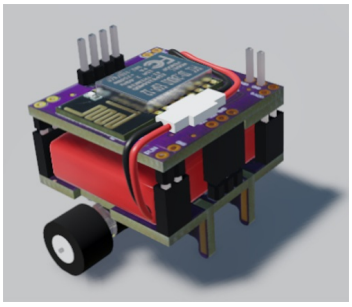

Figura 14: Diseño de los agentes [23].

Desarrollo de software de control por visión por computadora

En la parte del servidor se implementará un sistema de control y análisis de posición de los robots, esta es una de las partes más importantes del sistema ya que por medio de está se controlarán a los agentes dentro de la plataforma. Este sistema contará con una etapa de reconocimiento de imagen por medio de la cámara de la mesa y el procesamiento de la imagen que se realizará con ayuda de la librería *Open CV* en python.

Por medio de este análisis se podrá obtener la pose de cada uno de los agentes dentro de la plataforma con respecto de un marco de referencia general. Esto nos permitirá poder indicarles a los agentes las trayectorias que deben seguir durante la implementación de algún algoritmo. Este sistema se estará ejecutando en conjunto con las demás tareas del servidor por medio de multithreading, con el fin de obtener una mayor eficiencia en el tiempo de ejecución.

Desarrollo de una interfaz gráfica para el usuario

Para permitirle al usuario la interacción con el sistema se desarrollará una GUI o interfaz gráfica de usuario, está interfaz será desarrollada en Python por medio de la librería **Tkinter**. Con está interfaz el usuario podrá obtener la información necesaria de la plataforma, configurar parámetros de funcionamiento, cargar programas, etc. El objetivo principal de la GUI será el de poder desplegar la información requerida por el usuario de la forma más amigable y simple posible para su interpretación, al igual que permitirle el exportar los datos de cada simulación si lo desea. Para asegurarnos que la aplicación cumpla con los requerimientos necesarios, se trabajará en diferentes fases, cada una con diferentes objetivos.

Como primera fase se desarrollará un mockup de la aplicación, el cual contará con todas las vistas y ventanas de la misma, esto con el objetivo de asegurarnos que la estructura del sistema cumpla con todos los requerimientos desde el principio y evitar cambios grandes a futuro, de igual forma, nos aseguramos de que la navegación dentro de la aplicación sea lo más amigable posible.

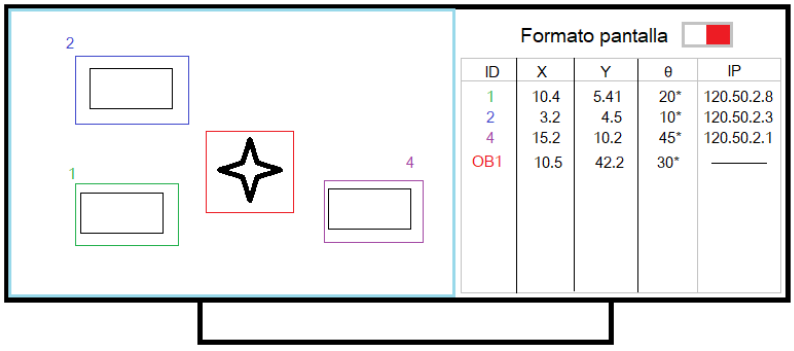

Figura 15: Mockup página principal.

Como ya se mencionó anteriormente para asegurarnos que el sistema sea lo más eficiente, se trabajará con multithreading, por lo que se realizarán pruebas en las cuales se medirá el tiempo de ejecución de varias ramas del programa. De igual forma se implementarán varias pruebas con información "$artificial$" antes de obtener los datos de Open CV para asegurarnos de un despliegue y procesamiento de datos eficiente.

Desarrollo de software de comunicación

Esta sección se centra en el diseño y desarrollo de una red de comunicación entre la aplicación y los agentes (módulo ESP8266-12e) por medio de la red WIFI . Dentro de los aspectos a tomar en cuenta durante el desarrollo de está sección es que el tiempo de envió de datos cumpla con los requerimientos del sistema, es decir, tengamos una latencia baja. De igual forma es necesario asegurarnos que dicha estructura sea escalable, para poder seguir obteniendo un buen rendimiento del sistema al incrementar la cantidad de agentes en al simulación.

Dentro de esta sección se desarrollará el software necesario tanto en el lado de la aplicación (servidor), como el software necesario en los microcontroladores para poder obtener una conexión estable. Dicha conexión se realizará por medio de un servidor MQTT, el cual se encontrará en la computadora central (servidor). Ya que de esta forma lograremos crear una red privada sin necesidad de tener una salida a internet, lo cual nos afectaría el tiempo de comunicación.

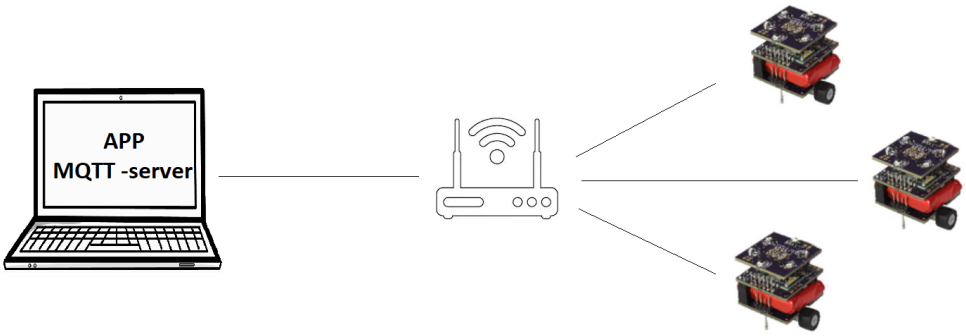

Figura 16: Diseño de la red de comunicación.

CAPÍTULO 8

Diseño experimental

8.1. Bootloader

Objetivo: Diseñar e implementar un sistema de carga de archivos masivo e inalámbrico que sea efectivo para 10 agentes.

Metodología: Crear un servidor web en la misma computadora donde se encuentra la aplicación, el cual contendrá el archivo a cargar a cada uno de los agentes, dicho archivo sera seleccionado desde la aplicación al momento de configurar la simulación. Al colocar el archivo en el servidor, se les notificará a los agentes que deben conectarse, descargar el archivo y almacenarlo como el archivo principal (main.py). Para esto es importante mencionar que la computadora en la cual se realicen las pruebas debe contar con un ambiente de desarrollo instalado, como Wamp o Xamp.

Materiales:

- 10 ESP8266 con micropython - Clientes.
- Computadora central con Wamp/Xamp - Servidor.
- Red Wifi - Canal de comunicación.

Los resultados de esta sección estarán representados por la cantidad de cargas de código exitosas por cada prueba. Al mismo tiempo se estará evaluando el comportamiento del sistema al ir incrementando la cantidad de agentes en la red, de esta forma podremos definir la escalabilidad del sistema de comunicación implementado.

8.2. Desarrollo de GUI

Objetivo: Desarrollar una GUI intuitiva y fácil de utilizar para el usuario

Metodología:

- Definir los pasos a seguir para la configuración de la plataforma, uso de está y manipulación de información obtenida luego de haber realizado la simulación.
- Implementar esta secuencia de pasos en la GUI de forma intuitiva y amigable para el usuario.
- Definir y desarrollar los diferentes objetos (tablas, gráficas, imágenes) por medio de los cuales se desplegará la información de la simulación en tiempo real.
- Realizar pruebas sobre la navegación del usuario y flujo del contenido dentro de la plataforma.
- Realizar las validaciones y alertas necesarias para facilitar la navegación y operación del sistema.

Herramientas:

- Python - Tkinter

Esta sección será evaluada por medio de una prueba de control de calidad, por medio de la cual se verificarán todas las validaciones necesarias que debe de tener el programa para funcionar fluidamente. De igual forma se validará que todas las herramientas de la aplicación funcionen correctamente y la información este siendo desplegada y almacenada de forma correcta.

8.3. Software de comunicación

Objetivo: Desarrollar el software para la comunicación inalámbrica tanto en la aplicación como en los microcontroladores.

Metodología: Definir la estructura de comunicación y los pasos necesarios para poder implementar una comunicación efectiva y funcional para el sistema, por medio de la cual se establezca un canal de comunicación con cada agente. Tanto el software del microcontrolador como el de la aplicación deberán estar relacionados para permitir un flujo óptimo en el funcionamiento al momento de la simulación.

Materiales:

- ESP8266 con Micropython
- ampy y esptool.py para el manejo de los archivos del microcontrolador
- Gui desarrollada
- Red WIFI

Está prueba sera evaluada por medio del flujo exitoso de información durante las pruebas de simulación.

8.4. Rendimiento del sistema

Objetivo: Medir y definir la tendencia del rendimiento del sistema en ciclos de ejecución y envío de datos por segundo de la aplicación, según a la cantidad de agentes dentro de la simulación.

Metodología: Medir la cantidad de ciclos de ejecución de la aplicación por segundo con: 1, 2, 5, 8 y 10 agentes y de esta forma poder graficar la cantidad de ciclos realizados en cada prueba y definir la tendencia del sistema al ir incrementando la cantidad de agentes, lo cual nos permitirá observar la escalabilidad del sistema. Para realizar estas pruebas se colocará un timer de 1 segundo en la aplicación y se llevará el registro de la cantidad de ciclos realizados durante ese tiempo por cada prueba.

Materiales:

- Red de comunicación WIFI
- Software de análisis de visión de computadora terminado
- Software de comunicación desarrollado tanto en los agentes como en la aplicación
- Red WIFI

CAPÍTULO 9

Aplicación Robotat - Python

Como se ha estado mencionando anteriormente, el sistema del Robotat cuenta con una aplicación principal, la cual es la encargada de realizar todo el procesamiento, análisis y envió de datos hacia los agentes, está arquitectura se definió de dicha forma, para poder aprovechar la mejor capacidad de procesamiento con la que cuentan las computadoras o servidores sobre los microcontroladores, permitiendo a los agentes obtener la información ya procesada y únicamente deben aplicar un algoritmo de control, el cual es computacionalmente más económico que el análisis realizado por la aplicación, para poder efectuar el comportamiento deseado por el usuario.

Figura 17: Capacidad de procesamiento.

En este capítulo se explicarán todos los procesos internos que realiza la aplicación, al igual que el flujo de estos durante el ciclo de uso por medio de la interfaz gráfica de usuario. Se describirá tanto el pseudocódigo, como el enlace entre la aplicación y los agentes para obtener un funcionamiento fluido durante la simulación. Para el desarrollo de la aplicación se tomaron como base los resultados, recomendaciones y problemas obtenidos en la fase anterior, para poder obtener un sistema más eficiente.

9.1. Definición de tecnologías

Antes de empezar con el desarrollo del software tanto de la aplicación como el de los microcontroladores, se verificaron todas las recomendaciones dadas en la fase anterior. Tomando en cuenta, tanto la comunicación, hardware implementado y lenguaje de programación utilizado. En base a eso, se decidió evaluar otro tipo de tecnologías, principalmente en cuanto al microcontrolador implementado y el lenguaje de programación. Con esto se busco establecer una misma base de trabajo tanto para la aplicación como para los agentes.

En la fase anterior del proyecto, se presentaron bastantes problemas al momento de querer unificar las diferentes tecnologías y plataformas en las que estaba desarrollado el sistema, ya que la etapa de comunicación entre el módulo de comunicación WIFI y el microcontrolador estaba desarrollada en C. De igual forma la parte de comunicación entre los agentes y la aplicación estaba desarrollada en C++ por medio de sockets y toda la información era almacenada y leída en una base de datos. Debido a esto se obtuvo un habiente de comunicación entre las diferentes etapas no muy fluido, lo cual llegó a tener un gran impacto en el rendimiento final del sistema. Principalmente en cuanto a la capacidad de procesamiento que se tenia en los microcontroladores y en la eficiencia de la comunicación, ya que únicamente se obtuvo un rendimiento de dos envíos de datos por segundo, lo cual no permitía obtener un buen rendimiento.

Primero se definieron los requerimientos tanto de la aplicación y de los agentes, los cuales son mostrados en la siguientes listas y según a estos se estableció cual es la mejor tecnología a implementar, tanto para la comunicación, procesamiento de datos, interfaz gráfica y le bootloader. Buscando siempre tener un entorno amigable entre la comunicación entre los agentes y la aplicación.

Aplicación

- Capacidad de implementar multithreading.
- Capacidad para implementar una interfaz gráfica.
- Capacidad para realizar el análisis de visión por computadora.
- Capacidad de conectarse y comunicarse por medio de una red wifi.
- Alta capacidad de procesamiento numérico.

Agentes

- Alta capacidad de procesamiento de datos
- Capacidad de conectarse y comunicarse por medio de una red WIFI.
- Capacidad de poder modificar su archivo de trabajo (Bootloader).
- Bajo consumo energético.
- Variedad de módulos y GPIOs para implementar los módulos necesarios.

Según a los requerimientos y recomendaciones mencionadas anteriormente, se definió que la mejor opción para la aplicación era desarrollarla en python. Ya que python cuenta con varias librerías para el desarrollo de una interfaz gráfica, dentro de las cuales se seleccionó Tkinter. De igual forma cuenta con una gran capacidad de procesamiento de datos y soporte en la librería OpenCV, por medio de la cual se realizaría el análisis de visión de computadora, proceso descrito en la tesis de José Molina, titulada *Diseño e Implementación del Controlador Punto a Punto del Sistema Robotat por Medio de Visión de Computadora y Fusión de Sensores*. También cuenta con soporte para la convección y comunicación por medio de wifi y un módulo de multithreading, este era uno de los requerimientos y recomendaciones más importantes ya que esto ayudaría a mejorar el tiempo de ejecución del programa.

Figura 18: Requerimientos aplicación.

En cuanto a los microcontroladores, se definió que la mejor opción era utilizar el ESP8266 (ESP 12e). Ya que este microcontrolador cuenta con un módulo de comunicación wifi interno el cual tiene comunicación directa con el microcontrolador, esto nos ayuda a mejora bastante en comparación a la fase anterior, ya que evitamos realizar una etapa de comunicación extra entre el módulo de comunicación wifi y el microcontrolador, estando ambos en diferentes lenguajes de programación, lo cual complicaba la comunicación y tenía un impacto en el rendimiento del sistema. La ventaja principal que tiene este microcontrolador es que cuenta con la posibilidad de utilizar el firmware de Micropython, el cual, como su nombre lo indica está basado en python. Esto nos permite trabajar únicamente con este lenguaje tanto en la aplicación, como los agentes facilitando así la comunicación entre ambas partes.

Figura 19: Requerimientos microcontrolador.

Estos microcontroladores cuentan con una mayor capacidad de procesamiento a los utilizados en la etapa anterior y al utilizad micropython, se pueden implementar varias librerías, como las de comunicación por medio de sokets, comunicación por medio de MQTT. Otra ventaja importante con la que cuenta micropython es su sistema de archivos internos, el cual nos permite ejecutar, modificar, eliminar y almacenar varios archivos dentro del mismo microcontrolador, lo cual ofrece una gran ventaja sobre el PIC utilizado en la fase anterior y facilita el poder cargarle un archivo externo.

9.2. Interfaz gráfica

La aplicación se estructuro en base a procesos, por medio de los cuales el usuario puede ejecutar acciones a lo largo de la simulación del sistema. Para lograr este enlace procesos - interfaz gráfica, luego de que se definieron los procesos e información sobre la cual el usuario podrá tener control, se plantearon los objetos, componentes y estructura de la GUI. Ya que la aplicación también cuenta con información y procesos, a los cuales el usuario no debe poder acceder, fue necesario organizar todos procesos con los que cuenta la aplicación y enlazarlos de una forma eficiente, para que el uso del sistema sea amigable con la forma en la que el usuario interactúa con la aplicación desde la interfaz gráfica. A continuación se muestran los procesos principales que se llevan a cabo durante la ejecución de la aplicación y como estos están enlazados con los procesos que maneja el usuario de una forma fluida.

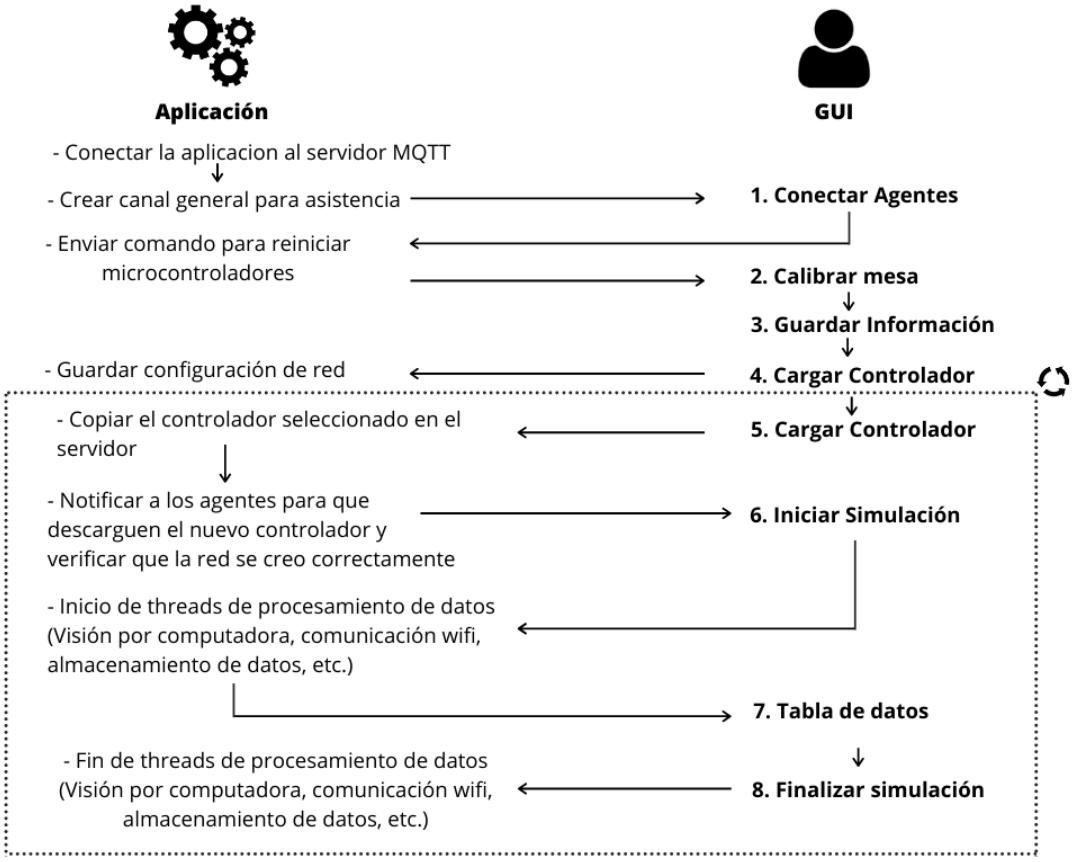

Figura 20: Flujo de procesos de la aplicación.

Para poder obtener un uso fluido de la aplicación de parte del usuario, fue necesario implementar varias restricciones y validaciones respecto al uso de las herramientas que se le proporcionan al usuario. Como ejemplo, podemos observar en la Figura 20, el usuario tiene control sobre el proceso de calibración de la mesa, ya que es necesario realizarlo manualmente antes de iniciar la simulación. Para esto fue necesario restringirle el uso de cualquier otro proceso al usuario hasta que realice la calibración de la mesa de pruebas. Según análisis anteriores se decidió qué información y cómo se le desplegaría esta al usuario, tomando en cuenta los objetos, vistas y procesos en los cuales se le mostraría dicha información.

9.2.1. Información relevante para el usuario

Al momento de la simulación la aplicación maneja bastante información, dentro de la cual hay datos que no son relevantes para el usuario, debido a esto es importante mostrarle únicamente la información relevante al usuario de forma concisa y fácil de interpretar. Según a la estructura y objetivo del sistema se decidió mostrarle al usuario los siguientes datos:

- ID del agente - relacionado con el tag físico del robot.
- Posición X, Y y ángulo respecto al marco de referencia general.
- Imagen en directo sin procesar del robotat.
- Imagen procesada con la información del agente seleccionado.

Luego de seleccionar la información importante para el usuario se seleccionaron los componentes ideales para dicho tipo de información, definiendo una tabla de datos para mostrar la pose e ID de cada robot y dos imágenes centrales para poder observar tanto el vídeo normal, como el vídeo procesado del robotat, obteniendo de está forma la vista de inicio de la aplicación, mostrada en la siguiente Figura 21.

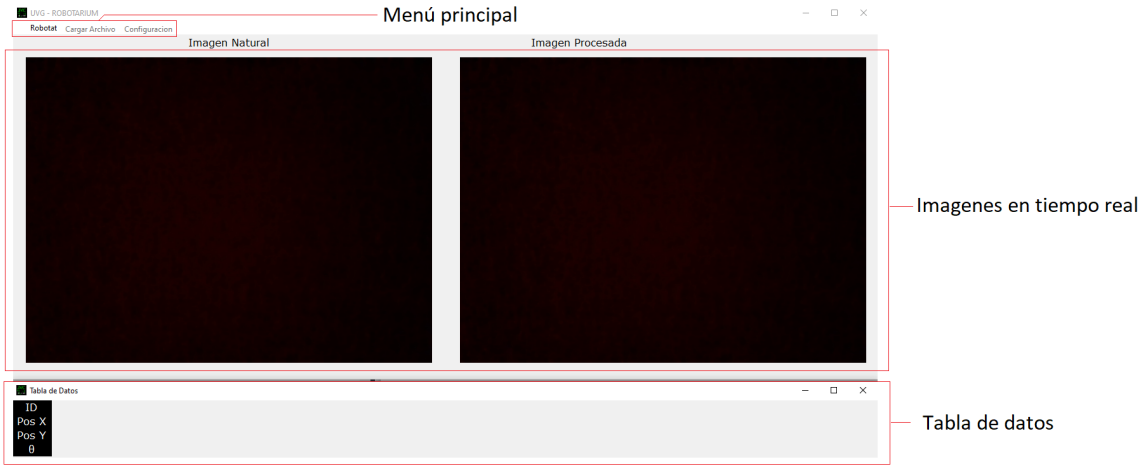

Figura 21: Vista de inicio de la aplicación.

9.3. Procesos de la aplicación

9.3.1. Configuración inicial

Cuando la aplicación inicia, se efectúan una serie de procesos internos, los cuales el usuario no puede visualizar y que se encargan de realizar todas las configuraciones iniciales del sistema. Dichos procesos se podrían definir el SET UP del programa.

- Declaración de variables
- Extraer controladores disponibles de la carpeta controladores
- Conexión al servidor MQTT y subscripción a tópico de asistencia
- Crear y mostrarle al usuario una instancia de la ventana principal

Al inicio la aplicación define todas las variables que se utilizaran durante la simulación, se obtiene todos los controladores guardados anteriormente por el usuario (para los agentes) y realiza la conexión al servidor MQTT, el cual se encuentra en la misma computadora, creando a su vez un objeto que cuenta con las propiedades de dicha conexión, el cual se estará utilizando a lo largo de toda la simulación para comunicarnos con los agentes. Durante este mismo paso, la aplicación se subscribe al tópico de asistencia, en el cual los agentes se estarán reportando a la aplicación cuando se enciendan. De igual forma se define la función de callback, la cual se ejecutará cuando llegue un mensaje proveniente de cualquier tópico por parte de los agentes, está función nos permitirá analizar y tomar decisiones sobre las acciones a ejecutar en base al mensaje entrante por parte de los agentes. Por último se crea una instancia de la ventana principal de la aplicación, la cual es desplegada al usuario para permitirle interactuar con el sistema, el funcionamiento de está ventana principal es explicada más a fondo en el capítulo "Interfaz gráfica". A continuación, se detalla el funcionamiento y orden en el cual se ejecutan los procesos durante la interacción usuario - aplicación.

Pseudocódigo

- Definición de variables
- Obtener el nombre de todos los archivos dentro de la carpeta controladores
- Crear el objeto que cuenta con las propiedades de la comunicación al servidor MQTT
  - Nos conectamos al servidor MQTT
  - Nos subscribimos al tópicos general de asistencia
  - Definimos la función callback para mensajes de entrada
  - Iniciamos el loop de comunicación
- Instanciamos la ventana principal de la aplicación

9.4. Menú principal - Robotat

Figura 22: Menú - Robotat.

Para mejorar la experiencia de uso del usuario, todos los procesos que el debe de realizar durante al simulación se encuentran enumerados en orden en este menú principal - Robotat. Con esto logramos obtener una interfaz gráfica fácil de utilizar e intuitiva para que el usuario efectué todos los procesos que se describen a contención.

1. Conectar a los agentes

Este proceso es el primero que se le presenta al usuario. Por este medio, la aplicación se comunica con los agentes utilizando el tópico general de comandos, enviándoles un comando para reiniciarlos. Al reiniciar a los agentes logramos que estos se vuelvan a reportar a la aplicación, obteniendo así el listado completo de agentes en uso durante la simulación. Este proceso está pensado para cuando se ingrese un nuevo agente al sistema o algún agente presente problemas internos, el usuario no tenga que reiniciar manualmente a los agentes o la aplicación, tomando en cuenta que al ser varios agentes el proceso puede ser tedioso e ineficiente. De esta forma logramos también que el usuario interactué lo menos posible con el código interno tanto de la aplicación, como de los agentes y lo haga por medio de la aplicación de una forma segura.

Pseudocódigo aplicación

- Estando subscritos al tópico general de asistencia (paso anterior)
- Publicamos 'Cerrar' en el tópico general para reiniciar a los agentes
- Por medio de la función callback obtenemos el ID del agente cuando estos se reporten
- Creamos un nuevo objeto para dicho agente

Pseudocódigo agentes

- Estando subscritos al tópico general (para obtener los comandos de la aplicación)
- Con el comando 'Cerrar' restablecemos al agente a su configuración por defecto
- Reiniciamos el microcontrolador
- Reportamos nuestro ID al tópico de asistencia

2. Calibración de la mesa

Este proceso se encarga de ejecutar el algoritmo de calibración de la mesa realizado por José Molina. Esta calibración se realiza con el objetivo de trabajar con una imagen 'recta' de la mesa de trabajo, ya que la cámara puede no estar alineada de forma perpendicular a la mesa, lo cual generaría errores al momento de calcular la pose de cada uno de los agentes. Es importante resaltar que este proceso únicamente se debe de realizar una vez al inicio de la sesión de trabajo o si la cámara es movida de lugar durante la simulación, lo cual no es recomendado.

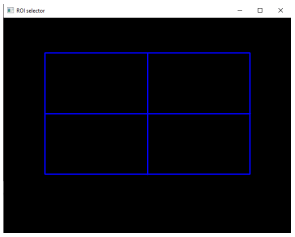

Figura 23: Calibración de la mesa.

3. Guardar información

En este proceso el usuario selecciona que información es la que desea guardar durante el proceso de simulación. Puede seleccionar si desea guardar el historial de la pose de cada uno de los agentes, al igual que generar una gráfica al final de la simulación, con la trayectoria de cada agente a lo largo de la simulación. Esto con el objetivo de poder tener material de análisis sobre el comportamiento y funcionamiento del algoritmo swarm ejecutado. La configuración seleccionada por el usuario es validada al momento de generar threads de almacenamiento de datos durante la simulación.

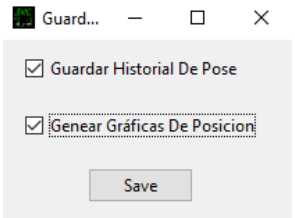

Figura 24: Guardar información.

4. Diseñar red de conexión

Figura 25: Menú para configurar la topología de la red de comunicación.

Este es uno de los pasos más importantes a realizar, ya que aquí es donde el usuario configura la forma en que se dará el flujo de información y la estructura de la red de comunicación entre los agentes y la aplicación. Por medio de este proceso el usuario puede seleccionar alguna de las dos estructuras de comunicación predeterminadas y/o también tiene la opción de diseñar la red de comunicación que el desea con una gran flexibilidad, ya que el puede definir los canales de comunicación y la información que fluye a través de dichos canales, como se muestra a continuación. Esta estructura de comunicación es explicada a fondo en el capítulo "Software de comunicacion".

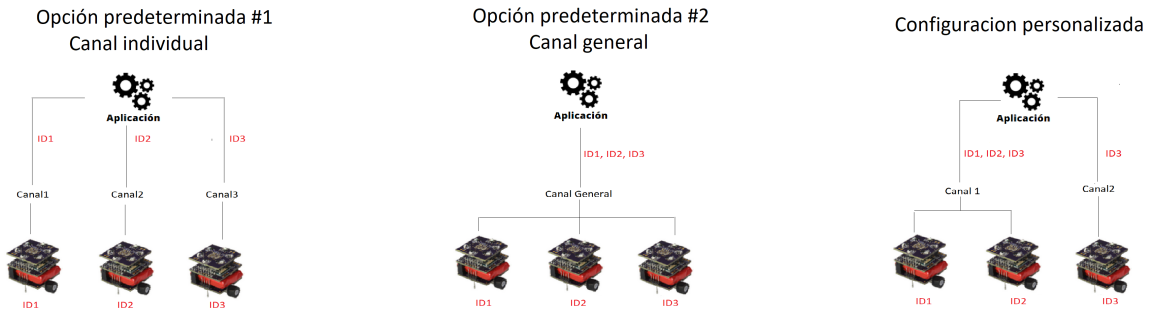

Figura 26: Flexibilidad en el diseño de la red de comunicación.

Al efectúala el proceso anterior se activa un proceso interno, el cual se encarga de guardar la configuración del diseño de red que el usuario acaba de realizar. Ya que como su nombre lo indica, en ese proceso la red únicamente se diseña, no se crea, la creación de la red se efectúa más adelante en conjunto con los agentes.

5. Cargar controlador - Bootloader

El proceso de carga del controlador se da en el paso cinco en el uso de la aplicación, en este paso se carga el nuevo archivo a cada uno de los agentes activos en el Robotarium. Para cumplir este objetivo de una forma eficiente y escalable, el proceso de descarga del archivo no se centraliza en la aplicación, si no que en cada uno de los agentes de forma individual y la aplicación únicamente funciona como el coordinador de las acciones a realizar. De esta forma logramos obtener un mejor tiempo de descarga general ya que todos los agentes realizan la descarga del archivo al mismo tiempo y no en serie como resultaría al centralizarlo en la aplicación. Para poder cargar un nuevo archivo en todos los agentes se creó un servidor web en la misma computadora donde se encuentra la aplicación. Dicho servidor se implementó por medio del entorno de desarrollo web **wamp** el cual funciona como centro de alojamiento del nuevo archivo a cargar a los agentes. Tanto la programación de la aplicación, como la de los microcontroladores se desarrolló y se estructuró en conjunto para poder realizar este proceso.

Es importante resaltar que micropython cuenta con un sistema de archivos interno, por medio del cual se pueden ejecutar y administrar los archivos deseados por el usuario. Por defecto al encender el microcontrolador, micropython ejecutara primero el archivo llamado ”boot.py”, por medio del cual nos conectamos a la red Wifi y luego ejecutara el archivo ”main.py”, en el cual cada gente reporta su ID a la aplicación. El objetivo de este bootloader, es poder descargar el archivo del servidor y colocarlo como el nuevo ”main.py” para que el controlador se empiece a ejecutar luego de conectarnos a la red wifi. También es importante tomar en cuenta que al momento de terminar el uso de la aplicación, se tiene que eliminar el controlador y restablecer el archivo ”main.py” original, para que al momento de iniciar nuevamente el agente, este se reporte a la aplicación correctamente.

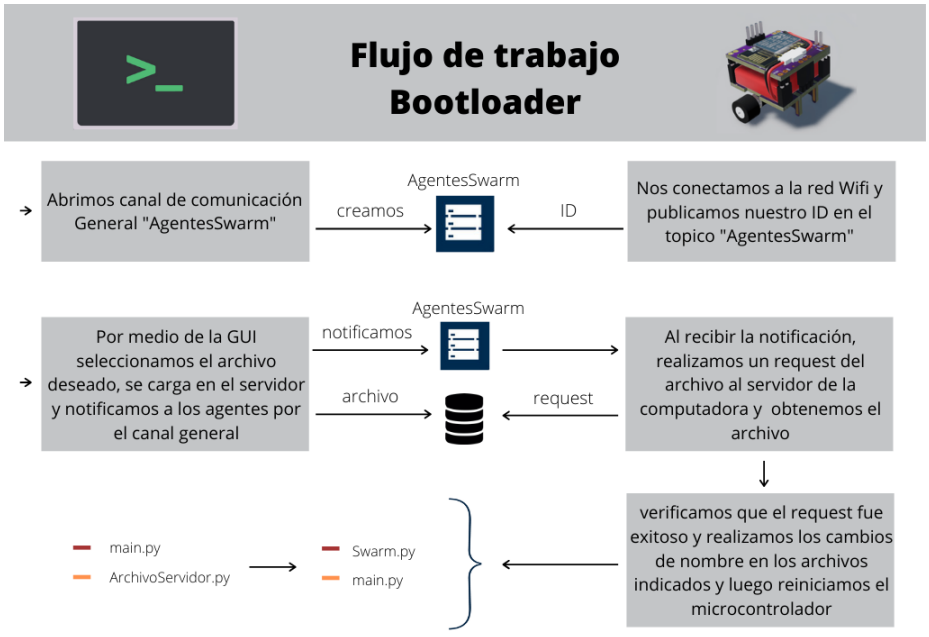

Figura 27: Proceso - Bootloader.

Canal de comunicación

Para poder comunicar la aplicación con cada uno de los agentes y de esta forma enviarles los comandos de carga de forma eficiente, se estableció una red de comunicación por medio del protocolo MQTT, utilizando el servidor Mosquitto. Como podemos observar en la Figura 27, al iniciar la aplicación, está crea un tópico general llamado ”AgentesSwarm” por medio del cual se envían los comandos generales a todos los agentes activos en el sistema, está configuración se explica más a detalle en el capítulo ”Red de comunicación”.

Bootloader - Aplicación

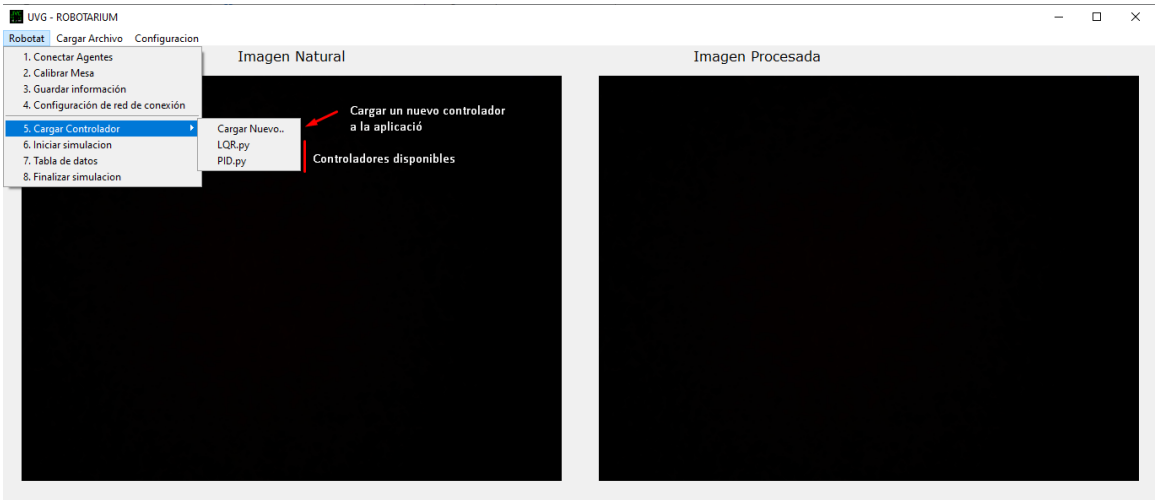

Figura 28: Interfaz para cargar controlador.

Para realizar el proceso de carga de un nuevo archivo, el usuario debe de seleccionar en la aplicación que archivo desea cargar a los agentes en el proceso **5. Cargar controlador**. Para esto, al momento de iniciar la aplicación se obtienen los nombres de todos los controladores existentes dentro de una carpeta llamada ”controladores”, como se explicó en el paso de configuración inicial. Dichos archivos son los controladores creados y almacenados anteriormente en la aplicación, los cuales son mostrados por medio de una lista como se puede observar en la Figura 28.

Cuando se selecciona un controlador, por medio de las librerías *shutil* y *os* de python, se crea una copia del archivo del controlador seleccionado, esta copia es cargada en el servidor wamp en la misma dirección URL a la cual los agentes realizaran la petición para descargar dicho archivo. Al terminar de copiar el archivo en el servidor, se envía el mensaje ”ArchivoServidor” al tópico general para notificarle a los agentes que el archivo ya se encuentra disponible en el servidor y realicen la descarga del archivo. Por último la aplicación se subscribe a todos los tópicos que el usuario diseño en la red, esto con el objetivo de confirmar que la red se creó correctamente, más información en el capítulo ”software de comunicacion”.

Para que el usuario tenga confirmación visual de que el proceso anterior se realizó con éxito, se despliega un mensaje en la pantalla. Es importante resaltar que en este punto el archivo únicamente se encuentra en el servidor, los agentes todavía no han realizado la descarga de este archivo.

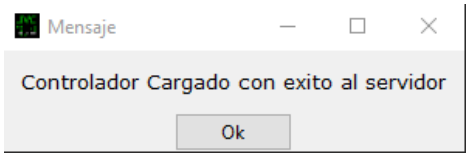

Figura 29: Confirmación de que el archivo está en el servidor.

Bootloader - Agentes

Al encender a cada uno de los agentes estos se conectan al tópico general "AgenetsSwarm" y "AsistenciaAgentes" a la espera de comandos por parte de la aplicación y para reportar su ID a la aplicación respectivamente. Al recibir el comando "ArchivoServidor", el agente por medio de las librerías *network* y *socket* de micropython realiza una petición al servidor con la URL del nuevo archivo. Dicha petición se lleva a cambo por medio de la función **Request Archivo Servidor()** en el archivo core.py en el microcontrolador.

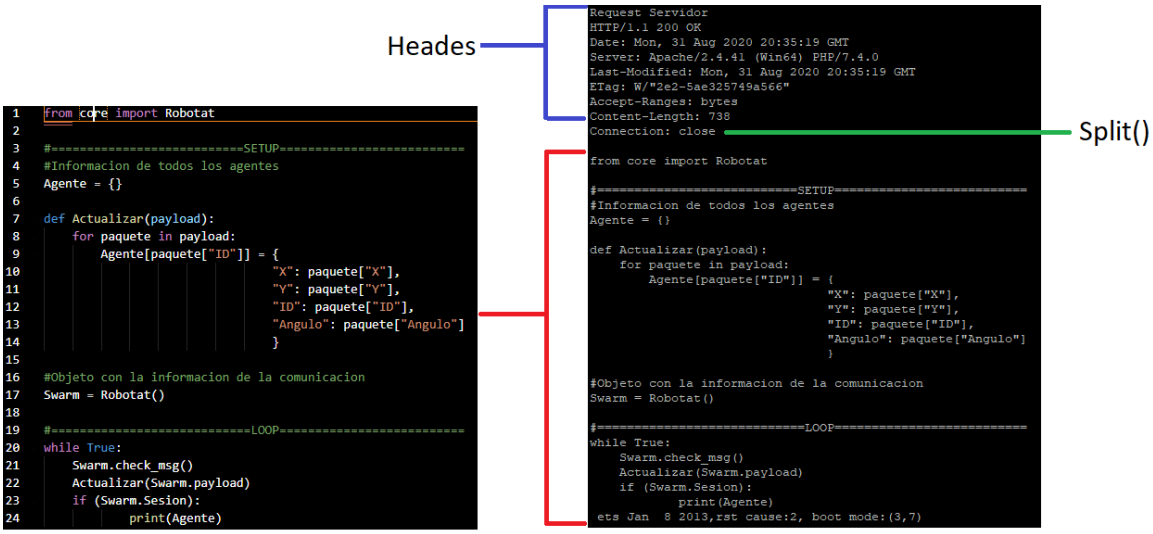

Figura 30: Software comunicación - Aplicación.

Cuando se realiza la petición del archivo al servidor, este nos envía como respuesta un paquete de datos que cuenta con headers y el contenido solicitado. Todo este paquete es leído por de los agentes en paquetes de 100 bytes y almacenado en una variable. Dentro de los headers enviados por el servidor se encuentra el código de respuesta de dicho request. Para poder extraer únicamente el contenido del archivo, se realizar un split() del contenido, luego de cerrar la conexión como podemos observar en la Figura 30. Al realizar esto logramos obtener todo el contenido (Código) del nuevo archivo en una variable, al terminar este proceso se crea un archivo nuevo dentro del microcontrolador llamado "ArchivoServidor.py", en el cual se almacena el código del nuevo controlador. Cuando todo este proceso se realizó de forma exitosa, se le notifica al usuario que ya todos los agentes cuentan con el nuevo controlador y que la red que el usuario diseño se creó de forma exitosa por medio del siguiente mensaje.

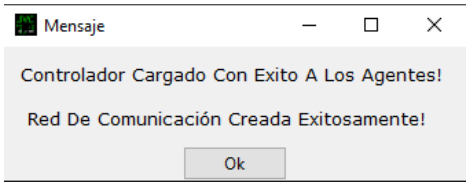

Figura 31: Confirmación de carga exitosa del programa.

Por último se renombra el archivo "main.py" a "Swarm.py" y el archivo "ArchivoServidor.py" como "main.py" y se reinicia el microcontrolador. Con esto logramos que cuando el microcontrolador inicie un nuevo ciclo de trabajo, el archivo principal de trabajo sea el archivo descargado en el servidor (controlador), el cual se empezara a ejecutar automáticamente luego del archivo boot.py. También almacenamos el archivo "main.py" por medio del cual se ejecutan las funciones para reportar el agente a la aplicación al encender el microcontrolador por primera vez. Es importante que este archivo sea el principal al momento de iniciar a los agentes por primera vez, ya que este es el único archivo por medio del cual los agentes se reportan, por eso al finaliza el uso de la aplicación se elimina el controlador y se restablece este archivo como el principal. Es importante mencionar que este proceso de cargar un nuevo controlador al agente se puede realizar múltiples veces durante el uso del sistema, ya que el microcontrolador sigue conectado al tópico general a al espera de comandos.

Pseudocódigo de los agentes

- El agente se subscribe al canal general de comandos
- La aplicación manda el comando para descargar el archivo
  - Realizamos la petición del archivo al servidor
  - Obtenemos la respuesta del servidor
  - Extraemos la información del archivo (código)
  - Almacenamos esa información en el archivo "ArchivoServidor.py"
- Realizamos el cambio de nombres en los archivos
- Reiniciamos el microcontrolador para ejecutar el controlador

Wamp

Como ya se indicó, para poder almacenar y descargar el archivo desde los agentes, se utilizó un servidor Web instalado en la misma computadora en donde se encuentra la aplicación. Este servidor crea una carpeta en la computadora (WWW), en la cual se deben de colocar todos los archivos con los que se trabajada en cada proyecto para que wamp los utilice como se muestra en la Figura 32. Por motivos de seguridad wamp no efectúa las peticiones externas a las realizadas por la computadora, por lo que las peticiones de los microcontroladores no se efectuaran (a pesar que estos se encuentran conectados a la misma red).

Figura 32: Proyecto Wamp.

Para evitar esto, fue necesario configurar wamp para que permitiera ser accesado desde otra computadora y así lograr ejecutar las peticiones de los agentes al momento de querer descargar los archivos. Para esto, se cambió la configuración de 'local' a 'all granted' como se muestra en la Figura 33. Al guardar está configuración las peticiones de los agentes serán ejecutadas por wamp y de está forma se podrá descargar el nuevo controlador a cargar a los agentes.

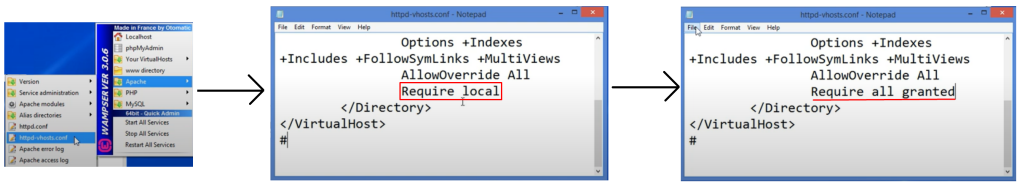

Figura 33: Cambio configuración wamp.

Como podemos observar este nuevo proceso de carga de archivos de forma inalámbrica nos permite obtener una flexibilidad mucho mayor sobre el uso del sistema comparado con la versión anterior, ya que esto nos permite realizar las simulaciones de una forma mucho más eficientes, tanto en tiempo, como en la facilidad de realizar las pruebas, ya que no es necesario que el usuario cambie de forma manual el código de cada uno de los agentes. Este nuevo proceso nos permite realizarlo de una forma global y escalable gracias a que el servidor wamp tiene la capacidad de ejecutar todas las peticiones de los agentes. De igual forma se tiene la base para poder modificar cualquier archivo o incluso todos los archivos con los que cuenta el microcontrolador al mismo tiempo, esto nos permite realizar grandes actualizaciones en el software de los agentes de forma inalámbrica, eficiente y escalable.

6. Iniciar simulación

Cuando el usuario inicia la simulación del algoritmo, la aplicación se desubscribe de los tópicos a los que se subscribió en el paso anterior, ya que en estos tópicos es donde estará enviando la información y estar subscrito en los mismos tópicos donde se está enviando la información generaría que la función callback de la aplicación se esté ejecutando cada envió de datos, lo cual sería ineficiente para el rendimiento del sistema. Luego de esto la aplicación genera una nueva thread por cada canal de comunicación, en donde cada thread verifica que datos son los que debe enviar por su respectivo canal y los envía, como se muestra en la siguiente figura:

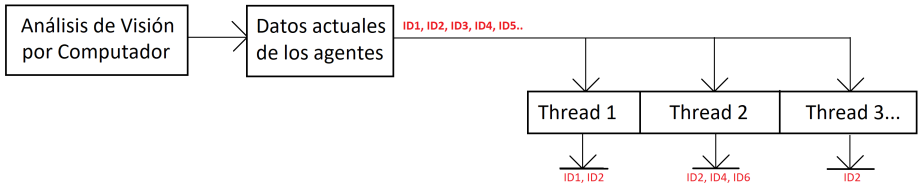

Figura 34: Generación de therads de comunicación.

Por último envía un comando a los agentes por el canal general de comandos, para que estos activen su sesión y de esta forma empezar a recibir los datos provenientes de la aplicación y ejecutar el controlador, este proceso se explica más a detalle en el capítulo "Software de comunicacion". Adicional a las threads de comunicación, la aplicación también inicia la ejecución de las threads de procesamiento y análisis de datos. estas estarían representadas por el primer cuadro de la Figura 34. Ya que sin estás nunca se actualizarían las poses estimadas de los agentes y siempre se estaría enviando la misma información, este proceso se detalla más en la tesis de José Molina. A continuación, se muestran los resultados obtenidos al momento de iniciar la simulación (envió de datos) desde la aplicación.

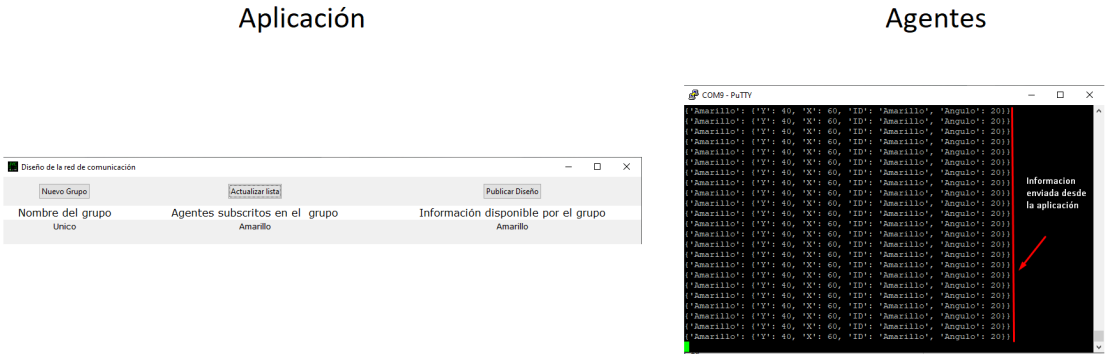

Figura 35: Inicio en la comunicación aplicación - agente.

Como podemos observar, al iniciar la simulación la aplicación se encarga de enviarlos datos correspondientes en cada uno de los canales definidos. Cada paquete de datos se encuentra en formato JSON, dentro del cual la información de cada agente está representada como un objeto que contiene la pose de dicho agente (Posición X, Y y ángulo). Todo este proceso se explica más a fondo en el capítulo "Software de comunicación".

7. Tabla de datos

Al seleccionar este proceso, al usuario se le despliega una nueva ventana con una tabla de datos que contiene la información actual de los agentes. Internamente un procesos se encarga de actualizar dicha información cada ciclo de ejecución de la aplicación.

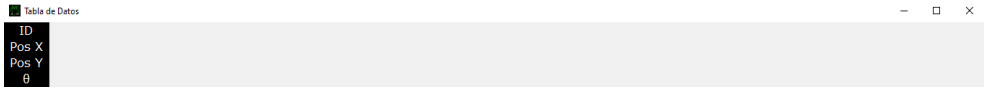

Figura 36: Tabla de datos - Cambiar por una con datos.

8. Finalizar simulación

En este proceso se envía un comando a los agentes para que detengan la sesión interna de cada uno y dejen de ejecutar el controlador. Este proceso es muy importante realizarlo al terminar la simulación del programa, ya que aquí es donde se paran de ejecutar las threads comunicación y procesamiento de datos, de lo contrario estas se seguirían ejecutando y consumiendo la capacidad de procesamiento de la computadora, teniendo un fuerte impacto en el rendimiento del sistema. Como precaución, este proceso también es ejecutado al momento de cerrar la ventana principal, para evitar que el usuario cierre la aplicación y las threads se sigan ejecutando mostrándole al usuario el siguiente mensaje visual para comprobar que todas las threds se terminaron de forma correcta.

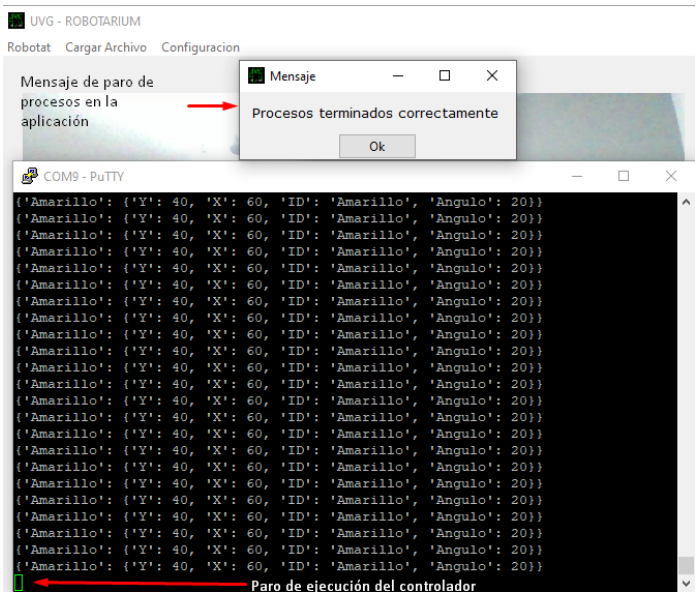

Figura 37: Fin de simulación.

Este proceso en conjunto con el de inicio de simulación, representan una gran ventaja para el usuario al momento de estar simulando el algoritmo, ya que él puede 'pausar' la simulación y analizar la situación actual físicamente o detener la simulación en caso no se esté realizando de la forma adecuada. De igual forma nos permite tener un proceso de programación y simulación más ordenado que en las fases anteriores del Robotat.

Procesos adicionales

Estos procesos adicionales no están relacionados con el funcionamiento del sistema directamente, pero fueron desarrollados para darle al usuario una mejor experiencia de uso al momento de que se presenten problemas durante la simulación o pueda evaluar mejor el rendimiento del sistema.

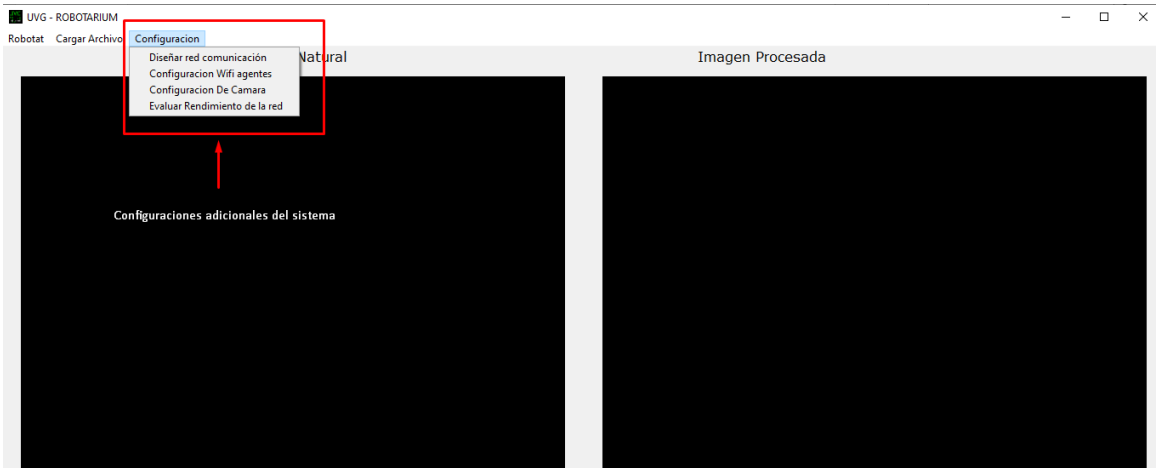

Figura 38: Menú de configuraciones adicionales.

Diseñar red de comunicación

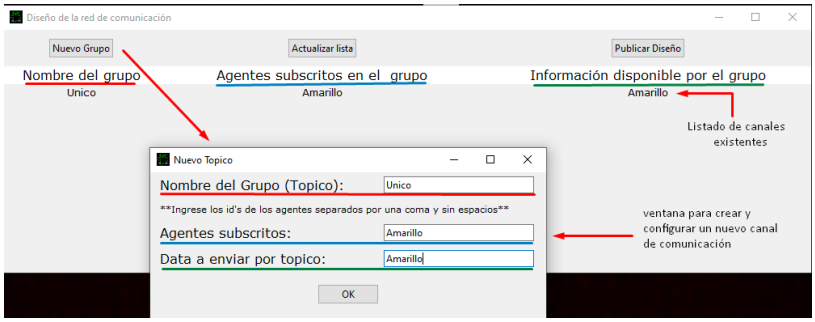

Figura 39: Interfaz para crear un canal de comunicación personalizado.

Por medio de esta ventana el usuario puede crear nuevos canales de comunicación personalizados, permitiéndole así explotar al máximo la flexibilidad del sistema y configurarlo de la forma necesaria para poder realizar algoritmos swarm que necesiten de dicha flexibilidad, este es uno de los avances más significativos con respecto a las fases anteriores ya que ahora el usuario tiene la opción y flexibilidad de poder diseñar diferentes topologías de comunicación de forma eficiente y no únicamente una comunicación directa con cada agente. En esta ventana el usuario también puede observar un listado con todos los canales (tópicos) generados hasta el momento para tener una representación visual y ordenada del esquema de la red creada. Por último, tiene la opción de 'Publicar el diseño', proceso en el cual le envía la estructura a los agentes para que estos la creen. Todo este proceso es descrito a profundidad en el capítulo 'Red de comunicación'.

Modificar configuración de red agentes

Por medio de este proceso el usuario puede modificar tanto la red, como la contraseña de la red a la cual se tiene que conectar los agentes. Este proceso se desarrollo pensando en el momento en que la red del Robotat cambie no se tenga que modificar manualmente el código de conexión de cada uno de los agentes para que esto se conecten a la nueva red, sino que por medio de este proceso se puede realizar de forma general.

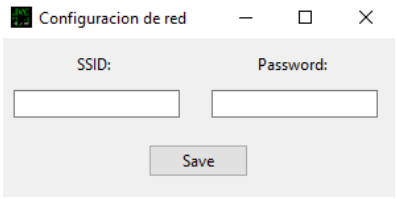

Figura 40: Configuración de red.

Configuración de cámara

Por este proceso el usuario define el puerto de entrado de la cámara. Debido a que este puede variar dependiendo de las condiciones del Robotat, se le permite al usuario seleccionar el puerto de entrada de la cámara y que de esta forma no tenga que interactuar directamente con el código de la aplicación para poder cambiar cambiar el puerto de entrada.

Diagrama de bloques

A continuación se describe por medio de un diagrama de bloques el flujo de ejecución que tiene el software de la aplicación. Todo el proceso mostrado en la Figura 41, es el descrito durante todo este capítulo. En el siguiente capítulo se describe el funcionamiento del software de los agentes y como es que este se relaciona con el de la aplicación.

Figura 41: Software comunicación - Aplicación.

CAPÍTULO 10

Agentes - Micropython

Como se mencionó en el capítulo anterior, en está fase del mega proyecto se decidió implementar un nuevo microcontrolador, el ESP8266, ya que luego de realizar un análisis de los requerimientos del sistema, este se definió como la mejor opción tanto por sus características de hardware, como de firmware. Como podemos observar en la Figura 42, el ESP8266 - ESP12e es un módulo Wi-Fi SOC (system on chip) el cual integra todos los módulos para la comunicación Wi-Fi, procesamiento, análisis de datos y hardware necesario para poder atender a los requerimientos del Robotat. La principal ventaja en cuanto al hardware que nos ofrece este módulo comparado con los utilizados en versiones anteriores en el Robotat, es que el módulo Wi-Fi cuenta con una conexión directa e interna con el microcontrolador, lo cual nos permite obtener los datos de forma directa, eliminando así esa etapa extra de comunicación UART entre módulo wifi y microcontrolador, la cual tenia un impacto en el rendimiento de los agentes en el sistema y complicaba la programación de los agentes.

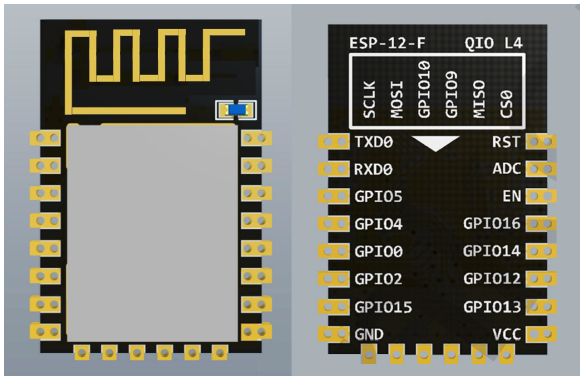

Figura 42: SOC ESP8266 - ESP12e.

Como podemos observar en el cuatro Comparativo 1, el ESP8266 presenta varias ventajas sobre el PIC utilizado anteriormente. Es importante mencionar que estas ventajas son importantes para poder cumplir con los requerimientos del sistema y con los objetivos de este trabajo. La principal ventaja que presenta el ESP8266 sobre el PIC16F88, implementado en la fase anterior, es la posibilidad de utilizar el firmware de Micropython. Esto nos permite obtener una gran ventaja al poder desarrollar el sistema completo sobre una misma plataforma basada en python, ya que tanto el software de la aplicación como los agentes estará desarrollados en este lenguaje. De esta forma obtenemos un flujo de información en el mismo formato con lo cual ya no es necesario estar cambiando el formato de la información entre cada uno de los pasos de comunicación como en la fase anterior.

	CPU	**Velocidad**	**Comunicación**	**Flash**
ESP8266	32-bits	hasta 160MHz	UART/IIC/PWM/ADC/SPI	4 M
PIC16F88	8-bits	hasta 20MHz	UART/SPI o I2C/ADC/PWM	7168 bytes
	Wi-Fi	**Lenguaje**	**Voltaje de trabajo**	**Empaquetado**
ESP8266	Si	Micropython	3.0V ∼3.6V, normalmente 3.3V	SMD
PIC16F88	No	C	2.0V ∼5.5V, normalmente 5V	THT y SMD

Cuadro 1: Tabla comparativa del microcontrolador de la fase anterior y el actual.

Micropython

Micropython es una implementación del lenguaje de programación python3 para microcontroladores, optimizado para poder trabajar con los recursos limitados que presenta en un microcontrolador y a su vez poder incluir librerías que permiten explotar dichos recursos de mejor forma que otros firmwares. La principal ventaja que presenta micropython sobre los otros firmares disponibles para el ESP8266, es su sistema de archivos interno, por medio del cual se pueden administrar varios archivos de trabajo dentro del microcontrolador, es decir, nos permite cargar y trabajar con varios archivos en el microcontrolador. Esto nos presenta una gran ventaja sobre las fases anteriores de este proyecto, ya que por medio de este sistema de archivos es posible realizar el bootloader y obtener un trabajo más ordenado y escalable en el código de los microcontroladores. En la siguiente figura, se puede observar dicho sistema de archivos en el microcontrolador, donde se pueden ver los tres archivos utilizados en el microcontrolador utilizando la librería os, por medio de la cual podemos trabajar con el sistema operativo interno del micropython.

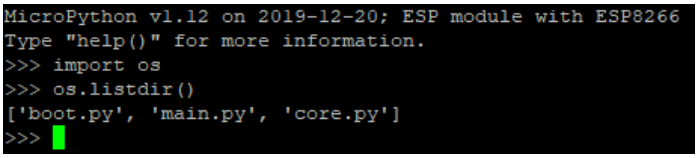

Figura 43: Sistema de archivos Micropython.

Herramientas de trabajo para micropython

Para poder trabajar con micropython en el ESP8266, es necesario instalar y utilizar dos herramientas de software por medio de las cuales podremos cargarle el firmware y administrar los archivos del microcontrolador desde la computadora.

Esptool

Para poder cargar el firmware de micropython al microcontrolador es necesario utilizar está herramienta de software desarrollada por espressif **esptool**. Por medio de la cual nos podemos comunicar directamente con el bootloader de la memoria ROM del microcontrolador y así poder guardar directamente el firmware, dicha herramienta está disponible en el repositorio de github de espressif. Para poder realizar el siguiente procesamiento, es necesario realizar un circuito por medio del cual colocamos al microcontrolador en modo de programación antes de ejecutar los siguientes comandos.

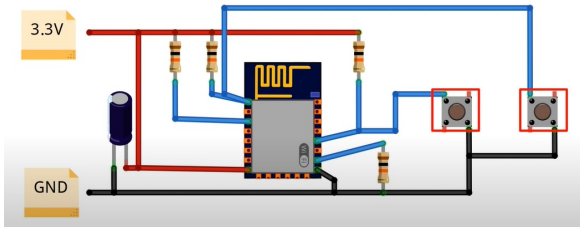

Figura 44: Circuito para reinicio y programación ESP8266.

Antes de poder cargarle el firmware es recomendado borrarle contenido que tiene la memoria como se muestra en la siguiente imagen, ya que de esta forma nos aseguramos de tener el espacio de la memoria completamente libre para cargarle el nuevo firmware, es importante mencionar que este procedimiento se debe de realizar desde la línea de comandos con la dirección donde se encuentra la carpeta de esptool indicando el puerto USB donde se encuentra conectado el microcontrolador.

» python esptool.py –port /dev/ttyUSBX $erase_flash \rightarrow Linux/Mac$
$>> pythonesptool.py --portCOMX \text{erase}_flash \rightarrow Windows$

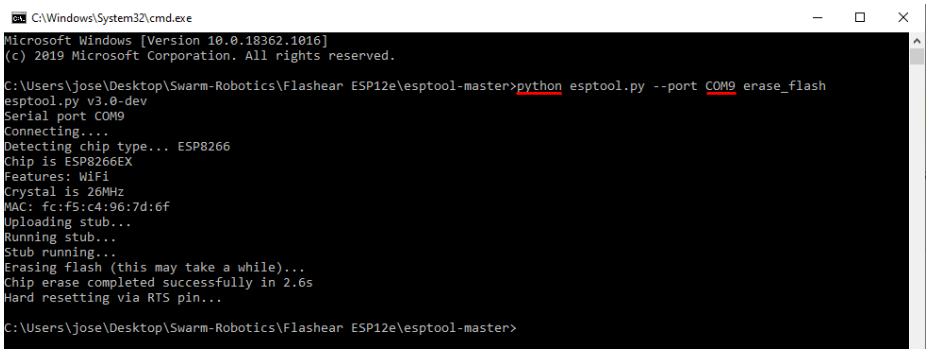

Figura 45: Borrar flash del ESP8266.

Luego de borrar el contenido de la memoria flash del microcontrolador, procedemos a cargarle el nuevo firmware. Para esto es necesario descargar la versión deseada de micropython de su página oficial, es recomendado utilizar una versión que ya se encuentre estable y no se esté actualizando diariamente, ya que así evitamos posibles problemás que se puedan presentar durante la ejecución del programa. Durante el desarrollo de este proyecto se utilizó la versión 1.12, como se observa a continuación, para cargarle dicho firmware es necesario ejecutar el siguiente comando desde la línea de comando con la dirección de la carpeta esptool y el firmware a cargar tiene que estár dentro de esta carpeta.

» python esptool.py –port /dev/ttyUSBX –baud 460800 $write_flash --flash_size = detect$ -fm dio 0 esp8266-20170108-v1.8.7.bin → $Linux/Mac$

>> $pythonesptool.py --portCOMX --baud460800write_flash --flash_size = detect$ -fm dio 0 esp8266-20170108-v1.8.7.bin → $Windows$

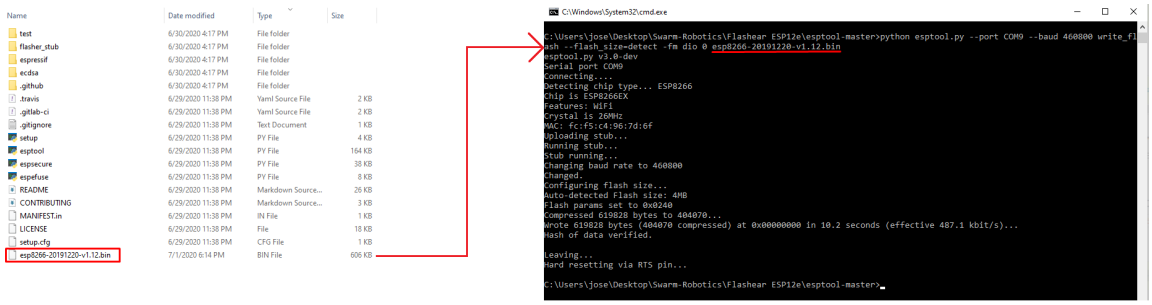

Figura 46: Cargar firmware de micropython al ESP8266.

Luego de haber instalado el software es necesario reiniciar el microcontrolador por medio del botón del circuito de la Figura 44. Para comprobar que todo se instaló correctamente se puede utilizar una herramienta como Putty, para comunicarnos con el microcontrolador. En la siguiente figura podemos observar la línea de comandos que nos despliega el microcontrolador al tener cargado micropython. De igual forma observamos el archivo boot.py que se instala por defecto, al obtener esto confirmamos que el firmware se instaló correctamente.

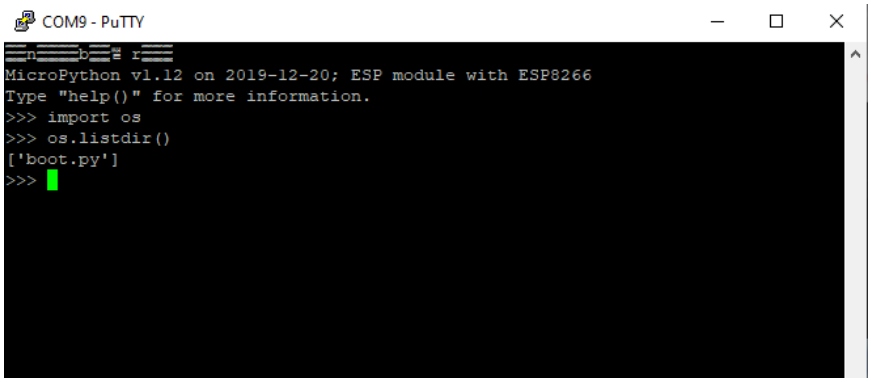

Figura 47: Software comunicación - Agente.

AMPY

Luego de tener instalado el firmware de micropython necesitamos poder cargarle los archivos de trabajo para que el microcontrolador los ejecute, para esto es necesario utilizar la herramienta AMPY (Adafruit MicroPython tool) la cual fue desarrollada por adafruit. Con esta herramienta podemos administrar los archivos en el microcontrolador por medio de un FTDI conectado al microcontrolador. Esta herramienta puede ser instalada desde la línea de comandos por medio del siguente comando.

»Pip install adafruit-ampy

Al tener instalado AMPY podemos ejecutar los siguientes comandos desde la linea de comando con la dirección en donde se encuentren los archivos con los que queremos trabajar. De esta forma podemos cargar, borrar, correr, etc. los archivos deseados en el microcontrolador, a continuación, se muestra el menú de ayuda de AMPY en el cual se observan todos los comandos con los que podemos trabajar.

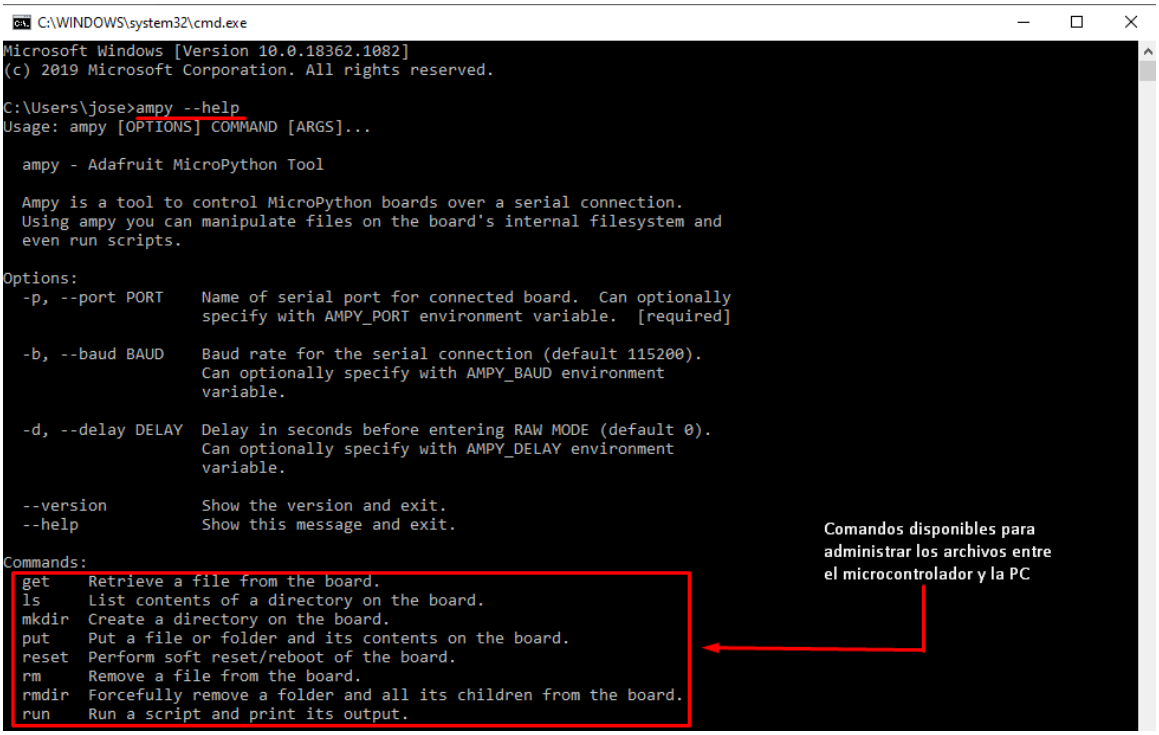

Figura 48: Software comunicación - Agente.

Software de los agentes

El software de los agentes trabaja en conjunto con el software de la aplicación por medio de comandos enviados desde está. Por medio de estos comandos los agentes ejecutan los procesos internos indicados por la aplicación, estos procesos internos son funciones que se encuentran en el archivo principal core.py, como se muestra a continuación en la Figura 49, cada agente inicia con tres archivos de trabajo:

- **boot.py**: Extracción de valores de la base de datos y conexión a la red Wi-Fi.
- **main.py**: Ejecutamos la función *ReportarAgente*() para reportar al agente a la aplicación.
- **core.py**: Aquí se encuentran todas las funciones para el funcionamiento del agente.

Estos archivos funcionan en el flujo descrito en la siguiente imagen:

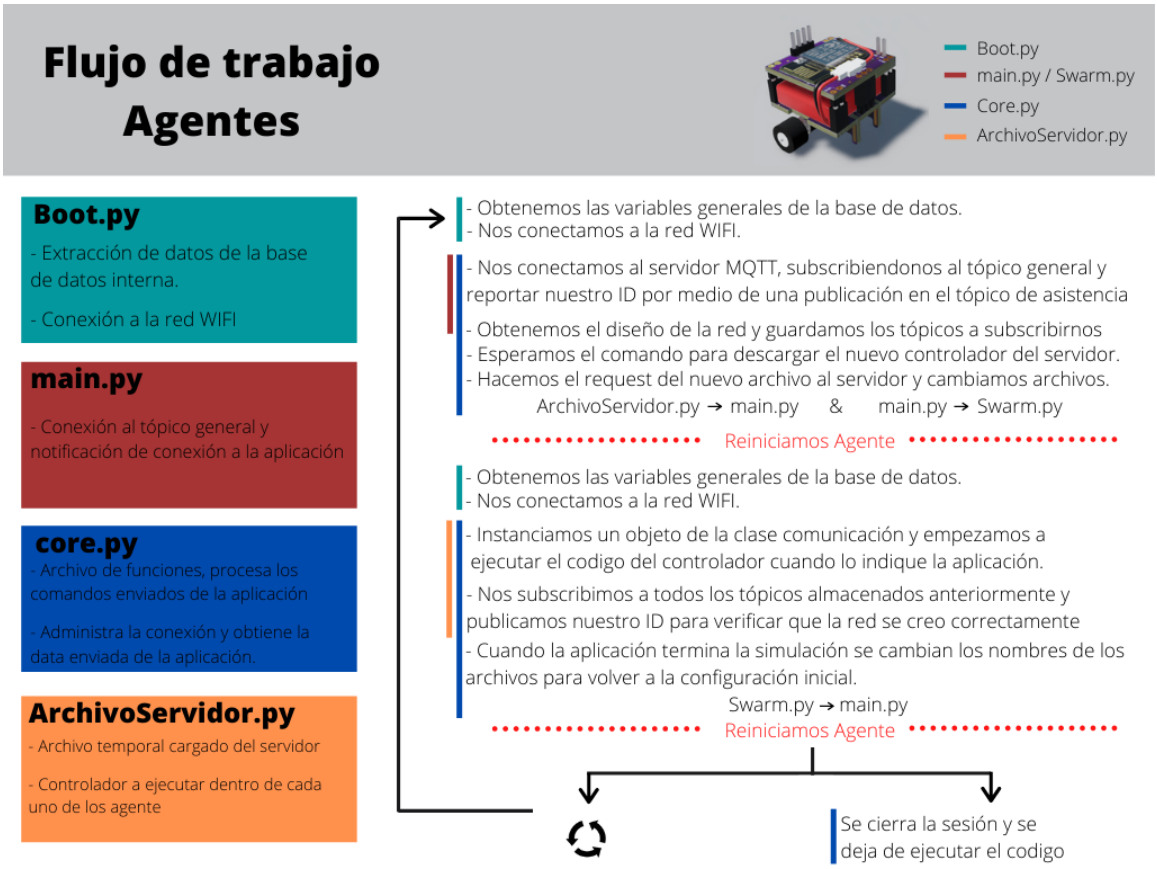

Figura 49: Software comunicación - Agente.

Como se puede observar en la imagen anterior el funcionamiento del agente depende únicamente del archivo principal (main.py) que se encuentre en ejecución, los cuales pueden ser el main.py o el controlador. De esta forma logramos ejecutar las funciones respectivas en cada una de las etapas de la simulación.

Cuando el agente realiza la petición del controlador al servidor, este lo almacena inicialmente como ArchivoServidor.py, para luego realizar el cambio de nombres por el main.py. En este punto es importante realizar un 'backup' del archivo main.py original como Swarm.py, ya que cuando la simulación termine se debe regresar dicho archivo como el main.py y eliminar el controlador. Este proceso es necesario ya que este archivo es el encargado de reportar al agente a la aplicación, si no se realizara el 'backup' de dicho archivo y se eliminara el controlador, el agente únicamente se quedaría con los archivos **boot.py** y **core.py** y ninguno de esos se encarga de reportar al agente a la aplicación.

Controlador

Toda la programación y estructura de los archivos del agente en micropython es diseñada para que el programa que el usuario le cargue a los agentes con el algoritmo de control sea lo más simple de desarrollar y implementar, es decir, que el usuario tenga a su disposición varias funciones que se encarguen de realizar los procesos de interacción con la aplicación, como la comunicación y recepción de datos. Esto se puede realizar gracias a que el microcontrolador está trabajando sobre una plataforma basada en python, por lo que el usuario puede importar funciones y variables de los otros archivos que se encuentran en el microcontrolador con gran facilidad, como por ejemplo el core.py como se muestra en la Figura 50. está figura muestra la estructura mínima necesaria de un archivo funcional al momento de cargarlo al agente por medio del bootloader.

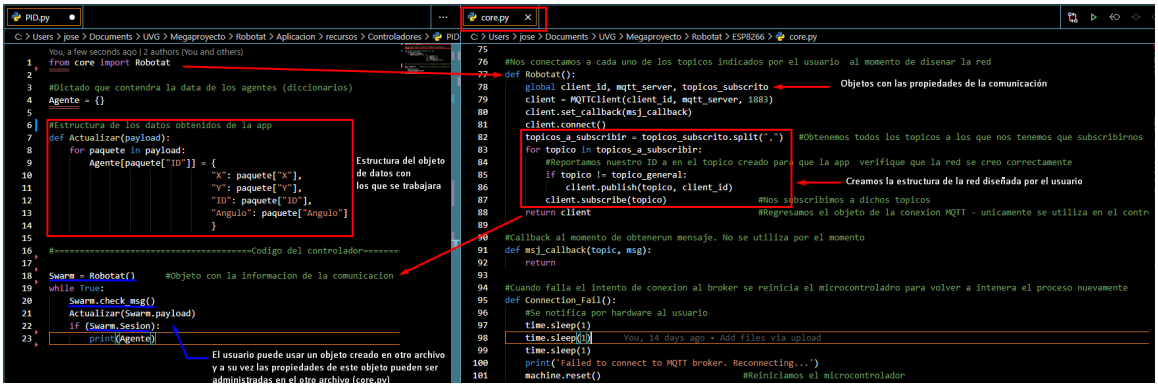

Figura 50: Software comunicación - Agente.

Este ejemplo únicamente se encarga de obtener los datos que la aplicación envía, generar un objeto por cada agente del cual obtenemos información y mostrar dicha información por el puerto serial. Como se puede observar, el usuario puede importar una función del core.py, la cual se encarga de subscribir al agente a todos los tópicos definidor por el usuario al momento de diseñar la red de comunicación y regresar el objeto de la comunicación, de esta forma el usuario tiene control sobre dicho objeto, por medio del cual, utilizando sus propiedades puede obtener la información de la app, el estado de la sesión, leer mensajes, etc. esta gran flexibilidad en el código del controlador nos ofrece varias ventajas sobre los planteamientos realizados en las fases anteriores del Robotat, donde únicamente se enviaba la velocidad de las llantas y los agentes no contaban con esta fase de procesamiento y análisis de datos que nos permite poder realizar de mejor forma los algoritmos swarm.

Esta arquitectura nos facilita bastante el proceso de generar un nuevo controlador ya que las funciones más importantes para la comunicación, procesamiento de datos, etc. se encuentran disponibles y el usuario únicamente las debe de utilizar y no programarlas lo cual también nos permite obtener un código más limpio en el controlador. También nos permite crear nuevas funciones/archivos para futuras interacciones con nuevas funciones, las cuales son igual de accesibles al usuario, por lo que se puede definir librerías de funciones para diferentes aplicaciones y únicamente sería necesario importar dichas funciones desde el controlador, lo cual le permite al usuario desarrollar controladores con gran facilidad y flexibilidad. También es importante resaltar que, esta arquitectura evita que el usuario tenga acceso o tenga que modificar dichas funciones principales, con lo cual se pueden evitar errores al momento del uso o tener varias versiones de la misma función.

Recepción de datos

Para la recepción de datos, como se mencionó anteriormente se definió una clase por medio de la cual se crea un objeto que cuenta con las propiedades necesarias, de las cuales las principales son las siguentes:

- *Clientid*: Es el identificador del agente dentro del servidor MQTT.
- **Sesión**: esta propiedad centa con el estado de la sesión de la simulación, es decir, cuando la simulación es iniciada desde la aplicación está propiedad es verdadera y cuando la simulación termina es falsa. Esto nos permite dejar de ejecutar el controlador cuando la simulación ya fue finalizada.
- **payload**: esta propiedad cuenta con el objeto de la información enviada por la aplicación en formato JSON, este objeto es el que se utiliza en la función Actualizar() del controlador, para extraer los datos recibidos.

La función **Actualizar()** utilizada en el controlador sirve para organizar la información recibida de los tópicos a los cuales el agente está subscrito y eliminar información duplicada, en el caso de que por dos tópicos el agente reciba la misma información. está función se decidió que se implementaría desde el controlador y no como una función del core.py, ya que así el usuario puede definir la estructura de datos que se recibirá. lo cual nos permite realizar cambios en la estructura de datos a trabajar rápidamente, esto nos permite poder realizar y ejecutar controladores que trabajen con diferentes estructuras de información, es decir, únicamente la posición en X/Y, la pose o únicamente el ángulo. Toda esta información es almacenada en el diccionario Agente, el cual cuenta con la infracción de todos los agentes de los cuales se ha recibido información en forma de objeto igualmente, como se muestra a continuación.

```
{'azul': {'x': 21, 'y': 45, 'angulo': 23, 'id': 'azul'}, 'amarillo': {'x': 40, 'y': 50, 'angulo': 50, 'id': 'amarillo'}}
```

Figura 51: Estructura de almacenamiento de datos en el controlador.

Como se puede observar el funcionamiento tanto de la aplicación, como de los agentes nos permite obtener una gran flexibilidad al momento de evaluar los algoritmos swarm, tanto en la etapa de programación, configuración como en la de simulación, ya que la aplicación trata de ser lo más parecida a un entorno de simulación como Webots, en el cual el usuario puede programar, cargar el controlador y pasar por diferentes etapas en la simulación, que le permiten extraer información valiosa. De igual forma gracias a la estructura en la cual funciona tanto la aplicación como los agentes podemos implementar nuevas etapas de procesamiento de datos, análisis, etc. en futuras iteraciones del Robotat, permitiéndonos así poder evaluar algoritmos más complejos tanto en cantidad de agentes como en la complejidad del algoritmo.

Con esta nueva iteración se solucionaron los problemás más relevantes obtenido en la fase anterior, los cuales llegaron a tener un gran impacto en el rendimiento del sistema y también se agregaron varios procesos extras que ayudan a mejorar la funcionalidad del Robotat. Dentro de lo cuales es importante resaltar la capacidad con la que cuentan los agentes para poder obtener y procesar información del resto de los agentes de la simulación, como se explicara a detalle en el capítulo "Red de comunicacion" y no solamente obtener la velocidad que este debe colocar en cada una de sus ruedas. De igual forma se implementó la recomendación de implementar multithreading del lado de la aplicación para realizar los diferentes procesos como análisis de visión por computadora, despliegue de datos, comunicación, etc. Ya que esto, como se muestra en el capítulo "Rendimiento del sistema" llego a tener un gran impacto en el rendimiento del sistema. Por último, la forma en que se estructuro el funcionamiento y la programación del sistema, permite la expansión del código para realizar una gran variedad de procesos adicionales para poder evaluar una gran variedad de algoritmos swarm de una forma ordenada y fácil para el usuario.

A continuación, se muestran dos imágenes que describen en forma de diagrama de bloques la interacción entre la aplicación y los agentes en las diferentes etapas de la solución, donde se puede apreciar el flujo del código y archivos del agente. También se observa la flexibilidad con la que cuenta la aplicación al momento de la simulación y como es que el usuario puede regresar a varias etapas de está al terminar una corrida, lo cual brida una mejor experiencia de uso al usuario. En el siguiente capítulo se explica a fondo la red de comunicación implementada en el Robotat, en la cual se definen todos los tópicos descritos en las imágenes.

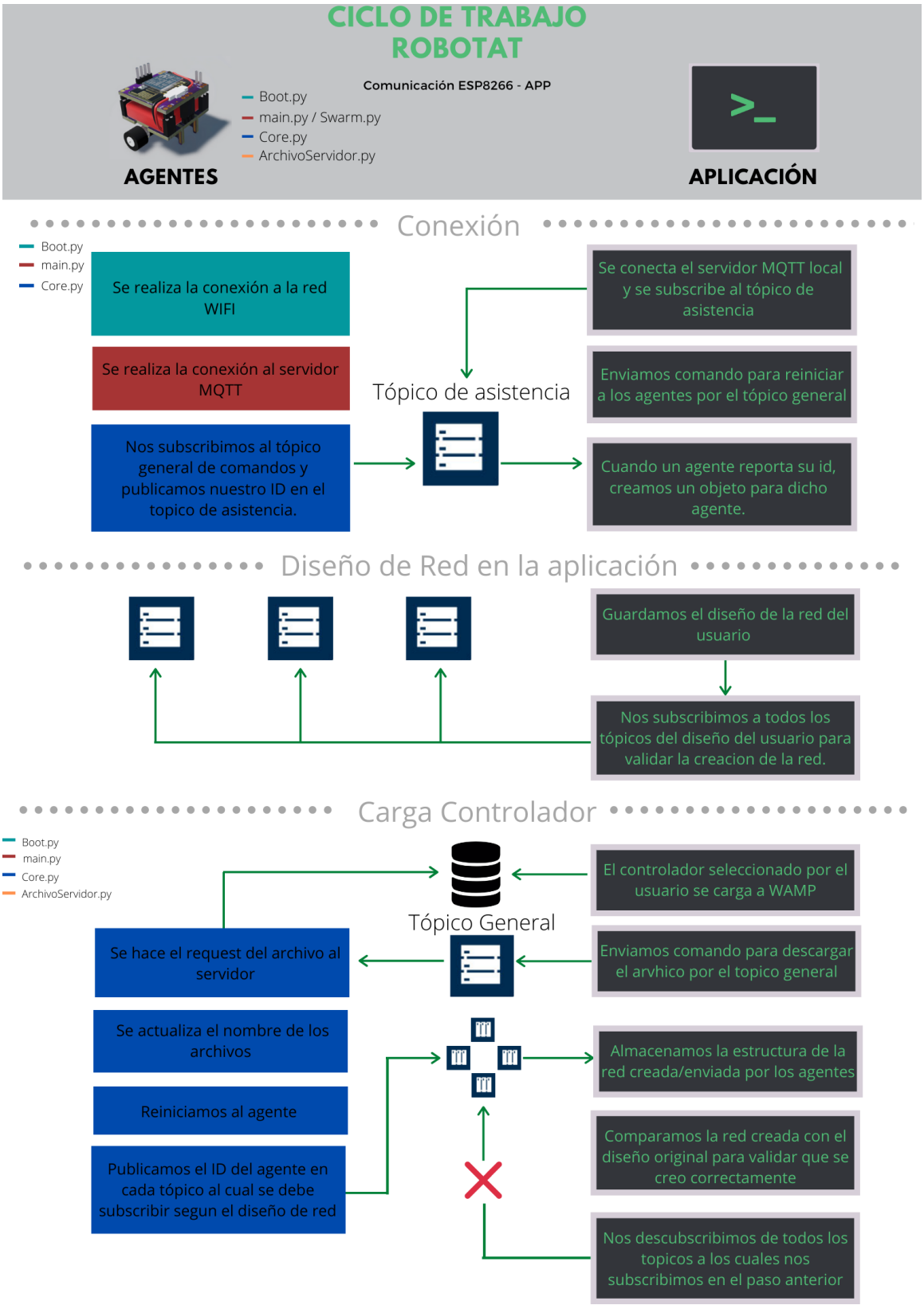

Figura 52: Ciclo de comunicación Robotarium.

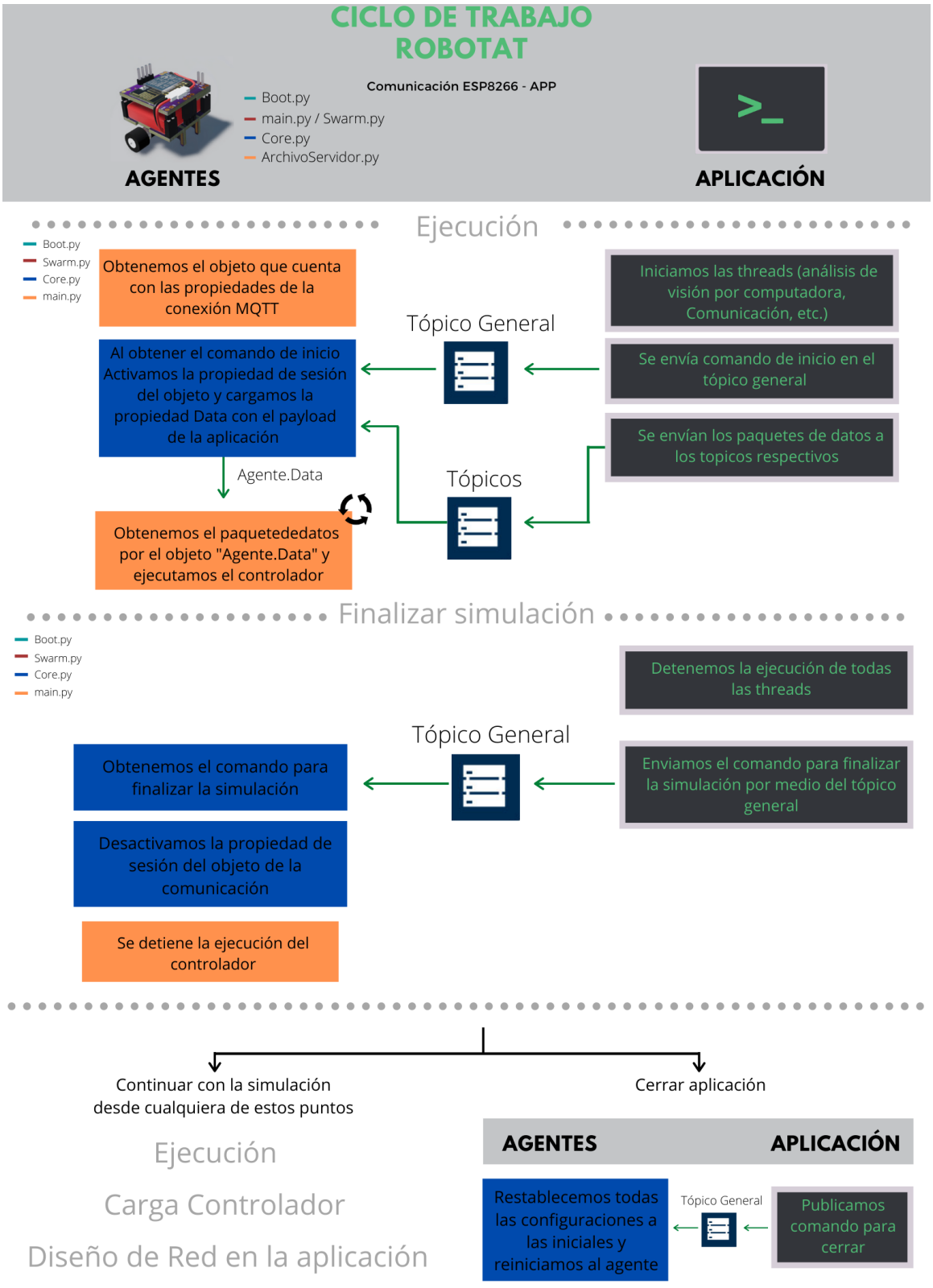

Figura 53: Ciclo de comunicación Robotarium.

CAPÍTULO 11

Red de comunicación

11.1. MQTT

Para realizar la red de comunicación dentro de la plataforma se implementó el protocolo de comunicación MQTT,el cual hoy en día está teniendo un gran impacto en el área de IOT. Este protocolo está diseñado para ser de ligero y funciona basado en una estructura de publicador-subscriptor por medio de un tópico, lo cual es ideal para enviar pequeños paquetes de información utilizando poco ancho de banda.

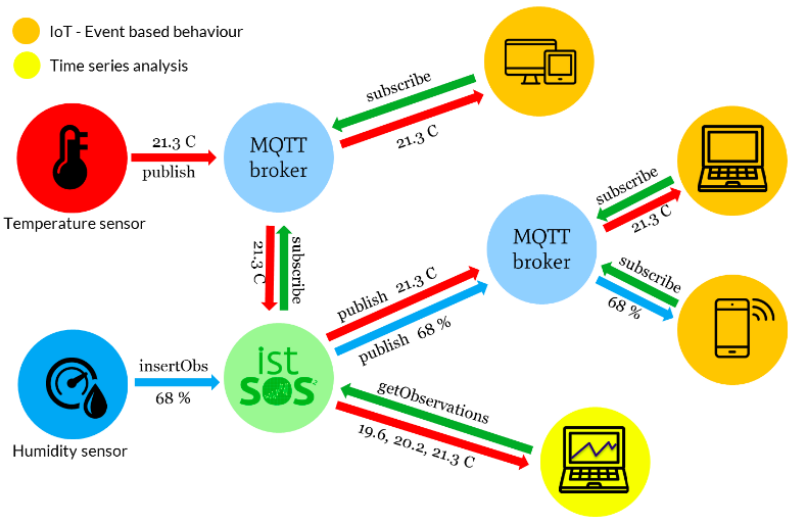

Figura 54: Esquema del protocolo MQTT.

Este protocolo funciona de una forma muy básica pero eficiente, por lo que se acopla muy bien para las necesidades del Robotat, ya que necesitamos una red de comunicación para enviar pequeños paquetes de información, que nos permita tener varios agentes conectados a esta red y que sea ligera. Al implementar este protocolo de comunicación nos desligamos de la dependencia de la cantidad de agentes en el sistema para tener un buen rendimiento, ya que está estructura no presenta una conexión directa entre la aplicación y cada uno de los agentes como en las fases anteriores del proyecto, si no que está comunicación se da por medio de los tópicos permitiéndonos obtener configuraciones en la topología de la red mucho más flexibles y con más agentes. Ahora la limitante de la red es la cantidad de tópicos que se crean y la cantidad de información que ser enviada por dicho tópico. Las tres definiciones principales para trabajar con este protocolo son las siguientes:

- **Tópico**: 'Canal' por medio del cual se da el traspaso de la información.
- **Publicador**: Dispositivo que envía un paquete de información a un tópico especifico.
- **Subscriptor**: Dispositivo que obtiene la información de todos los tópicos a los cuales está subscrito.

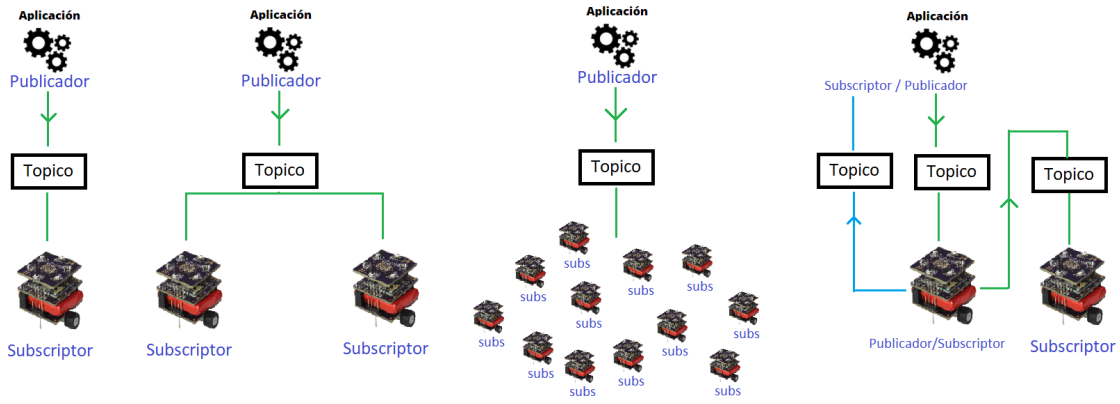

Figura 55: Flexibilidad del protocolo MQTT.

Utilizando ese protocolo se puede diseñar una topología donde la aplicación se define como un publicador y los agentes como subscriptores, ya que la aplicación es la encargada de enviar la información a los agentes y estos de recibirla como en las fases anteriores. Pero este protocolo también nos abre las puertas para poder implementar nuevas configuraciones donde un agente puede publicar información tanto a otros agentes como a la aplicación, lo cual nos permitiría realizar simulaciones más complejas donde los agentes también le pueden enviar información a la aplicación para que está la procese o incluso permitir la comunicación entre los mismos agentes como se explicara más adelante donde un dispositivo se define tanto como publicador y subscriptor. Con este protocolo logramos eliminar los problemas obtenidos en la fase anterior, en donde únicamente se tenía un canal directo de comunicación con el agente, por lo que al ir incrementando la cantidad de agentes decrementaba el rendimiento del sistema, con este protocolo logramos obtener una gran flexibilidad en la topología a diseñar, tanta que incluso el usuario puede diseñar la topología de comunicación que el desea utilizar durante la simulación.

Para crear la red de comunicación en el Robotat se diseñó el proceso **4. configurar red de comunicación** en la aplicación, por medio del cual el usuario puede diseñar la topología de la red que desea utilizar durante la comunicación. En este proceso se le presentan dos opciones al usuario, el puede seleccionar dentro de las opciones predefinidas o crear una nueva. A continuación, se explica cómo funciona cada una de estas al igual que la forma en que se crean por medio de los agentes.

11.2. Topología predeterminada

Dentro del proceso de configurar la red de comunicación se le presentan dos topologías al usuario, se definieron de esta forma ya que posiblemente sean las dos topologías más utilizadas en las simulaciones que se estarán llevando a cabo y de esta forma el usuario puede utilizar cualquiera de esas dos configuraciones de una forma sencilla por medio del menú mostrado en la Figura 25.

- **Grupo general**: Por medio de esta configuración se crea un tópico general por medio del cual se llevara a cabo toda la comunicación entre los agentes y la aplicación. Como se mencionó anteriormente en este caso la aplicación es la que publica la información de todos los agentes en el tópico y todos los agentes obtienen está información al subscribirse a dicho tópico. está topología nos presenta la ventaja que todos los agentes llegan a tener el conocimiento de la pose de todos los otros agentes, los cual puede llegar a ser bastante útil para algunos algoritmos de robótica de enjambre.

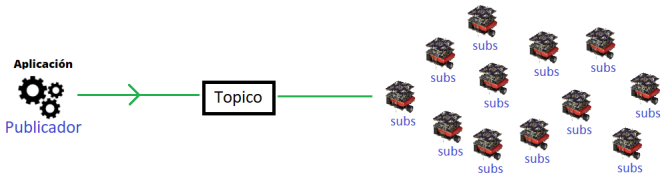

Figura 56: Topología de comunicación general.

- **Comunicación individual**: esta topología crea un tópico por cada uno de los agentes presentes en el sistema, por medio del cual la aplicación (publicador) envía la información de ese único agente, el cual está subscrito a dicho tópico. De esta forma logramos obtener una comunicación 'privada' y directa con cada uno de los agentes que se encuentren en el sistema, a diferencia del anterior ahora cada agente únicamente conoce su pose y no la del resto de os agentes, está topología también es una de las más implementadas en robótica de enjambre.

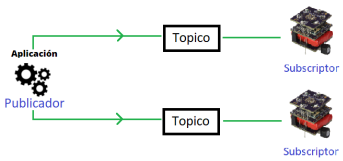

Figura 57: Topología de comunicación individual.

11.3. Topología variable

Para aprovechar de mejor forma la flexibilidad que nos presenta el protocolo MQTT para poder implementar diferentes topologías de comunicación dentro del sistema, en la última opción del menú 'Crear configuración adicional' se le permite al usuario crear la topología que el desea implementar, por medio de este menú el usuario puede crear un nuevo tópico indicando el nombre de dicho tópico y especificar que agentes estarán subscritos a este tópico, al igual que la información que se enviara por medio de este tópico.

Como se mencionó en el capítulo "Aplicacion Robotat - Python" la aplicación obtiene la pose de cada uno de los agentes por medio de un algoritmo de visión por computadora, gracias a esto la aplicación conoce la pose de cada uno de los agentes activos en el sistema. Dicha pose es la que utilizan los agentes al momento de ejecutar su algoritmo de control interno, pero también puede existir el caso en que para evaluar algún algoritmo de robótica swarm algún agente necesite saber la pose de otro agente en el sistema. Para lograr esto, a dicho agente le tendríamos que enviar tanto la información de si mismo, como la del otro agente para que dicho algoritmo swarm se pueda ejecutar de forma exitosa. Así como está ejemplo, pueden existir una gran variedad de algoritmos que necesiten una configuración especifica de la comunicación para poder ser evaluados en una mesa de pruebas. Para poder cumplir con esta necesidad, se desarrolló este proceso, en el cual, como se mencionó anteriormente el usuario define la topología a su necesidad indicando la información que le llega a cada uno de los agentes, pudiendo así evaluar algoritmos más complejos. La siguiente imagen muestra visualmente como se pueden implementar diferentes topologías y también controlar el flujo de información por cada tópico.

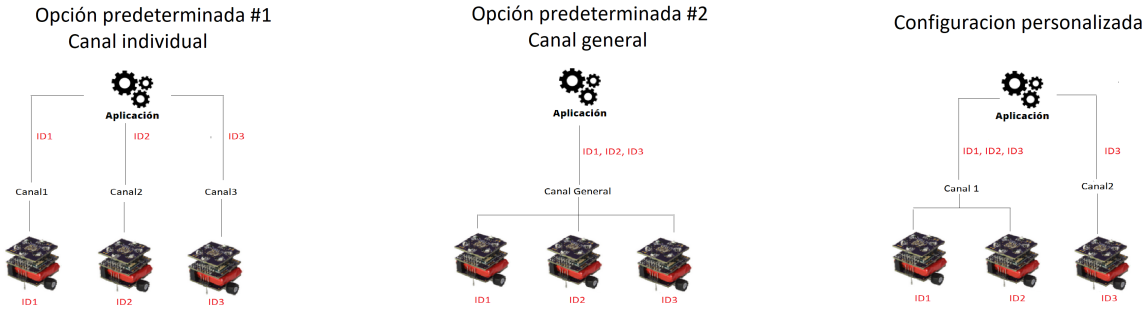

Figura 58: Flexibilidad en el diseño de la red de comunicación.

Es importante resaltar que en esta etapa del sistema la red únicamente es diseñada, a continuación, se muestra el proceso donde la red es creada por medio de los agentes al momento de cargarles el controlador.

Creación de la red de comunicación

Como se ha mencionado anteriormente, la red de comunicación se crea por medio de los agentes luego del proceso de carga del controlador. Para poder crear la red de una forma eficiente, se definió que cada agente verificara a que tópicos se deben de subscribir y suscribirse a dichos tópicos luego de reiniciarse al descargar el archivo del servidor. Al momento de diseñar la red, la aplicación guarda la está estructura de la topología diseñada por el usuario en dos variables que tienen la siguiente estructura, donde se guarda que agentes se deben de subscribir a cada tópico y que información será enviada en cada tópico.

Tópico - Agentes subscritos

Tópico 1	Agente1, Agente2, Agente 3
Tópico 2	Agente1, Agente5
Tópico 3	Agente3, Agente1, Agente2
Tópico 4	Agente2

Tópico - Información a enviar

Tópico 1	Agente1, Agente2, Agente3
Tópico 2	Agente2, Agente5
Tópico 3	Agente2
Tópico 4	Agente3

Figura 59: Estructura en que se guarda la topología.

Luego de que el usuario diseña la red que se utilizara durante la simulación, desde la aplicación tiene la opción para ”publicar” este diseño. Por medio de este proceso la aplicación publica en la variable que contiene los tópicos creados y los agentes que se tienen que subscribir a dicho tópico. Cuando los agentes obtienen está variable, proceden a verificar a que tópicos se deben de subscribir recorriendo cada posición (tópico) arreglo. Cuando el agente encuentra su identificador guarda el nombre del tópico en el cual se encuentra, de esta forma cada agente obtiene el nombre de todos los tópicos a los cuales se tiene que subscribir y los almacena en una variable string en la base de datos.

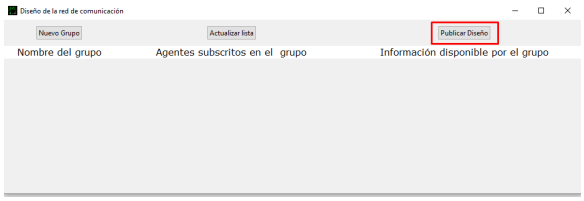

Figura 60: Publicar topología.

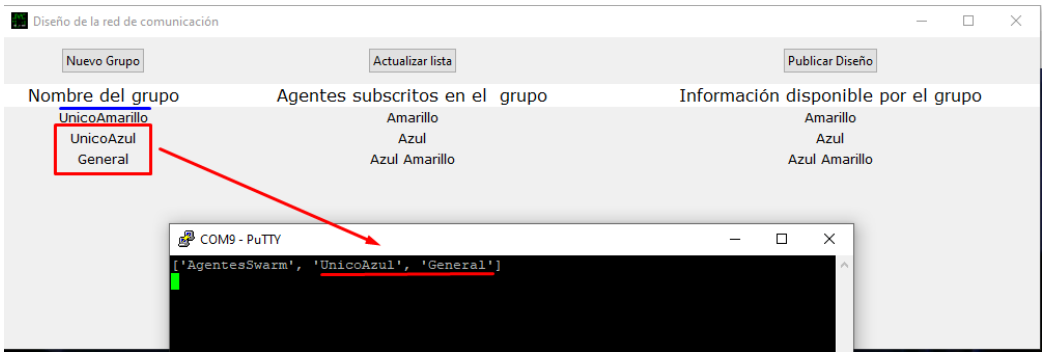

Figura 61: Filtrado de de tópicos a suscribirse - Agentes.

De esta forma hacemos que cada agente 'cree' la parte de la red de comunicación en la que el se ve involucrado, logrando así un proceso eficiente y escalable ya que este proceso no se centra en un único ente para poder crear la red si no que se crea de forma distribuida.

Formato del payload

Para poder implementar una comunicación eficiente, el formato en el cual se envía la información, el cual se puede observar en la Figura 37, se definió con la siguiente estructura:

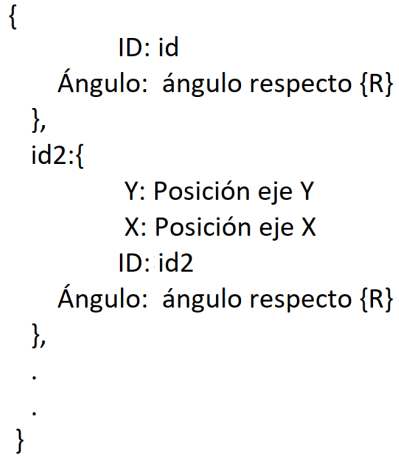

Figura 62: Formato del envió de datos.

Como podemos observar en la Figura 62, la información es enviada en formato de objetos, cada objeto representa y contiene la información de un agente. De esta forma logramos estructurar y enviar el mensaje de forma eficiente en la aplicación y leerlo de forma sencilla en cada uno de los agentes. El código para leer y almacenar está información de forma ordenada en el agente se presenta en la Figura 63, por medio de la función Actualizar().

```
def Actualizar(payload):
    for paquete in payload:
        Agente[paquete["ID"]] = {
                                "X": paquete["X"],
                                "Y": paquete["Y"],
                                "ID": paquete["ID"],
                                "Angulo": paquete["Angulo"]
                                }
```

Figura 63: Lectura y almacenamiento de datos en el microcontrolador.

Este proceso es uno de los que más se mejoró en comparación a las fases anteriores del Robotat, ya que el modelo desarrollado en esta etapa nos permite implementar una gran variedad de configuraciones, tanto en la topología de la red, como en la información que se envía en cada uno de los canales. De igual forma nos permite desligarnos de la relación entre la cantidad de agentes y el rendimiento y nos permite enfocarnos en la configuración de los tópicos, lo que nos permite obtener un sistema más escalable y eficiente al momento de la simulación.

CAPÍTULO 12

Rendimiento del sistema

Para poder validar la escalabilidad y rendimiento del sistema, se desarrolló un proceso interno en la aplicación por medio del cual está auto evalúa su rendimiento, determinando a la cantidad de paquetes de información que puede enviar por segundo en cada tópico, siendo esta medida el periodo de actualización de datos que tiene cada agente al momento de ejecutar su algoritmo de control. De esta forma logramos obtener una muy buena aproximación del rendimiento que tendrá le sistema general durante la simulación en base a la red especifica que diseño el usuario. Esto representa una gran ventaja ya que este proceso le permite al usuario poder variar la topología de comunicación y observar el rendimiento que tendrá el sistema durante la simulación con dicha topología especifica, permitiéndole tomar decisiones sobre cuál es la topología más eficiente para implementar durante la simulación

Para evaluar el rendimiento del sistema, se decidió desarrollar este proceso y no solamente realizar las pruebas en base a las topologías más utilizadas los algoritmos de robótica swarm y calcular el rendimiento aproximado del sistema, ya que de está forma le permitimos observar al usuario el rendimiento del sistema para cualquier configuración deseada.

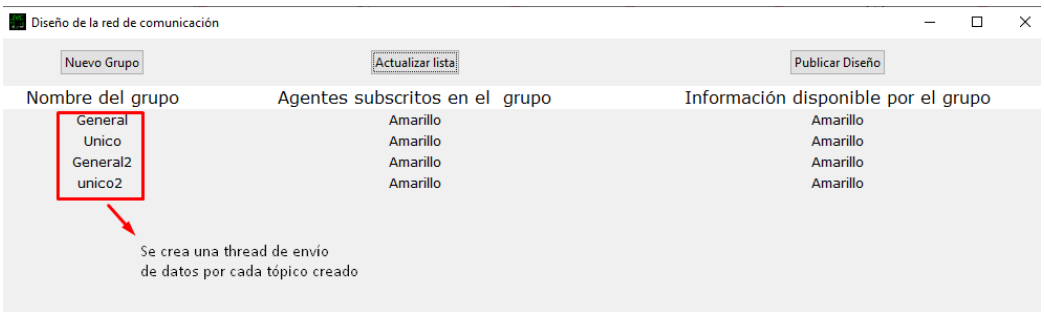

Figura 64: Creación de la topología de comunicación.

Como se puede observar el la Figura 64, en el menú de diseño de la red de comunicación, el usuario tiene una vista gráfica de la topología de red que se acaba de configurar. Luego de realizar dicha configuración en la opción de 'configuración' el usuario tiene acceso al proceso 'Evaluar Rendimiento de la red', Figura 65, por medio del cual como su nombre lo indica la aplicación auto evaluara la cantidad de paquetes de información que puede enviar cada segundo, durante 5 segundos.

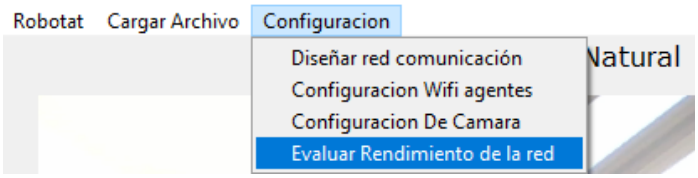

Figura 65: Proceso para evaluar rendimiento.

Al seleccionar está opción, la aplicación inicia las threads de comunicación y procesamiento de datos por medio de las cuales obtiene y envía la pose de cada uno de los agentes al igual que lo haría al momento de iniciar la simulación, pero en este proceso la aplicación no envía los datos a servidor MQTT para que los agentes obtengan la información, en vez de eso cada thread de comunicación (cada tópico) cuenta con un contador que se incrementa cada ciclo de ejecución durante 1 segundo. De esta forma logramos obtener la cantidad de envíos (ciclos ejecutados) por segundo de cada thread, este proceso se repite durante 5 segundos reiniciando el contador al final de cada segundo.

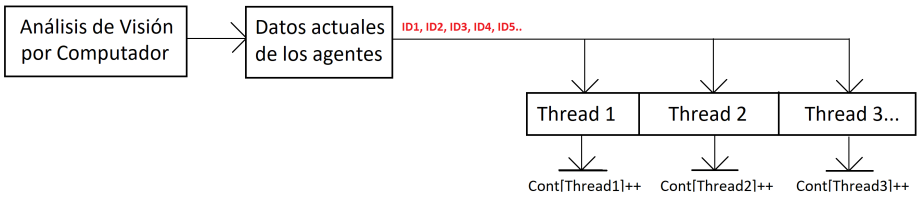

Figura 66: Evaluación del rendimiento del sistema.

Al terminar este proceso la aplicación cuenta con un arreglo con la cantidad de envíos de información que logro hacer la aplicación a cada tópico durante los primeros 5 segundos de ejecución. Está información se le muestra al usuario en la ventana mostrada en la Figura 68, para que el usuario pueda observar y analizar el rendimiento de diferentes topologías y definir cuál es la que se acopla mejor al algoritmo que se quiere evalúa.

Como se puede observar en la Figura 68, al usurario se le despliega el historial, la cantidad máxima, promedio y minina de publicaciones realizadas en cada tópico durante los primeros 5 segundos de la simulación, al igual que una gráfica donde se observa el comportamiento de la comunicación durante esos 5 segundos.

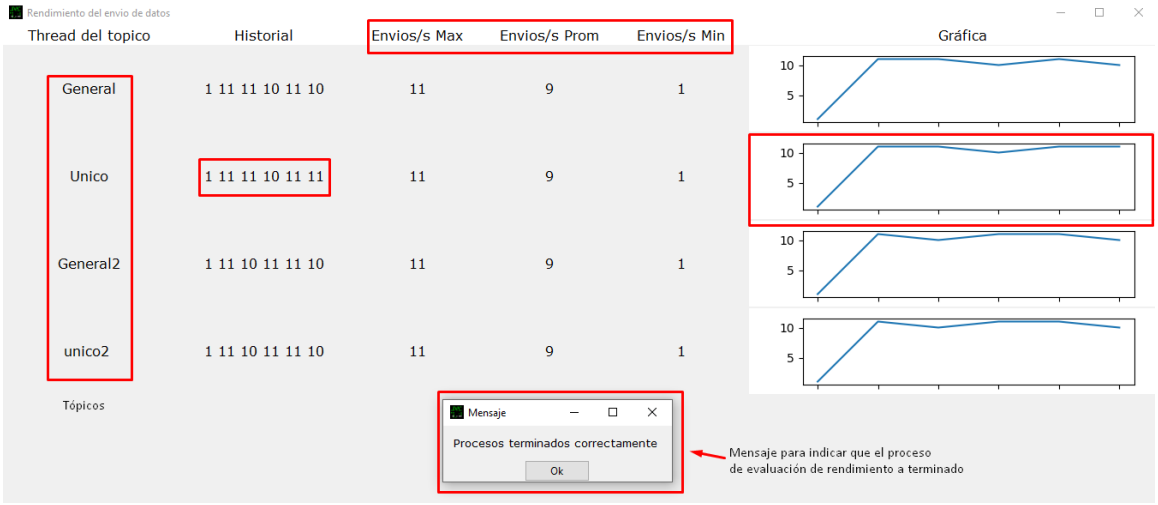

Figura 67: Ventana de resultados.

Como se puede observar la aplicación cuenta con un tiempo de actualización de 10 envíos por segundo a los agentes ya al momento de estar simulando el algoritmo swarm, el promedio de 9 envíos por segundo mostrado en la pantalla es debido al primer ciclo de ejecución de la rutina de comunicación, en la cual únicamente se da un envió de datos. Quitando este primer ciclo, observamos que el periodo de actualización de datos en el resto de la aplicación oscila entre 10 y 11 actualizaciones de datos por segundo a cada agente, con lo cual, logramos obtener el rendimiento deseado y a su vez logrando cumplir el ultimo objetivo de este trabajo. Para determinar la escalabilidad del sistema se realizaron varias pruebas, en las cuales se variaba tanto la topología de la red, como la cantidad de información que se enviaba en cada tópico. Ya que, como se menciono anteriormente, al utilizar el protocolo MQTT nos desligamos de la limitante directa que teníamos con la cantidad de agentes durante la simulación en las fases anteriores, con este protocolo la limitante teórica se encuentra en la cantidad de tópicos que se creen y la cantidad de información que se envié por cada uno de estos tópicos.

En base a esto se realizaron pruebas incrementando la cantidad de tópicos que se utilizarían en la red, incrementando la cantidad de información que se enviaría a cada tópico y incrementando ambas limitantes, para poder definir la escalabilidad del sistema en base a esas dos características. Es importante mencionar que estas pruebas son realizadas implementando un delay en cada thread de comunicación equivalente al tiempo que toma en ejecutarse el algoritmo de visión de computadora (0.087987) para obtener un resultado mas acertado al momento en que se unifique este código con el realizado por José Molina. Al ingresar este delay al sistema, el tiempo de envió de datos máximo que se puede lograr es de 11.37 11 envíos por segundo.

Prueba base

Esta configuración es la que se tomara como base a comparar con el resto de pruebas en las que se modificaran las características de la red de comunicación. Esta prueba genera un tópico de comunicación por cada agente activo en la red y un tópico general por medio del cual se envía la información de todos los agentes.

Diseño de la red

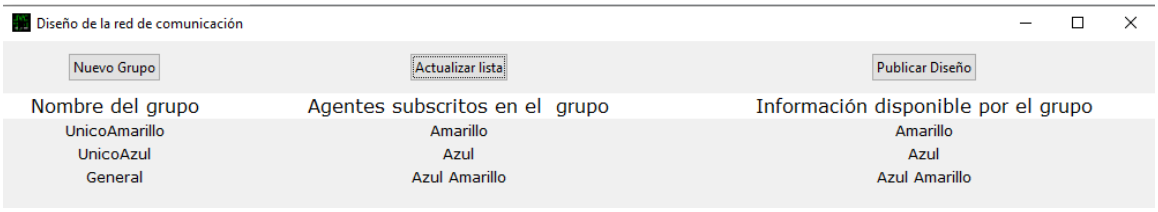

Figura 68: Configuración de red base.

Diseño de la red

Figura 69: Rendimiento de la configuración de red base.

En la Figura 69, se observar un rendimiento de 11 envíos de datos por segundo en promedio en cada tópico, ya durante el periodo de simulación. Con esto observamos que el tiempo de actualización de datos queda limitado y dependiente al tiempo tomado por el algoritmo de visión de computadora.

Incrementando la cantidad de tópicos en la red

En esta configuración se incremento la cantidad de tópicos por medio de los cuales se enviaría la información, pero no la cantidad de información que se envía en cada tópico.

Diseño de la red

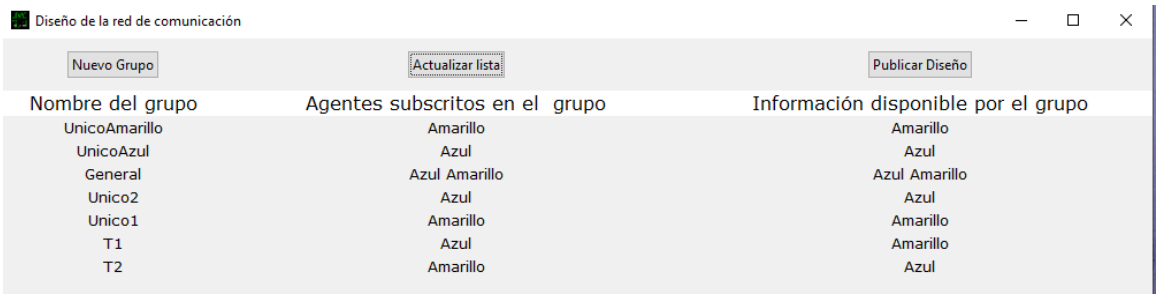

Figura 70: Configuración de red incrementando tópicos.

Diseño de la red

Figura 71: Rendimiento de la configuración con tópicos incrementados.

De igual forma, en la Figura 71, se observar un rendimiento de 11 envíos de datos por segundo en promedio en cada tópico, ya durante el periodo de simulación. Con esto comprobamos que el incrementar la cantidad de tópicos de la red no tiene un impacto en el rendimiento del sistema y que nuevamente el tiempo de actualización de datos queda limitado y dependiente al tiempo tomado por el algoritmo de visión de computadora.

CAPÍTULO 13

Conclusiones

- Se logró desarrollar una plataforma basada en python incluyendo tanto en la aplicación, como en el microcontrolador con Micropython, lo cual mejorando el rendimiento del sistema respecto a la fase anterior.
- Se logró desarrollar una aplicación intuitiva para el usuario, por medio de la cual se pueden programar, configurar, controlar y administrar los agentes del sistema.
- Se progarmó un proceso que le permite al usuario evaluar el rendimiento del sistema para tener una idea clara sobre el tiempo de actualización de información que tendrá cada agente durante la simulación en base a la topología de comunicación configurada.
- Se determinó que el tiempo de envió de datos únicamente se ve limitado a el tiempo que tome en ejecutarse el algoritmo de visión por computadora.
- Al evaluar el rendimiento del sistema se determinó que el promedio de envió de datos en general es de 10 envíos por segundo.
- Al implementar multithreading en la aplicación se mejoró el rendimiento comparado con la la fase anterior, mejorando así también la escalabilidad.
- Al implementar el protocolo de comunicación MQTT, se obtuvo una gran flexibilidad durante el diseño de la red de comunicación, permitiéndonos así implementar una gran variedad de topologías de comunicación.
- Se desarrolló un proceso de carga del controlador a los agentes, el cual nos se ejecuta de forma masiva, independiente, inalámbrica y nos permite validar si este proceso se realizo de forma correcta.
- Se utilizó un servidor web para el proceso de descarga del controlador, lo cual nos permite obtener una alta escalabilidad, ya que le servidor nos permite ejecutar varias peticiones de forma simultenea.
- Se utilizó un servidor web para el proceso de descarga del controlador, ya que sus características nos permite obtener una alta escalabilidad en dicho proceso.

CAPÍTULO 14

Recomendaciones

1. Realizar las pruebas de rendimiento del sistema con el prototipo final de los agentes y el algoritmo de visión de computadora unido con la aplicación, para observar el rendimiento real del Robotat.

2. Realizar pruebas de rendimiento con más de 20 agentes poder observar de mejor forma la escalabilidad de la red de comunicación (tópicos e información que se envía por los tópicos) en diferentes configuraciones.

3. Implementar multiprocessing dentro de la aplicación para poder realizar simulaciones mas complejas, las cuales presenten varias etapas o modelos de análisis de información de forma más eficiente.

4. Realizar un proceso de actualización de software por medio del cual se puedan cargar de forma independiente el resto de archivos que utiliza el microcontrolador (main.py, boot.py, etc) para poder actualizarlos de forma masiva e inalámbrica en el momento que se realice un cambio en alguno de estos archivos o se decida trabajar con otros, este proceso seguiría la misma lógica que el proceso actual de carga del controlador.

5. Mejorar visualmente la ventana y el proceso 'Diseño de la red de comunicación' para que el usuario pueda ingresar la configuración deseada de una forma mas amigable.

6. Utilizar la dirección MAC del microcontrolador para generar el identificador del agente dentro de la red de comunicación, para que este se genere de forma automática al cargarle el programa al microcontrolador.

CAPÍTULO 15

Bibliografía

[1] D. P. y Paul Glotfelter, "*The Robotarium: A remotely accessible swarm robotics research testbed*", Ingles, 15 de Nobiembre* de 2016*.

[2] C. A. y Radhika Nagpal, "*Kilobot: A Low Cost Scalable Robot System for Collective Behaviors.*", Ingles, mayo de 2012*.

[3] M. L. G. y Lawrence H. Kim, "*Zooids: Building Blocks for Swarm User Interfaces*", Ingles, oct. de 2016*.

[4] A. J. R. y Marlon J. Castillo, "*Orus and Bitbot: A low cost testbed and robots for Experimentation in Swarm Robotics*", English, *Articulo*, págs. 1-4, Diciembre de 2019.

[5] Designbuzz, *Swarming Robots - The Future of Robotics*, abr. de 2013. dirección: `https://designbuzz.com/swarming-robots-the-future-of-robotics/`.

[6] *Significado de Wifi*, ago. de 209AD. dirección: `https://www.significados.com/wifi/`.

[7] N. autor, "*Titulo de sección*", en *Titulo de libro*, ép. s*, Editor*, ed., e*, vol. v*, Notas y observaciones*, Dirección*: Editor, mes-nom* de 20XX, cap. Cap-n*, xx-yy*.

[8] *MQTT*. dirección: `http://www.tst-sistemas.es/mqtt/#:~:text=MQTT%20(Message%20Queue%20Telemetry%20Transport,a%20los%20denominados%20%E2%80%9Ct%C3%B3picos%E2%80%9D.`.

[9] A. J. R. Hernández, "*Diseñar e implementar una red de comunicación inalámbrica para la experimentación en robótica de enjambre*", Enero* de 2019*.

[10] *modo de red*. dirección: `http://www.docmirror.net/en/linux/howto/networking/OLSR-IPv6-HOWTO/intro.html`.

[11] R. Kyocera, *Tipos de seguridad wifi mas habituales*, mar. de 2017. dirección: `https://smarterworkspaces.kyocera.es/blog/tipos-de-seguridad-wifi-mas-habituales/`.

[12] I. Togizbayev, *Third week (CN)-TogNets*, sep. de 2019. dirección: `https://medium.com/@islam090301/third-week-cn-tognets-eba542330382`.

[13] A. D. Moor, *Python GUI programming with Tkinter*, English, e1. 35 Livery Street Birmingham B3 2PB, UK.: Packt, mayo de 2018.

[14] Onlinebooksreview y Onlinebooksreview, *Tips and Tricks for Creating a Reliable Embedded System*. dirección: https://www.onlinebooksreview.com/articles/best-tips-and-tricks-for-creating-a-reliable-embedded-system.

[15] M. Gudino, *What is a Microcontroller? A Look Inside a Microcontroller: Arrow.com*, mar. de 2020. dirección: https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller.

[16] A.-T. team, "*ESP-01 WiFi Module*", Ingles, * de 2015*.

[17] *MicroPython*. dirección: https://micropython.org/.

[18] T. DiCola, *MicroPython Basics: Load Files Run Code*. dirección: https://learn.adafruit.com/micropython-basics-load-files-and-run-code/overview.

[19] Ene. de 2018. dirección: https://mosquitto.org/.

[20] *MQTT broker*. dirección: IMG%20https://micropython-iot-hackathon.readthedocs.io/en/latest/mqtt.html.

[21] G. Zaccone, *Python parallel Programming Cookbook*, English, e1. 35 Livery Street Birmingham B3 2PB, UK.: Packt, Agosto de 2015.

[22] V. K. G, *DIY: Multithreading vs Multiprocessing in Python*, abr. de 2020. dirección: https://levelup.gitconnected.com/diy-multithreading-vs-multiprocessing-in-python-fb93698ca7f3.

[23] *Accessible Robotics Swarm*. dirección: https://www.asme.org/topics-resources/content/accessible-robotics-swarm.

CAPÍTULO 16

Anexos

16.1. Repositorio de trabajo

Todo el código desarrollado durante esta fase del Robotat y los recursos necesarios para la aplicación funcione correctamente se encuentran en el siguiente repositorio publico de github.

Robotat: https://github.com/Joancf97/Robotat.