Desarrollo de una placa de expansión para los agentes Pololu 3Pi+ que expanda sus capacidades dentro del ecosistema Robotat.

José Luis Alvarez Pineda

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo de una placa de expansión para los agentes Pololu 3Pi+ que expanda sus capacidades dentro del ecosistema Robotat.

Trabajo de graduación presentado por José Luis Alvarez Pineda para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo de una placa de expansión para los agentes Pololu 3Pi+ que expanda sus capacidades dentro del ecosistema Robotat.

Trabajo de graduación presentado por José Luis Alvarez Pineda para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

(f) _____

M.Sc. Miguel Enrique Zea Arenales

Tribunal Examinador:

(f) _____

M.Sc. Miguel Enrique Zea Arenales

(f) _____

Ing. Jonathan Mansilla

(f) _____

Ing. Christopher Chiroy

Fecha de aprobación: Guatemala, 20 de enero de 2024.

Prefacio

Agradezco a la Fundación Juan Bautista Gutiérrez y a su equipo de becas, quienes me acompañaron y me brindaron la oportunidad de estudiar la carrera que deseaba en la UVG.

Quiero expresar mi profundo agradecimiento a mi padre, Carlos, y a mi madre, Nohemy, quienes me respaldaron a lo largo de mi carrera, proporcionándome las herramientas y los recursos necesarios para seguir adelante, y siempre creyendo en mí.

Agradezco a mis hermanos, Carlos y Karla, por su apoyo constante y sus palabras de aliento a lo largo de este proceso.

También deseo expresar mi gratitud a mi asesor de tesis, el MSc. Miguel Zea, por compartir su conocimiento, dedicar su tiempo para resolver mis múltiples dudas y motivarme constantemente a mejorar mi trabajo.

No puedo dejar de mencionar al Dr. Luis Rivera, quien también brindó parte de su tiempo para responder a mis preguntas generales sobre mi trabajo, y junto a Miguel, buscaron herramientas y alternativas para mejorar la calidad de mi proyecto.

Por último, pero no menos importante, quiero agradecer a mis amigos y compañeros de carrera. Juntos nos apoyamos académica y emocionalmente a lo largo de este proceso, y todas las experiencias compartidas han dejado una huella significativa en mi vida.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este trabajo se desarrolló con el objetivo de ampliar las capacidades de los agentes Pololu 3Pi+ utilizados en la plataforma de pruebas Robotat. Se estructuró en tres partes principales, donde cada parte se complementa para dotar al agente de capacidades adicionales.

En primer lugar, se configuró el TinyS3 en PlatformIO utilizando el framework ESP-IDF para habilitar la comunicación WiFi con un cliente a través de un socket TCP. Además, se configuró el módulo SPI para comunicarse con la OpenMV Cam H7, logrando una tasa de transmisión de hasta 40 Mbps sin pérdida de datos, permitiendo realizar una transmisión en tiempo real que alcanza hasta 8 FPS con una imagen de baja resolución en blanco y negro. Luego, se fabricó una PCB de doble cara con componentes SMD en la cual se montaron el TinyS3, la OpenMV Cam H7 y un módulo elevador de voltaje DC. Esta placa tiene la capacidad de montar un manipulador serial externo, además, se realizó un análisis de potencia de los módulos para determinar las características intrínsecas de la placa y se determino que el uso de una celda 18650 de 3000 mAh de capacidad proporcionaba hasta 2 horas de autonomía para pruebas.

El manipulador serial se diseñó con 2 grados de libertad y un efector final tipo garra, utilizando servomotores MG90 como actuadores en sus juntas. Logró una precisión de hasta 0.1° de movimiento utilizando la cinemática inversa analítica o configuraciones en bruto de sus juntas. Por último, se implementó un algoritmo de estacionamiento automático bajo demanda, utilizando el algoritmo de RRT proporcionado por Matlab y espacios de estado de Dubin para orientar al agente con una precisión de hasta 2° a lo largo de su trayectoria, permitiendo al agente estacionarse en una estación de carga libre en la plataforma del Robotat.

Abstract

This work was carried out with the aim of expanding the capabilities of the Pololu 3Pi+ agents used in the Robotat testing platform. It was structured into three main parts, where each part complements the others to provide the agent with additional capabilities.

First, TinyS3 was configured in PlatformIO using the ESP-IDF framework to enable WiFi communication with a client through a TCP socket. Additionally, the SPI module was configured to communicate with the OpenMV Cam H7, achieving a transmission rate of up to 40 Mbps without data loss, allowing real-time transmission reaching up to 8 FPS with a low-resolution black and white image. Subsequently, a double-sided PCB with SMD components was manufactured, on which the TinyS3, the OpenMV Cam H7, and a DC voltage booster module were mounted. This board has the capacity to mount an external serial manipulator. Furthermore, a power analysis of the modules was conducted to determine the intrinsic characteristics of the board, and it was determined that the use of a 3000 mAh 18650 cell provided up to 2 hours of autonomy for testing.

The serial manipulator was designed with 2 degrees of freedom and a claw-type end effector, using MG90 servomotors as actuators at its joints. It achieved a precision of up to 0.1° of movement using either analytical inverse kinematics or raw joint configurations. Finally, an automatic parking algorithm was implemented on-demand, using the RRT algorithm provided by Matlab and Dubin state spaces to guide the agent with an accuracy of up to 2° along its path, allowing the agent to park at a free charging station on the Robotat platform.

CAPÍTULO 1

Introducción

El Robotat busca la implementación de plataformas robóticas para la experimentación y el estudio de algoritmos de control, tanto individual como de enjambre. Para ello, se utilizan los agentes Pololu 3Pi+. Sin embargo, el uso de estos agentes requiere un posicionamiento y configuración manual. Además, es importante destacar que estos agentes solo permiten validar algoritmos de control en el seguimiento de trayectorias.

Por esta razón, el objetivo de la investigación fue el diseño y la fabricación de una placa de expansión que permita la integración de un módulo de visión por computadora y un manipulador serial, al mismo tiempo que habilitara la ejecución de algoritmos de estacionamiento automático en estos agentes robóticos.

El trabajo se dividió en tres partes fundamentales, todas ellas interconectadas mediante un *soft access point* (AP) creado con el TinyS3, a través del cual se gestionó la recepción y transmisión de datos de control. En una primera etapa, se estableció la interconexión entre el módulo TinyS3 y la cámara OpenMV Cam H7 mediante el bus de comunicación SPI. Este proceso permitió la adquisición de imágenes en tiempo real y su posterior transmisión a través de una conexión WiFi a un cliente que las solicitara.

En la segunda parte, se abordó la implementación de un manipulador serial de tres grados de libertad diseñado para ampliar las capacidades de los robots Pololu 3Pi+. Este manipulador serial se controló de forma remota mediante comunicación WiFi, lo que permitió una configuración dinámica de sus actuadores basada en la recepción de datos en bruto o coordenadas específicas. La cinemática inversa analítica desempeñó un papel esencial en el cálculo de las configuraciones necesarias para el manipulador serial.

Por último, en la tercera fase del trabajo, se centró en la implementación de un algoritmo especializado para la planificación de trayectorias y la ejecución de maniobras de estacionamiento automáticas. Este algoritmo se activó tanto bajo demanda del usuario, como en situaciones críticas o niveles bajos de batería. Su función principal fue generar trayectorias eficientes que permitieran a los agentes robóticos regresar a su estación de carga, preparándolos para futuros experimentos y tareas.

CAPÍTULO 2

Antecedentes

En el Robotat se busca la implementación de plataformas robóticas para la experimentación y estudio de algoritmos individuales de control y para algoritmos de enjambre. Para la implementación de algoritmos de enjambre y control individual están las plataformas Pololu 3pi+ y se está integrando los drones *Crazy Fly*. Al implementar robots diferenciales como parte de las pruebas se debe de buscar una manera de integrar rutinas para un almacenamiento adecuado que permita su puesta en marcha de manera rápida, para ello la implementación de parqueo automático y zonas de recarga mejoraría la autonomía de las pruebas y el cuidado de los robots.

2.1. El ecosistema Robotat

El ecosistema Robotat es una plataforma para la experimentación y validación de prototipos y algoritmos de robótica instalado en el laboratorio CIT-116 de la Universidad del Valle de Guatemala. Este opera con un sistema de captura de movimiento OptiTrack y se comunica con agentes robóticos dentro de él por medio de WiFi. En un inicio Perafán en su tesis de graduación [1] desarrolló una plataforma de comunicación por WiFi que utilizaba el protocolo MQTT para comunicarse con agentes, permitiendo el conectar hasta 11 agentes previo a aumentar su latencia [1]. Posteriormente este modelo de comunicación se descartó y en la actualidad utiliza un protocolo TCP/IP para realizar la comunicación por WiFi desde un servidor local.

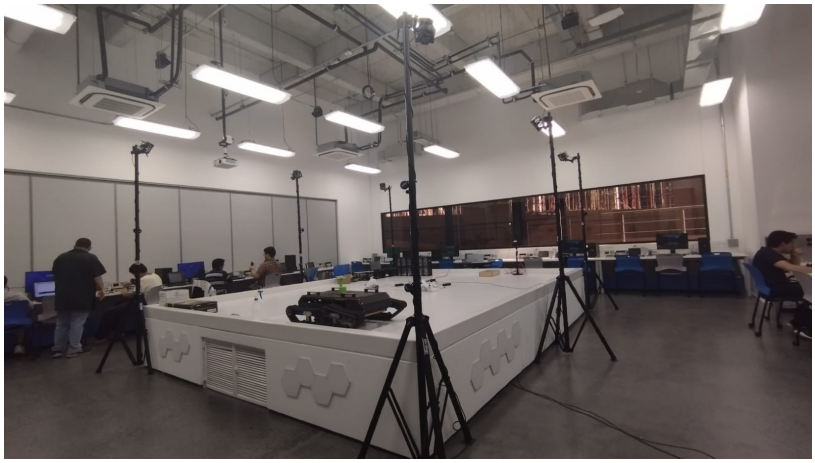


Figura 1: Plataforma de pruebas del ecosistema Robotat.

2.2. Robots diferenciales como plataformas educativas

En el Robotarium de GeorgiaTech [2] se utilizan los GRITSBots que son una plataforma de bajo costo, tamaño reducido y un diseño modular que apunta a simplificar la implementación de múltiples agentes en el Robotarium. Esta simplificación se logra por medio de características de los GRISBots como la carga automática, registro automático en los servidores y reprogramación por medio de comunicación inalámbrica. Estos agentes se comunican con el módulo central por medio de Wifi [3].

En la Universidad de Melbourne un grupo de estudiantes de Maestría desarrollaron un robot diferencial con un enfoque educativo para tareas automatizadas e implementación de machine learning, conservando aspectos como modularidad y tamaño reducido, pero en este caso utilizaron componentes de alto costo como la placa Nvidia Jetson Xavier NX como módulo de control y procesamiento principal. Estos apuntan a una implementación de enjambres en almacenes, pero su escalabilidad es limitada por el alto costo de su producción [4].

Las dos alternativas previamente presentadas son ejemplos de plataformas desarrolladas específicamente para un ecosistema o propósito general, pero existen también robots diferenciales comerciales utilizados por distintas universidades como plataformas de prueba y desarrollo, permitiendo saltar el paso de desarrollo de estos agentes. Este caso se puede observar en la Universidad del Valle de Guatemala y la implementación de los Pololu 3Pi+ a su ecosistema Robotat, que de momento han permitido presentaciones de control individual de cada agente. Otra alternativa comercial son los E-Puck empleados en EPFL y son utilizados para cursos como procesamiento de señales, control automático y estimación de posición y trayectorias para robots móviles [5] y también están los Khepera de Kteam que han sido utilizados por distintos laboratorios e instituciones, tanto de manera virtual como en simuladores tales como Webots.

La implementación de plataformas comerciales presenta otras características que le dan un valor agregado, como una comunidad activa y comunicación con los desarrolladores pa-

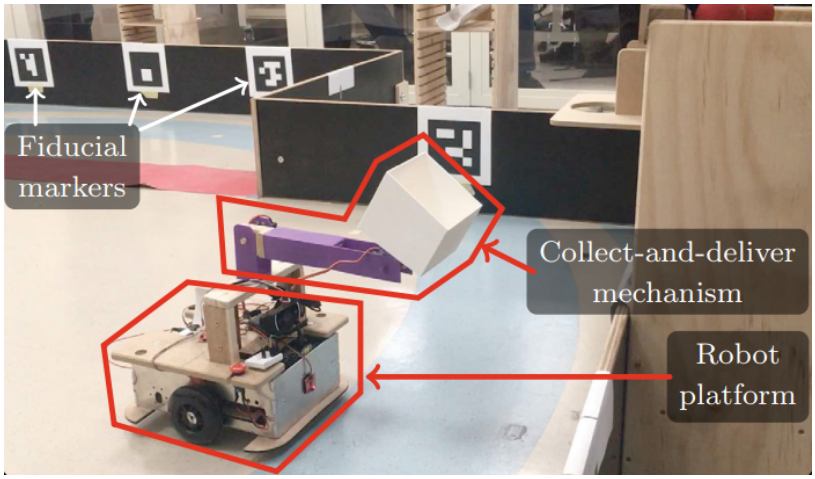
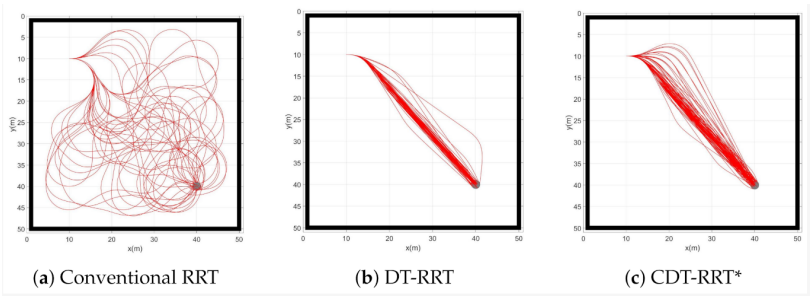


Figura 2: Plataforma robótica implementada en la Universidad de Melbourne [4].

ra la consulta de posibles implementaciones con las plataformas, repuestos establecidos y eliminación del tiempo de diseño, producción y pruebas de plataformas similares.

2.3. Algoritmos de planeación y generación de trayectorias para parqueo automático

Los esquemas de planeación de trayectorias pueden ser divididos en dos grandes grupos, aquellos que se basan en algoritmos de búsqueda en una cuadricula y otros que se basan en algoritmos de búsqueda por pruebas. Para el control de plataformas con ruedas Qi Kong et al [6] utilizó el esquema del algoritmo de búsqueda en cuadrícula, buscando la ruta óptima y una alternativa, para luego suavizarlas y procurar el movimiento adecuado de la plataforma. En cuanto al grupo de algoritmos basados en búsqueda por pruebas están los algoritmos de Rapidly exploring Random Trees (RRT), Closed Loop Rapidly exploring Random Trees (CL-RRT), Dual Tree Rapidly exploring Random Tree (DT-RRT), entre otros. El paper de Hyunki Kwon et al.[7] propone una nueva alternativa a estos métodos que es el Car like Dual Tree Rapidly exploring Random Tree (CDT-RRT) el cual es un DT-RRT orientado específicamente a robots sobre ruedas. En su propuesta se compara su desempeño contra el DT-RRT para un robot móvil de 4 ruedas y se observa que ambos logran la convergencia a sus objetivos, por lo cual tanto DT-RRT como CDT-RRT son alternativas viables para los algoritmos de planeación de trayectorias en robots móviles.



(a) Conventional RRT (b) DT-RRT (c) CDT-RRT*

Figura 3: Comparativa de algoritmos RRT para planeación y generación de trayectorias [7].

2.4. Módulos de visión de computadora en el mercado

En la actualidad existe una alta variedad de módulos orientados a visión de computadora o compatibles con módulos de cámara para la misma tarea. De la familia de OpenMV están los módulos Cam M4 y M7, los cuales ya han sido descontinuados, pero pueden ser encontrados en el mercado. De la misma familia estan también las Cam H7 y H7 Plus que son los modelos más recientes y potentes de la compañía [8]. Otra alternativa es la Pyboard lite v1.0, el cual es un módulo orientado al procesamiento y análisis de imágenes que es compatible con diversos modelos de cámara y todo su procesamiento se basa en MicroPython [9]. Se puede mencionar la Pixy2 la cual está orientada al análisis de imágenes, es compatible con Arduino y cualquier otro microcontrolador capaz de comunicarse al menos por SPI [10]. También está la JeVois-A33 en su modelo estándar, la cual tiene capacidad de implementar redes neuronales orientadas a visión de máquina de distintos *frameworks* [11]. Todos los modelos previamente mencionados tienen la capacidad de integrar inteligencia artificial para el reconocimiento de colores, objetos, líneas, entre otros.

Si bien existe una alta gama de módulos y cámaras en el mercado las cuales se pueden emplear en múltiples aplicaciones de visión de computadora, las alternativas previamente presentadas presentan una baja barrera de entrada, bajo costo, amplia documentación y la comunidad que las utiliza sigue activa, haciéndolas alternativas viables para su implementación en investigación, desarrollo y experimentación a nivel universitario.

En la UVG se han realizado pruebas con la ESPCam y la JeVois Cam. Se comparó la JeVois con la ESPCam y se encontró que si bien su desempeño fue similar en las pruebas realizadas, la alternativa seleccionada fue la JeVois Cam por su alta velocidad de procesamiento y la facilidad de implementar el reconocimiento con el entorno desarrollado por el fabricante. Al observar el microcontrolador en el que se basa la JeVois Cam y la OpenMV Cam H7 se puede observar que la segunda utiliza la misma arquitectura a una menor velocidad, pero tiene la capacidad de implementar las mismas características de reconocimiento de imagen e inteligencia artificial que la primera con un menor consumo de potencia, un diseño más reducido y capacidad modular de intercambiar las cámaras, presentándose como una mejor alternativa para mejorar la autonomía de las plataformas donde se implementará sin disminuir su alcance y características [12].

CAPÍTULO 3

Justificación

El ecosistema Robotat es una mesa de pruebas para múltiples estilos de robot dedicada al estudio individual de plataformas robóticas y robótica de enjambre. Esta cuenta con un sistema de captura de movimiento de la marca OptiTrack y en los últimos años se ha implementado la comunicación en una red WiFi por medio del protocolo TCP/IP para la adquisición de los datos del sistema de captura y comunicación con algunos agentes robóticos que han sido ajustados para interactuar en el ecosistema. La implementación de la placa de expansión a los Pololu 3Pi+ permitirá la integración de múltiples agentes en el ecosistema, brindándoles la capacidad de comunicarse por medio de WiFi adentro del ecosistema. También dotará a los agentes con la capacidad de implementar visión de máquina utilizando el módulo de cámara de OpenMV H7, con el cual se podrá acceder a un *webstream* que muestre lo que ve el agente u obtener las imágenes en bruto para su procesamiento posterior. En la placa se incluirá un micro brazo robótico el cual también se manejará por medio del ecosistema Robotat, ampliando el alcance de la interacción del agente con su entorno.

El ecosistema Robotat apunta a la implementación de una gran cantidad de agentes para los algoritmos de enjambre. Por esta razón, la implementación de un algoritmo para el parqueo automático de los agentes Pololu al tener baja batería o recibir la orden es parte esencial de la autonomía del ecosistema. El algoritmo mejorará el desempeño de los agentes en el ecosistema ya que estos se retirarán de la pista previo a detenerse, mejorando la ejecución de los experimentos. Otro aspecto relevante es el orden y puesta en marcha de las pruebas ya que los agentes retornarán a una posición fija luego de terminar las pruebas y en ella se recargará por lo cual estarán listos para futuras pruebas y se minimizará el riesgo de la falta de agentes por descarga o estar mal colocados al inicio de la prueba.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Diseñar y manufacturar una placa de expansión para los robots Pololu 3Pi+ que permita la integración de un módulo de visión de computadora, un micro-manipulador robótico y correr algoritmos de parqueo automático en estos agentes.

4.2. Objetivos específicos

- Diseñar y manufacturar una placa de expansión basada en ESP32 adecuada según los requerimientos de tamaño, periféricos y potencia.
- Diseñar e implementar un micro brazo robótico para el Pololu 3Pi+.
- Adaptar el módulo OpenMV Cam H7 al módulo ESP32 utilizando un protocolo de comunicación que permita una alta transferencia de datos.
- Implementar un algoritmo de parqueo automático activado ante niveles bajos de batería o de manera remota, utilizando rastreo de posición por medio del sistema OptiTrack.

CAPÍTULO 5

Alcance

Este trabajo se centra en el diseño y la fabricación de una placa de doble cara compatible con el agente Pololu 3Pi+. Esta placa puede ser manufacturada tanto en una empresa externa como en el taller interno (MakerLab) de la Universidad del Valle de Guatemala, utilizando tecnología SMD en sus componentes. La placa tiene la capacidad de integrar un TinyS3, la OpenMV Cam H7 y el manipulador serial desarrollado. Además, utiliza una celda 18650 de 3000 mAh como fuente de alimentación para los módulos, cumpliendo con los requerimientos de potencia y autonomía.

El TinyS3 utiliza un Socket TCP para comunicarse con los clientes y el bus SPI para comunicarse con la OpenMV Cam H7, logrando una tasa de transferencia de hasta 40 Mbps. El manipulador serial emplea servomotores MG90 como actuadores en sus juntas y puede ser fabricado utilizando tecnología FDM. Las juntas pueden ser modificadas por el cliente a través de configuraciones en bruto o coordenadas, usando como controlador el TinyS3.

En la implementación del algoritmo de estacionamiento bajo demanda se utiliza el algoritmo RRT de Matlab para definir la trayectoria y los marcadores de cuerpo rígido del Optitrack para detectar obstáculos. El uso de este algoritmo se debe a su rapidez en la generación de la trayectoria y la capacidad de implementar el espacio de estados de Dubin para la orientación.

CAPÍTULO 6

Marco teórico

6.1. El uniciclo

El robot móvil sobre ruedas más simple es el uniciclo, el cual tiene una rueda de radio r. La configuración de la rueda se escribe como $q = (\phi, x, y, \theta)$, donde (x, y) es el punto de contacto, ϕ es el ángulo de dirección de la rueda y θ es la rotación de la rueda. La configuración del cuerpo es (ϕ, x, y) [13].

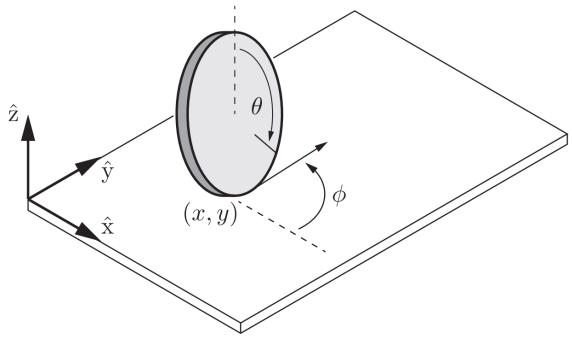


Figura 4: Parámetros y configuración del uniciclo [13].

Las ecuaciones que describen la cinemática del sistema son

$$\dot{q} = \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ r\cos\theta & 0 \\ r\sin\theta & 0 \\ 1 & 0 \end{bmatrix} = G(q)u = g_1(q)u_1 + g_2(q)u_2. \tag{1}$$

Las entradas de control de este modelo son $u = (u_1, u_2)$, donde u_1 es la velocidad para

adelante y atrás de la rueda y u_2 es la velocidad de rotación de la rueda sobre el plano. Estas velocidades se encuentran restringidas por las capacidades del robot como [13]

$$\begin{aligned} -u_{1,max} < u_1 < u_{1,max}. \\ -u_{2,max} < u_2 < u_{2,max}. \end{aligned} \tag{2}$$

En el caso de robots móviles con ruedas con modelos cinemáticos no holonómicos estos tendrán la forma $\dot{q} = G(q)u$, para los cuales se deben de tener las siguientes consideraciones [13]:

- No existe derrape, es decir, cuando las magnitudes del vector u son ceros la velocidad del modelo será cero.
- Los campos vectoriales $g_i(q)$ son generalmente funciones de la configuración q utilizada.
- $\dot{q}$ es lineal durante el control.

A la ecuación 1 se le puede hacer una modificación eliminando la cuarta fila, ya que muchas veces no es de interés el conocer el ángulo de rotación de la rueda, quedando la ecuación de control simplificada como [13]:

$$\dot{q} = \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ r\cos\theta & 0 \\ r\sin\theta & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{3}$$

6.2. Robot diferencial

El robot diferencial o *diff-drive* es una de las arquitecturas más simples para robots móviles con ruedas. Consiste en dos ruedas controladas de manera independiente, ambas con un radio r las cuales rotan sobre un mismo eje, así como una o más ruedas giratorias o deslizadores de baja fricción para mantener al robot horizontal. Sabiendo que la distancia entre las llantas es $2d$ y tomando como referencia (x, y) en el punto medio entre las ruedas, la configuración se puede escribir como $q = (\phi, x, y, \theta_L, \theta_R)$ en donde θ_R y θ_L son el ángulo de rotación que presentan las ruedas. Su ecuación cinemática se puede escribir como [13]:

$$\dot{q} = \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \\ \dot{\theta}_L \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} -r/2d & r/2d \\ \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_L \\ u_R \end{bmatrix}. \tag{4}$$

En 4 u_L y u_R son las velocidades angulares de las llantas, en donde un valor positivo mueve al robot para adelante y un valor negativo de reversa. Estas velocidades se encuentran restringidas por el robot en un intervalo $[-u_{max}, u_{max}]$. Al igual que la ecuación del uniciclo,

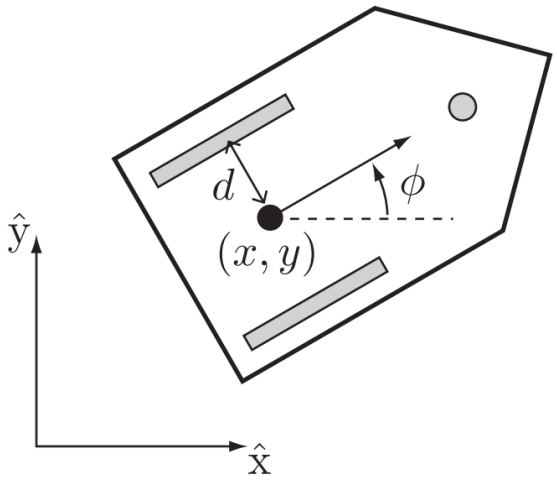


Figura 5: Robot diferencial con una rueda giratoria[13].

de 4 se pueden eliminar las últimas dos filas al no interesar el ángulo de rotación de las ruedas quedando [13]

$$\dot{q} = \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -r/2d & r/2d \\ \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \end{bmatrix} \begin{bmatrix} u_L \\ u_R \end{bmatrix}. \tag{5}$$

Las dos ventajas de un robot diferencial son su simplicidad, ya que los motores se encuentran conectados directamente al eje de cada rueda y la gran maniobrabilidad que estos presentan, ya que tienen la capacidad de girar sobre si mismos, aunque, estos no son ideales para el exterior [13]. La aproximación queda:

$$\begin{aligned} \dot{x} &= v\cos\theta. \\ \dot{y} &= v\sin\theta. \\ \dot{\theta} &= w. \end{aligned} \tag{6}$$

En este modelo v es la velocidad lineal para adelante o atrás del robot diferencial y w es la velocidad de rotación del robot diferencial medida en el punto medio del eje de las ruedas. Esta aproximación se utilizará para los métodos de control planteados.

6.3. Control de posición utilizando control PID

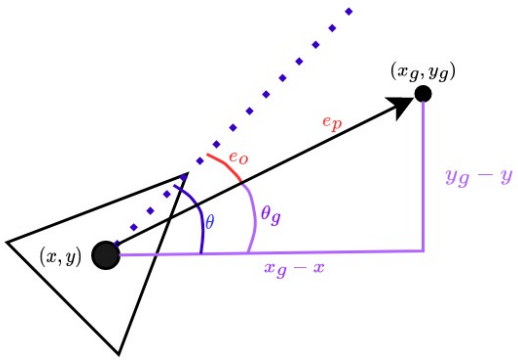


Figura 6: Error de posición y orientación para el robot diferencial.

Para la implementación de este controlador se calculan dos errores, uno de posición e_p y el otro de orientación e_o. Estos son calculados respecto a un punto deseado (x_g, y_g, ϕ_g) como se observa en la Figura 6. Para el error de posición se utiliza:

$$e_p = \left\| \begin{bmatrix} x_g - x \\ y_g - y \end{bmatrix} \right\|_2 . \tag{7}$$

La magnitud de θ_g se determina con:

$$\theta_g = \arctan\left(\frac{y_g - y}{x_g - x}\right). \tag{8}$$

Para el error de posición se requiere acotar su magnitud en el intervalo de $[-\pi, \pi]$, por ello este se puede calcular con:

$$e_o = \text{atan2}\left(\frac{sin(\theta_g - \theta)}{cos(\theta_g - \theta)}\right). \tag{9}$$

Aplicando estos errores a la forma estándar del PID para determinar v y w se obtiene

$$\begin{aligned} v &= PID(e_p) = K_{P_p} e_p + K_{I_p} \textstyle\int_0^t e_p(\tau)\mathrm{d}\tau + K_{D_p} \dot{e}_p. \\ w &= PID(e_o) = K_{P_o} e_o + K_{I_o} \textstyle\int_0^t e_o(\tau)\mathrm{d}\tau + K_{D_o} \dot{e}_o. \end{aligned} \tag{10}$$

Utilizando este esquema se desacopla la posición y orientación, pero presenta problemas de convergencia en algunos casos y su convergencia se realiza en forma de espirales. Para corregir este problema se utiliza un acercamiento exponencial en v calculándose como

$$v = -k(e_p)e_p = \frac{v_o(1 - e^{-\alpha e_p^2})}{e_p} e_p. \tag{11}$$

6.4. Control de posición utilizando control LQI

Otro método de control para este sistema es por medio de LQR, pero este método de control es susceptible a presentar un error constante en su salida ante errores de modelado. Para mitigarlo se incorpora al control una constante de integración [14].

Estableciendo un nuevo estado w se tiene

$$w(t) = \int_0^t (y(\tau) - y_d(\tau))d\tau. \tag{12}$$

Su derivada es

$$\frac{\mathrm{d}}{\mathrm{dt}}w(t) = y(t) - y_d(t) = Cx(t) - y_d(t). \tag{13}$$

Por lo tanto si $\frac{\mathrm{d}}{\mathrm{dt}}w(t) = 0$, entonces el error en estado estable se vuelve 0. Modificando el problema de estado estable agregando la nueva variable w se obtiene

$$\begin{array}{c} E\frac{\mathrm{d}}{\mathrm{dt}}x = Ax + Bu. \\ \frac{\mathrm{d}}{\mathrm{dt}}w = Cx - y_d. \end{array} \tag{14}$$

En esta nueva definición la variable E puede tomar un valor unitario y se puede reescribir en forma matricial como

$$\begin{bmatrix} \dot{x} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ -1 \end{bmatrix} y_d. \tag{15}$$

para lo cual se pueden formar nuevas matrices aumentadas que tienen la forma

$$\begin{array}{ll} A_+ = \begin{bmatrix} A & 0 \\ C & 0 \end{bmatrix}. & B_+ = \begin{bmatrix} B \\ 0 \end{bmatrix}. \\ C_+ = \begin{bmatrix} C & 0 \end{bmatrix}. & x_+ = \begin{bmatrix} x \\ w \end{bmatrix}. \end{array} \tag{16}$$

Las matrices aumentadas se componen de las matrices discretizadas del modelo original. Con esta nueva notación la ecuación queda

$$\begin{array}{c} \dot{x}_+ = A_+x_+ + B_+u. \\ y = C_+x_+. \end{array} \tag{17}$$

Teniendo esta ecuación presente, la entrada de control u al sistema queda como

$$u = -K_+x_+. \tag{18}$$

En donde K_+ se puede calcular con el mismo método que K para el control con LQR, con la diferencia que se usa A_+ en vez de A y B_+ en vez de B, las matrices Q y R para penalizar las variables se siguen trabajando de la misma manera. El tamaño de K_+ es de $n+m$, donde n es la cantidad de variables de estado en el sistema y m es la cantidad de salidas. Estos último valor de K_+ son las constantes para el termino integral de control por salida. Este control permite que la referencia de entrada pueda variar con el tiempo y se ajuste la salida del controlador [14].

Al aplicar este controlador al robot diferencial se requiere de un difeomorfismo que permita él acerca el modelo a un sistema lineal, este se puede hacer como se ve en la Figura 7

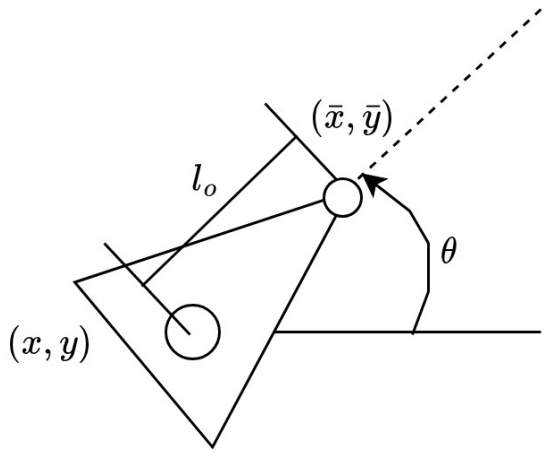


Figura 7: Modelo canónico simplificado para robots móviles.

El nuevo $\tilde{x}$ y $\tilde{y}$ permitirán mapear las velocidades virtuales obtenidas por el LQI. Las ecuaciones de estos quedan como

$$\begin{aligned} \tilde{x} &= x + l_o \cos\theta. \\ \tilde{y} &= y + l_o \sin\theta. \end{aligned} \tag{19}$$

Derivando estas ecuaciones para plantear las matrices de espacio de estados quedan

$$\begin{aligned} \dot{\tilde{x}} &= \dot{x} - l_o \sin\theta\dot{\theta}. \\ \dot{\tilde{y}} &= \dot{y} + l_o \cos\theta\dot{\theta}. \end{aligned} \tag{20}$$

Suponiendo que se puede controlar para mover el robot directamente a $(\tilde{x}, \tilde{y})$ se puede plantear el grupo de matrices como

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & l_0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{21}$$

De esto se puede ajustar la ecuación anterior para obtener los valores de v y w utilizables en el modelo aproximado, quedando como

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & l_0 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \mathbf{M}(l_0, \theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{22}$$

Donde $\mathbf{M}(l_0, \theta)$ es el difeomorfismo que permite el mapeo de las velocidades virtuales obtenidas con el LQI a las del modelo aproximado con el uniciclo.

6.5. Regresando del uniciclo al robot móvil

Al regresar del modelo del uniclo al robot móvil se asocia la Figura 8 con la v y w obtenidas, para v se utiliza

$$v = \sqrt{\dot{x}^2 + \dot{y}^2} = \sqrt{\left(\frac{r(u_r + u_l)}{2}\cos\phi\right)^2 + \left(\frac{r(u_r + u_l)}{2}\sin\phi\right)^2} = \frac{r(u_r + u_l)}{2}. \tag{23}$$

y para el calculo de w se utiliza

$$w = \frac{r(u_r + u_l)}{2d}. \tag{24}$$

Teniendo esta relación presente, la transformación para el modelo simplificado al robot diferencial es [13]

$$\begin{aligned} u_L &= \tfrac{v - wd}{r}. \\ u_R &= \tfrac{v + wd}{r}. \end{aligned} \tag{25}$$

en donde sus límites dependen únicamente de las capacidades y construcción del robot móvil con ruedas.

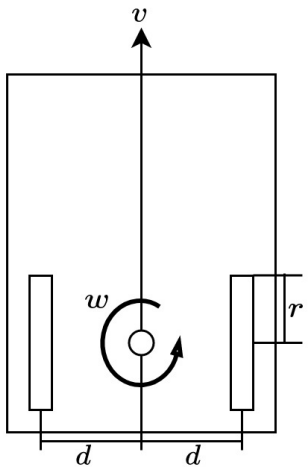


Figura 8: Representación de v y w en el robot diferencial.

6.6. Actuador serial o brazo robótico

Los actuadores seriales o brazos robóticos son manipuladores de cadena cinemática abierta, compuestos de cuerpos rígidos (eslabones) unidos por juntas revolutas actuadas. El punto de interés de este robot es su efector final y para definir su pose, posición y orientación, se utiliza una matriz homogénea de 4x4 como 26 en donde ${}^{B}\mathbf{R}_{E}(\mathbf{q}) \in SO(3)$ es la matriz de rotación que indica la orientación del efector final y ${}^{B}\mathbf{o}_{E}(\mathbf{q})$ es una vector de 3x1 que contiene las posición en $(x, y, z) \in \mathbb{R}^3$ del efector final [13].

$$ {}^{B}\mathbf{T}_{E}(\mathbf{q}) = \begin{bmatrix} {}^{B}\mathbf{R}_{E}(\mathbf{q}) & {}^{B}\mathbf{o}_{E}(\mathbf{q}) \\ \mathbf{0}_{1X3} & 1 \end{bmatrix}. \tag{26} $$

La orientación y posición del efector final se definen con la configuración $\mathbf{q}$, siendo este un vector como 27 en donde los valores de q_1 hasta q_N representan la configuración, ángulo, de cada junta.

$$ \mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_N \end{bmatrix} \in \mathbb{R}^N. \tag{27} $$

La matriz 26 se puede crear a partir del producto de matrices exponenciales atribuidas a un movimiento de tornillo o como el producto de múltiples matrices de traslación y rotación en $\mathbb{R}^3$, donde la convención de parámetros de Denavit-Hartenberg es ampliamente utilizada [13].

6.7. Parámetros de Denavit-Hartenberg

Los parámetros de Denavit-Hartenberg permiten obtener la cinemática directa de una cadena abierta por medio de marcos de referencia colocados en cada una de las juntas, con los cuales se determinan los desplazamientos y rotaciones relativos entre cada marco. Para trabajar con los parámetros de Denavit-Hartenberg se debe de contar con un marco de referencia fijo (base) donde comienza la cadena y un marco de referencia al final de la cadena, el cual describe la pose del efector final. Al ser juntas revolutas de un grado de libertad, las juntas son enumeradas desde 0 hasta n, donde la junta a tierra o base es 0 y el marco de referencia del efector final es n. Los marcos de referencia de las juntas son etiquetados desde 0 hasta n. Las variables que denotan la configuración de la i-ésima junta son θ_i y la cinemática directa de la n-ésima junta se puede expresar como

$$T_{0n}(\theta_1, \ldots, \theta_n) = T_{01}(\theta_1)T_{12}(\theta_2)\ldots T_{n-1,n}(\theta_n). \tag{28}$$

donde $T_{i-1,i} \in SE(3)$ denota el desplazamiento relativo entre los marcos $i-1$ y i. Para la configuración utilizada en los brazos robóticos puede ser encontrada de manera directa por inspección o se especifica como parte del proceso de diseño [13].

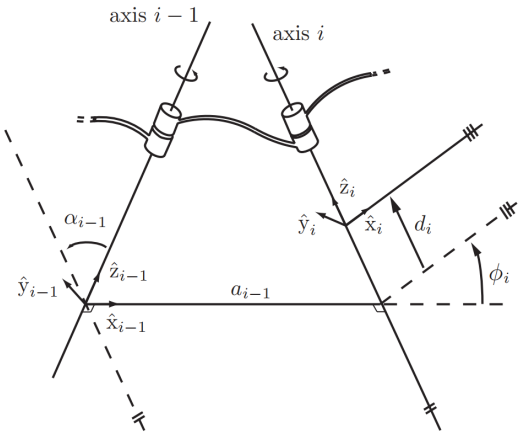


Figura 9: Parámetros de Denavit-Hartenberg ilustrados [13].

En vez de asignar los marcos de referencia de manera arbitraria hasta llegar al efector final, en la convención de Denavit-Hartenberg se sigue un grupo de reglas para asignar los marcos como se ve en la Figura 9, donde se ilustra la asignación para dos juntas revolutas $i-1$ e i conectados por un eslabón $i-1$ [13].

La primera regla es que el eje $\hat{z}_i$ coincida con el eje de la junta i y lo mismo con el eje $\hat{z}_{i-1}$ y la junta $i-1$, su dirección de rotación positiva en el eje $\hat{z}$ se determina con la regla de la mano derecha. Con la dirección del eje $\hat{z}$ asignada se procede a determinar el origen del marco de referencia de la junta. Se busca una línea ortogonal que interseque ambos ejes $\hat{z}_{i-1}$ y $\hat{z}_i$ de las juntas. Al tener ambos ejes conectados por una línea perpendicular, el origen del marco $i-1$ se encuentra en el punto donde la línea intereseca el eje $i-1$. Ahora para determinar el eje $\hat{x}$ del marco de referencia, basta con alinear su dirección hacia la otra

junta paralelo a la línea perpendicular que las interseca y el eje $\hat{y}$ se determina a partir del producto cruz $\hat{x} \times \hat{y} = \hat{z}$ entre los ejes [13].

Luego de haber asignado los marcos de referencia se pueden asignar los parámetros de la siguiente manera:

- El largo de la línea perpendicular a_{i-1} se llama **largo de la junta**. Este largo no necesariamente corresponde al largo del eslabón físico.
- El ángulo α_{i-1} desde $\hat{z}_{i-1}$ hasta $\hat{z}_i$, medido desde $\hat{x}_{i-1}$ se conoce como **giro de la junta**
- La distancia d_i es la distancia desde la intersección de $\hat{x}_{i-1}$ y $\hat{z}_i$ hasta el origen del marco colocado en la i-ésima junta, con su dirección positiva colocada a lo largo del eje $\hat{z}_i$. Este parámetro es conocido como ***offset* de la junta**
- El **ángulo de la junta** ϕ_i es el ángulo desde $\hat{x}_{i-1}$ hasta $\hat{x}_i$, medido sobre el eje $\hat{z}_i$

Estos parámetros constituyen los parámetros de Denavit-Hartenberg. En el caso de los manipuladores seriales o brazos robóticos los parámetros a_{i-1}, α_{i-1} y d_i son constantes definidas por su construcción y ϕ_i es la única variable medida en su junta. El vector ϕ se relaciona con la configuración de las juntas como

$$\phi = \mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_N \end{bmatrix} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix}. \tag{29}$$

6.8. Espacio de configuración, de tarea y de trabajo

El espacio de configuración es un espacio de tamaño n que contiene todas las posibles configuraciones, también conocido como espacio-C. La representación de este espacio depende de la configuración del robot y sus actuadores. Por ejemplo para un actuador serial 2R su espacio de configuración es un toroide como se aprecia en la Figura 10 el cual aún es entendible por el ser humano.

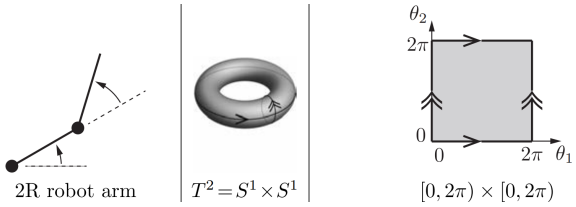


Figura 10: Espacio de configuración para un manipulador serial de dos juntas revolutas (2R)[13].

El espacio de tarea es el espacio donde una tarea del robot se puede representar de manera natural. Un ejemplo es un robot cartesiano para dibujar en una hoja de papel, en

donde su espacio de tarea es $\mathbb{R}^2$ o para un robot que debe de manipular un objeto rígido, su espacio de tarea es el espacio-C donde se representa tanto la posición como orientación del marco en el efector del final del robot. Este espacio se define bajo el criterio del diseñador y la tarea que busca cumplir [13].

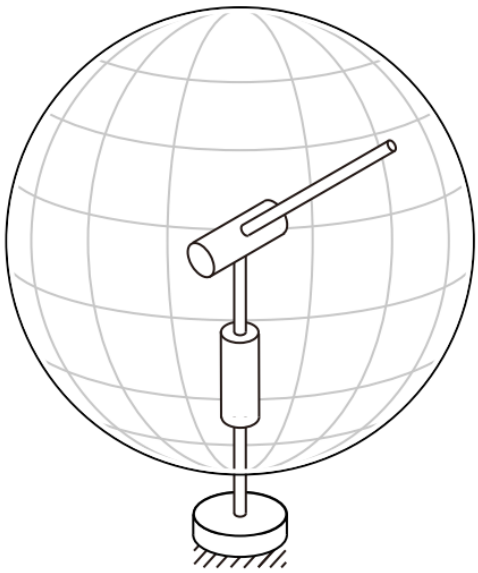


Figura 11: Espacio de trabajo para un manipulador serial de dos juntas revolutas (2R) [13].

En el espacio de trabajo se especifica la configuración que es capaz de alcanzar el efector final. Como se observa en la Figura 11 el robot 2R cuenta con un espacio de trabajo esférico contenido en el espacio $\mathbb{R}^3$. Los puntos de este espacio deben de ser alcanzables por el efector final como mínimo con una de las configuraciones del robot, aunque la mayoría son alcanzables con 2 o más. La definición de este espacio se basa en la estructura del robot.

La diferencia principal entre el espacio de configuración y los espacios de tarea y trabajo es su representación. El espacio de configuración deja de tener una representación intuitiva al aumentar las juntas, dificultando descripción de obstáculos y el trazar trayectorias adecuadas para una tarea en específico. Por otro lado, los espacios de tarea y trabajo son altamente intuitivos para el ser humano, ya que su posición puede ser representada por coordenadas cartesianas (x, y, z) y su orientación por ángulos de RPY ($Roll$,$Pitch$,Yaw), facilitando la colocación de obstáculos y la interacción del efector final con ellos. Teniendo esto en cuenta, el mapear puntos del espacio de trabajo al espacio de configuración facilita la interacción de los manipuladores con el mundo exterior [13].

6.9. Cinemática inversa para manipuladores seriales

Asumiendo que el efector final permite su representación con un vector x el cual está determinado por la cinemática directa $x = f(q)$ y que $f : \mathbb{R}^n \to \mathbb{R}^m$ es diferencial, se puede elegir un vector x_d como la configuración deseada del efector final. Definiendo $g(q)$ por el método de Newton-Raphson como $g(q) = x_d - f(q)$, con la meta de encontrar un valor q_d

que cumpla con [13]

$$g(q_d) = x_d - f(q_d) \tilde{=} 0. \tag{30}$$

Si se toma una suposición inicial q_o bastante cerca a la solución q_d, la cinemática por medio de la expansión de Taylor de primer orden queda

$$x_d = f(q_d) = f(q_o) + \left.\frac{\partial f}{\partial q}\right|_{q^o} (q_d - q_0). \tag{31}$$

en donde

$$J(q^o) = \left.\frac{\partial f}{\partial q}\right|_{q^o}.$$
$$\Delta q = (q_d - q_0).$$

Con $J(q^o) \in \mathbb{R}^{mxn}$, siendo este el Jacobiano evaluado en q_0, reescribiendo se obtiene

$$J(q^o)\Delta q = x_d - f(q_o). \tag{32}$$

Despejando para resolver Δq y estableciendo que $e = x_d - f(q_o)$ se obtiene

$$\Delta q = J^{-1}(q_o)e. \tag{33}$$

En 33 se supone que la matriz $J(q_o)$ es invertible, en caso de no serlo se puede utilizar una aproximación de esta inversa por el método de la pseudo-inversa del Jacobiano, el algoritmo de Levenberg-Marquardt (*Damped Leaste-Squares*), la transpuesta del Jacobiano o cualquier otro método que permita obtener su aproximación [13].

Tomando esta aproximación, se puede reescribir 33 como

$$\Delta q = \begin{bmatrix} J_p^{-1}(q_o) \\ J_o^{-1}(q_o) \end{bmatrix} \begin{bmatrix} e_p \\ e_o \end{bmatrix}. \tag{34}$$

donde el Jacobiano tiene una parte de posición J_p y una parte de orientación J_o, así como el error es tanto de posición e_p como de orientación e_o. En el caso donde el espacio de trabajo $x \in \mathbb{R}^3$, J_p está contenido en las primeras 3 filas de J y J_o en las últimas 3 [13].

El cálculo del error de posición simplemente es

$$e_p = p_d - po. \tag{35}$$

donde p_d y p_o son las posiciones deseada e inicial del efector final representadas en (x, y, z) como un vector columna del espacio.

Para el cálculo del error de orientación se debe transformar la matriz de rotación a cuaterniones y realizar la diferencia como

$$e_o = Q_d * Q_o^{-1}. \tag{36}$$

Tomando en cuenta 34, la cinemática inversa para el efector puede ser realizada para la posición únicamente sin cuidar la orientación al tomar solo la primera fila de 34, para la orientación sin cuidar su posición final al tomar solo la segunda fila de 34 o para una posición y orientación deseada 33 [13].

6.10. Mapas binarios en MatLab

Los *binaryOccupancyMap* o mapas binarios permiten crear un objeto mapa de $\mathbb{R}^2$, los cuales pueden ser utilizado para representar y visualizar el espacio de trabajo de un robot incluyendo los obstáculos y paredes. Al integrar a estos mapas valores de posición estimados con sensores, se pueden mapear los obstáculos [15].

Estos mapas pueden ser utilizados para algoritmos de robótica como la planificación de trayectorias, también se utilizan para aplicaciones de mapeo o cálculo de la localización de un agente. Las celdas en este puede tener valores de 0 o 1, donde 0 representa un punto libre y 1 un punto ocupado o porción de un obstáculo [15].

Los parámetros mínimos para crear uno son el ancho y largo del área de trabajo, en metros, así como la resolución. La resolución depende del tamaño del agente, buscando un equilibrio entre el desempeño del agente y una densidad mínima que lo permita. Otros parámetros que se pueden configurar son los marcos de referencia global, local y de la cuadricula. El marco global indica el punto $(0, 0)$ en el mapa, se puede agregar un offset a este si los sensores a emplear lo requieren. El marco local es el punto relativo al mapa pegado al agente [15].

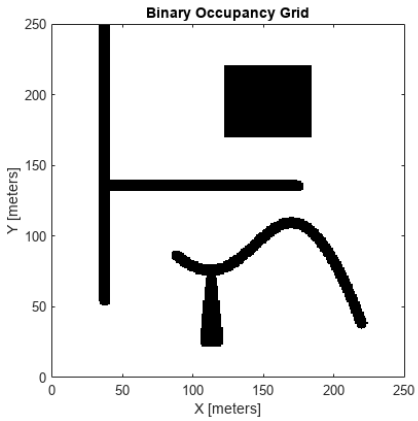


Figura 12: Ejemplo de un mapa binario de 250x250 m y sus obstáculos [15].

6.11. Planeación de trayectoria por medio de RRTs

La planeación de trayectorias se puede ver como un proceso de búsqueda en un espacio métrico $\mathbf{X}$, para una trayectoria continua que parte desde un punto x_{inicio} hasta una meta $\mathbf{X_{meta}} \subset \mathbf{X}$ o un punto x_{meta}. Donde $\mathbf{X} \in \mathbb{R}^2$ o $\mathbf{X} \in \mathbb{R}^3$ y es aplicable en robots con restricciones no holonómicas y planificación cine dinámica. Para la planificación se toma en consideración un espacio constante $\mathbf{X_{obs}} \subset \mathbf{X}$, este espacio contiene la información de los obstáculos, paredes o zonas donde no debe transitar el robot para llegar a la meta. A la vez se cuenta con el espacio $\mathbf{X_{libre}} = \mathbf{X_{obs}}^C$ que representa el espacio transitable por el robot [16].

Para un punto inicial x_{inicio} y un RRT T con K vértices, se puede construir como:

```
GENERATE_RRT(x_init, K, Δt)
1  T.init(x_init);
2  for k = 1 to K do
3      x_rand ← RANDOM_STATE();
4      x_near ← NEAREST_NEIGHBOR(x_rand, T);
5      u ← SELECT_INPUT(x_rand, x_near);
6      x_new ← NEW_STATE(x_near, u, Δt);
7      T.add_vertex(x_new);
8      T.add_edge(x_near, x_new, u);
9  Return T
```

Figura 13: Pseudo algoritmo de construcción de un RRT [16].

Se toma como el primer vértice de T el punto $x_{inicio} \in \mathbf{X_{libre}}$. Para cada iteración desde 1 hasta K se toma un valor aleatorio $x_{rand} \in \mathbf{X}$, luego se busca el vértice más próximo x_{cerca} en una distancia menor o igual a p. Tomando una entrada u se busca minimizar la distancia entre x_{cerca} y x_{rand} que solamente recorre en el espacio $\mathbf{X_{libre}}$. Al detectar colisiones este camino se elimina, al no contar con caminos que pasen por $\mathbf{X_{libre}}$ el nodo x_{rand} se descarta y se calcula otro. Al cumplir con la trayectoria se nombran los nuevos estados o vértices iniciales para el algoritmo y así calcular nuevos puntos aleatorios, agregando el camino anterior al árbol T. Al cumplir con el número de iteraciones o llegar a x_{meta}, se regresa el árbol en conjunto con la trayectoria deseada [16].

Entre las propiedades que presentan los RRT están:

- Se tienden a expandir a zonas inexploradas del espacio utilizado.
- La distribución de los vértices en el RRT se acerca a la distribución del muestreo, provocando un comportamiento consistente.
- Es relativamente simple, lo que permite el análisis de rendimiento.
- Puede ser implementado en una variedad de algoritmos para planificar la trayectoria.

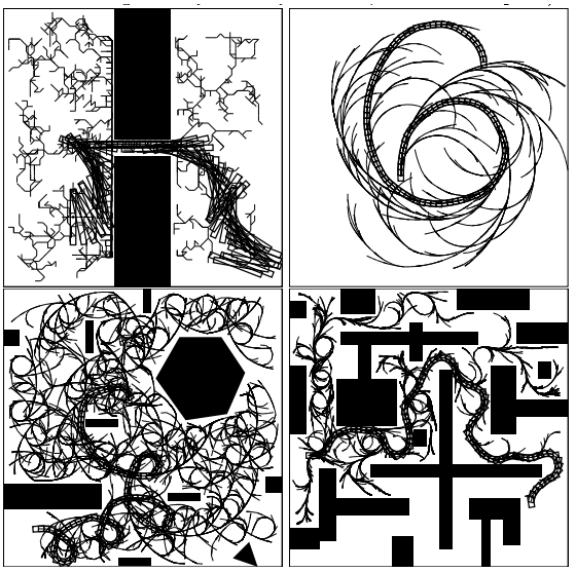


Figura 14: Ejemplos de las trayectorias generadas por un RRT en $\mathbb{R}^2$ [16].

Si bien las trayectorias planificadas con RRT son aplicables a diversos robots y evitan los obstáculos, usualmente no son óptimas al no ser las rutas más cortas que permiten llegar a la meta como se observa en la Figura 14. Para complementar la solución y optimizar la ruta se puede tomar en consideración variaciones de RRT como RRT* y RRT*-Smart. El primero optimiza para la selección de los vértices x_{cerca} a elegir desde los x_{rand} para siempre agarrar el de camino más corto, el segundo implementa el mismo cambio y a la vez agrega una optimización de trayectoria al eliminar vértices x_{rand} redundantes de la selección de nuevos vértices para la trayectoria [17].

6.12. Protocolos TCP/IP

Es una suite de protocolos de comunicación utilizados para interconectar dispositivos de red en Internet, también se utiliza en redes privadas. TCP e IP son los dos protocolos principales de la suite y estos presentan una capa de abstracción entre la capa de aplicación, conexión y envió. TCP/IP especifica cómo se debe de intercambiar la información por internet al proveer comunicación punto a punto e identificar cómo se desglosa está en paquetes, direcciones, transmisores, ruteo y recepciones hasta el destinatario [18].

TCP se encarga de definir cómo las aplicaciones pueden crear los canales de comunicación a través de la red, además de ensamblar los mensajes en pequeños paquetes antes de ser transmitidos por Internet. IP define cómo hacer el ruteo y direccionamiento de cada paquete para que este llegue de manera adecuada a su destino [18].

Comúnmente los protocolos TCP/IP contienen lo siguiente

- *Hypertext Transfer Protocol (HTTP).*
- *HTTP Secure.*

- Protocolos para manejar la transmisión de paquetes entre computadoras.

Las 4 capas del modelo TCP/IP, como se observa en 15, son las siguientes:

- **Application layer**, contiene los protocolos para manejar las conexiones en la red, permitiendo estandarizar el intercambio de información.
- **Transport layer**, se encarga de mantener viva la comunicación punto a punto a través de la red, estas pueden ser TCP o UDP.
- **network layer**, maneja los paquetes e interconexiones de redes para el transporte de los paquetes en la red.
- **Physical layer**, son los protocolos para la recepción y decodificación de los paquetes enviados como el Ethernet.

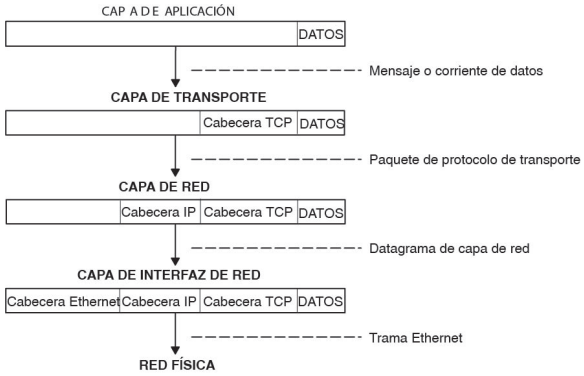


Figura 15: Intercambio de información desde un cliente hasta un remitente [19].

6.13. Pololu 3Pi+

Es un robot móvil sobre ruedas el cual está basado en el microcontrolador ATmega32U4 compatible con Arduino, este es programable por medio de USB y cuenta con librerías y ejemplos para operar el robot a distintos niveles. Tiene la capacidad de incluir otros periféricos sin alterar mucho su estructura, ya que incluye *headers* conectados a las líneas de potencia y I/O del microcontrolador [20].

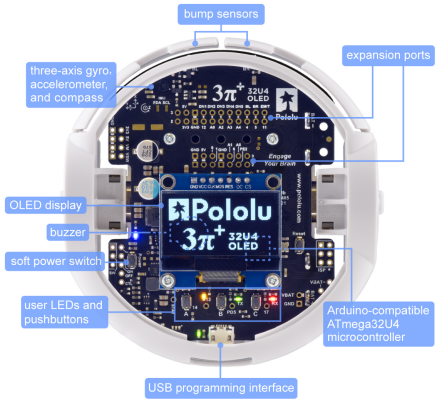


Figura 16: Vista superior del Pololu 3pi+ junto con sus características [20].

Además, incluye 2 puentes H y *encoders* de cuadratura para un control de lazo cerrado de los motores. Integra un módulo IMU completo con un acelerómetro de 3 ejes, un giroscopio y magnetómetro, 5 sensores reflectivos orientados hacia abajo para el seguimiento de líneas o detección de bordes, bumpers derecho e izquierdo en la parte frontal del robot, entre otros periféricos utilizables como se ve en las Figuras 16 y 17 [20].

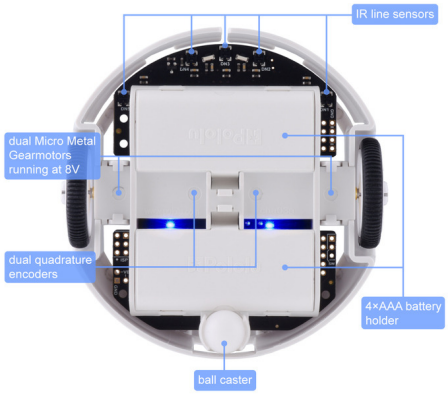


Figura 17: Vista inferior del Pololu 3pi+ junto con sus características [20].

6.14. TinyPICO

Es una placa de pequeño formato como se ve en la Figura 18, 18x32 mm, basada en el módulo ESP32 Pico D4 con la capacidad de ser programado en Espressif IDF, MicroPython y el IDE de Arduino. Cuenta con el chip CH9102F para la conexión *Serial2UART* con la computadora. El módulo ESP32 Pico D4 es de 32Bits de doble núcleo con una frecuencia de reloj de 240MHz. Tiene la capacidad de conexión Wifi a una frecuencia de 2.4GHz por el protocolo 802.11b/g/n, además de contar con conexión Bluetooth 4.2 en modo normal y *Bluetooth Low Energy (BLE)*. Entre los buses de comunicación disponibles esta I2C, SPI y UART [21].

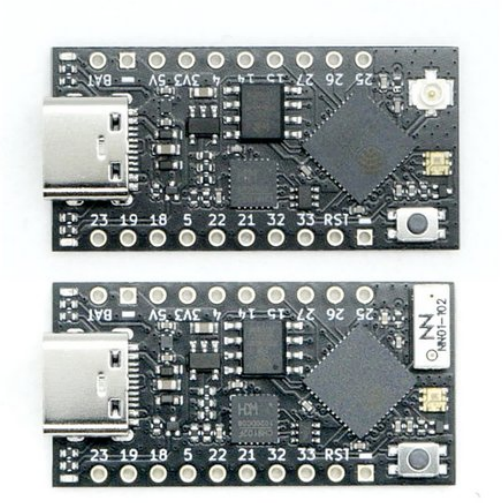


Figura 18: Placa de desarrollo basada en ESP32 TinyPico [21].

Cuenta con un modo de bajo uso de corriente con un consumo tan bajo como 20uA al estar en modo de reposo. Presenta dos líneas de potencia, una de 5V y una de 3.3V las cuales están aisladas. La línea de 5V esta activada únicamente al estar la placa alimentada por USB [21].

OpenMV Cam H7

Es una placa pequeña, de bajo consumo, basada en el microcontrolador STM32H743VI que permite la implementación de visión de máquina, procesamiento de imágenes y la implementación de algoritmos de análisis de imágenes utilizando redes neuronales. Esta cámara se puede programar con *scripts* de Python que permiten, en un alto nivel, el acceso a la salida de un algoritmo de visión de maquina o en bajo nivel el control directo de cada uno de los pines I/O en la placa [8].

Figura 19: Open MV CAM H7 [8].

El microcontrolador STM32H743VI corre a 480 MHz con 1MB de SRAM y 2MB de flash, con pines I/O que toleran 3.3V y 5V. Entre las interfaces que contiene están

- Una interfaz USB capaz de transmitir a 12Mbs con la computadora, identificado como un puerto COM desde la terminal.
- Un puerto para micro SD con una velocidad de lectura/escritura de hasta 100Mbs, lo que permite la captura de imágenes y procesamiento de visión de maquina desde la micro SD.
- Un bus SPI que transmite hasta 80Mbs en modo *half-duplex* y velocidades más bajas en *full-duplex*, utilizado para la transmisión de imágenes hacia un *shield* WiFi u otro microcontrolador.
- Otros buses de comunicación como I2C, CAN y UART para la comunicación con sensores y otros microcontroladores.
- Un ADC de 12 bits y un DAC de 12 bits.
- Interrupciones y PWM en los 10 pines de I/O en la placa, 3 de ellos tienen la capacidad de manejar servomotores.

Además presenta un módulo de cámara desmontable que permite el montar distintos tipos de cámara acorde a los requerimientos de análisis como lo es una cámara térmica [8].

Entre las aplicaciones que puede tener la OpenMV Cam H7 están

- Soporte para modelos de segmentación y clasificación de imágenes de TensorFlow Lite, tanto para su despliegue como para entrenamiento.

- Diferenciación de cuadros, permitiendo detectar el movimiento en una escena por los cambios entre sus cuadros.
- Rastreo de color y grupos de colores en una escena.
- Detección de cara, rastreo de ojos y detección de personas.
- Captura de imágenes y vídeo a distintas resoluciones, tanto a color como en blanco y negro.

Existen otras aplicaciones aparte de las listadas previamente. Las cuales pueden ser mezcladas y combinadas de distintas maneras para desempeñar una tarea en específico [8].

CAPÍTULO 7

Integración de la OpenMV Cam H7 al agente robótico

En este capítulo se presenta la integración de la OpenMV Cam H7 con el TinyS3 para la captura y envío de imágenes a través de WiFi. La conexión WiFi se establece mediante un *socket* TCP desde un *soft Access Point* creado en el TinyS3, garantizando así la integridad de los datos durante la transmisión.

Para la selección del bus de comunicación entre la OpenMV Cam H7 y el TinyS3, se realizaron pruebas de integridad de la imagen y se evaluaron los cuadros por segundo (FPS) alcanzados. Esto se hizo teniendo en cuenta que se utilizará para una transmisión en tiempo real desde la perspectiva del robot Pololu 3Pi+.

7.1. Metodología

Con el fin de aprovechar al máximo las capacidades del TinyS3, se optó por utilizar el *framework* ESP-IDF en PlatformIO. Este *framework* permite la configuración a nivel de hardware de los buses, periféricos y módulos de comunicación utilizando parámetros base propuestos por el fabricante, lo que garantiza su correcto funcionamiento y facilita la transición a los parámetros específicos requeridos por el proyecto. Además de permitir la configuración a bajo nivel, este *framework* permite la integración de *FreeRTOS* en el proyecto, lo que facilita la ejecución de múltiples procesos en tiempo real. En 15.2 se encuentra el repositorio del proyecto.

7.1.1. Selección del bus de comunicación entre el TinyS3 y OpenMV Cam H7

Para la selección del bus de comunicación se tomó en consideración la velocidad máxima de transferencia, el modo de transmisión y la cantidad de pines a utilizar. Tomando como base los buses que soporta la OpenMV Cam H7, se comparó UART y SPI en el Cuadro 1.

Bus	Velocidad de transferencia	Modo de transmisión	Pines
UART	hasta 7. 5Mb/s	*Half y Full duplex*	2
SPI	hasta 80 Mb/s	*Half y Full duplex*	5

Cuadro 1: Comparacion de buses

La imagen utilizada tiene dimensiones de 80×60 píxeles en blanco y negro, y se envía como un vector unidimensional de 4800 bytes. La selección del bus a utilizar se basará en la mayor cantidad de FPS (cuadros por segundo), tanto en promedio como en su valor máximo, alcanzados durante la transmisión y en la fiabilidad a lo largo del tiempo.

7.1.2. Configuración del TinyS3

La configuración del TinyS3 comienza con la declaración de tres funciones para crear el *soft Access Point (AP)*. Estas funciones son:

```
1 static void wifi_event_handler(void* arg, esp_event_base_t event_base,
    int32_t event_id, void* event_data);
2 void wifi_init_softap(void);
3 void nvs_init(void);
```

Código 7.1: Funciones para configurar el WiFi

La función **nvs_init()** que se encuentra en el Código 7.1 tiene la responsabilidad de inicializar el *Non Volatile Storage* o NVS. El proceso comienza con la inicialización del NVS y luego se verifica si hay páginas libres disponibles o si la versión es la más actual. En caso que no se cumpla alguna de estas dos condiciones, se borra el NVS y se inicializa nuevamente. Esta verificación de páginas libres se realiza para evitar la sobre escritura en sectores de memoria durante la ejecución del código, lo cual podría provocar errores de ejecución o la corrupción de información. Al finalizar esta configuración se retorna una respuesta en la consola para facilitar la detección de errores.

Una vez configurado el NVS, se procede a crear el *handler* para el punto de acceso (AP), el cual se configura mediante la función **wifi_event_handler(...)**. Este *handler* se encarga de imprimir en la consola cuando un dispositivo se conecta o desconecta de la red, lo que permite realizar un seguimiento de errores y asegurar una conexión adecuada con el TinyS3.

Para crear el punto de acceso (AP), se utiliza la función **wifi_init_softap()**. En esta función, se inicia creando la estructura de red necesaria para gestionar las conexiones y se inicializa el *stack TCP/IP*. Posteriormente, se establece un bucle de eventos para gestionar eventos basados en la recepción de mensajes o solicitudes de envío a direcciones específicas.

Una vez configurado esto se procede a crear la interfaz de red AP, la cual asigna los parámetros necesarios para que el TinyS3 funcione como un punto de acceso cuando se configure el subsistema WiFi. A continuación, se inicializa el subsistema WiFi con las configuraciones básicas y se registra el *handler* previamente creado, marcando así la finalización de la configuración de la estructura de red. Luego, se continúa ajustando los parámetros de la red.

Se asignan valores a la SSID, la contraseña y se especifica el número máximo de conexiones permitidas. Con estos parámetros definidos, se configura el modo de la red, se aplican los nuevos parámetros y se realiza la inicialización. Al concluir esta función se imprimen en la consola los datos asignados a la red para su verificación y seguimiento.

Luego de haber configurado el submódulo WiFi, se continua con la creación del socket TCP y un *handler* el cual responde en base los mensajes enviados al TinyS3. Para su configuración se utilizan las siguientes funciones:

```
1 static void tcp_socket_init(void *arg);
2 void handle_socket(void *pvParameters);
```

Código 7.2: funciones para crear el socket TCP

La función **tcp_socket_init()** se encarga de crear y configurar el *socket* TCP. Esta función se asigna a una tarea. El proceso comienza declarando las variables y estructuras necesarias para configurar el *socket*. En ellas se especifica que se utilizará la familia IPV4 para el protocolo IP, se indica la dirección IP y el puerto que se utilizarán. Luego, se crea y configura un nuevo *socket* para utilizar la familia de direcciones IPV4 y se establece que el *socket* puede ser reutilizado una vez que se cierre una conexión, con el fin de evitar problemas relacionados con direcciones o puertos durante la ejecución. A continuación, se enlaza el *socket* a la dirección IP y al puerto previamente declarados, y se coloca en modo escucha. De esta manera el *socket* está listo y atento para recibir conexiones entrantes.

Una vez que el socket está en modo escucha, se entra en un bucle infinito. En este bucle se crea una tarea cada vez que se acepta una conexión y se obtiene la dirección IP del cliente correspondiente. La tarea ejecutará la función **handle_socket(...)**.

En la función **handle_socket(...)**, se recibe el parámetro pasado en la creación de la tarea y se convierte en un entero. Luego se ingresa en un bucle infinito. Dentro de este bucle, se crea un búfer temporal donde se almacenarán los mensajes recibidos del cliente y se verifica su longitud. Si la longitud es menor a 0, indica un código de error durante la recepción y se sale del bucle. Si es igual a 0, significa que la conexión se cerrará y también se sale del bucle. Si la longitud es mayor a 0, se realiza una comparación basada en el primer byte recibido del mensaje. Después de la comparación se puede realizar una de las siguientes acciones:

- Encender o apagar el Pololu 3Pi+.
- Solicitar el último fotograma capturado al cliente.
- Cambiar la configuración de los servomotores en función de valores en bruto.
- Cambiar la configuración de los servomotores utilizando coordenadas específicas.
- Solicitar el dato en bruto del voltaje de la celda.
- Actualizar las velocidades de las ruedas del agente.

- Solicitar el último valor estimado del agente, calculado con odometria.

Al momento de salir del bucle se cierra el *socket*, finalizando la comunicación con el cliente y esperando una nueva conexión, eliminando la tarea creada de paso.

Al terminar de configurar el submódulo de WiFi y *socket* se continua con la configuración del bus SPI, este permite la comunicación del TinyS3 con la OpenMV Cam H7. Para su configuración se cuenta con las siguientes funciones:

```
1 void SPI_init_config(void);
2 static void AskForPicture(void *arg);
```

Código 7.3: Funciones para configurar el SPI

SPI_init_config() se encarga de configurar la interfaz SPI. Comienza declarando una estructura que contiene los pines a utilizar para MISO, MOSI y CLK en el SPI, además de deshabilitar la señal para mantener el puerto en caso de colisiones, la protección de escritura y el tamaño máximo del mensaje que se espera recibir o enviar. Luego, se declara una estructura que contiene los parámetros para el modo esclavo del SPI. En ella se indica el tipo de comunicación, el pin utilizado para CS, la cantidad máxima de transmisiones continuas que pueden presentarse y los *callbacks* en el Código 7.4 que se ejecutan antes y después del envío de datos.

Una vez definidas ambas estructuras, se procede a inicializar la interfaz. Esto se hace indicando el canal de DMA (*Direct Memory Access*) como selección automática y configurando la interfaz según las estructuras previamente definidas.

Finalmente, se configura el pin que se utilizará para el *Handshake* como una salida digital y se coloca en estado alto después de su configuración.

Los *callbacks* utilizados antes y después del envió de datos son:

```
1 void my_pre_cb(spi_slave_transaction_t *trans);
2 void my_post_cb(spi_slave_transaction_t *trans);
```

Código 7.4: Callbacks del SPI

Estas funciones se encargan de encender y apagar, respectivamente, el pin de *handshake* al comenzar una transacción.

En la función **AskForPicture()**, se realiza la solicitud de una imagen a la OpenMV Cam H7 a través de SPI. El proceso comienza ingresando a un bucle infinito en el que se verifica constantemente el estado de una bandera y *semaphore* para determinar si se debe iniciar con la captura de imágenes.

Cuando se requiere la imagen, se declara una estructura para configurar la transacción SPI. Esta estructura se inicializa con valores predeterminados en todos sus campos y se llena el búfer destinado a la recepción del mensaje con un valor por defecto. Luego, se establece la longitud de la transacción en bits y se indica el búfer donde se almacenarán los datos recibidos. Una vez que todo está configurado, se inicia la transacción utilizando la configuración previamente definida para SPI. Al finalizar la transacción, el último fotograma se copia a un arreglo utilizado durante el envío por WiFi.

Fuera de la verificación de la bandera se incluye un retraso de 100 ms. Este retraso tiene la función de reiniciar el "watchdog timer"del TinyS3, el cual no se activa durante la ejecución de estructuras de control y este controla el tiempo de solicitud de fotograma, limitando la actualizacion con la camara a un maximo de 10 FPS.

Luego de tener los datos de la transacción del SPI en el búfer, el siguiente caso en la función **handle_socket(...)** envía los datos al cliente bajo demanda:

```
if (xSemaphore2 != NULL && xSemaphoreTake(xSemaphore2, (TickType_t)
    portMAX_DELAY) == pdTRUE)
{

    int to_write = BUFFER_SIZE;
    int written = 0;

    while (to_write > 0)
    {
        written = send(sock, &rx_buffer, to_write, 0);
        if (written < 0) {
            #ifdef testeo
                ESP_LOGE(TAG, "Error occurred during sending: errno %d",
    errno);
            #endif
            to_write = 0;
        }
        else {
            #ifdef testeo
            ESP_LOGI(TAG, "Enviados %d bytes", written);
            #endif
            to_write -= written;
        }
    }

    xSemaphoreGive(xSemaphore2);
}
```

Código 7.5: Envío de datos por TCP

Este caso se ejecuta dentro de la tarea del *handle* WiFi. En esta función se comprueba si se puede realizar el envió o espera hasta poder realizarlo, cuando es capaz de realizarlo toma el *semaphore* para realizar de manera segura la transacción.

Primero, se declara una variable de tipo entero para almacenar el tamaño del búfer que contiene la información a enviar y otra variable para determinar cuantos bytes de información se han enviado. Luego, se entra en un bucle donde se verifica si el valor del búfer es mayor a 0. Si es mayor a 0, se envía la información al cliente conectado y se registra la cantidad de bytes escritos en la variable designada. Luego, se resta esta cantidad de bytes del valor de la variable que almacena el tamaño del búfer a enviar. Este proceso continúa hasta que se hayan enviado todos los datos de manera correcta. Una vez que todos los datos se han enviado correctamente, se libera el *semaphore* y se sale del *handle* WiFi.

Sin embargo, si se producen errores durante la transmisión, se muestra el mensaje de error en la consola y se coloca a 0 los datos pendientes por enviar, cancelando prematuramente el envió.

7.1.3. Configuración de la OpenMV Cam H7

La configuración de la cámara se realiza en MicroPython, como se observa en el siguiente fragmento de código:

```
import sensor, image, pyb
from pyb import SPI

RED_LED_PIN = 1

sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.QQQVGA)

handshake = pyb.Pin('P4',pyb.Pin.IN)
pyb.LED(RED_LED_PIN).on()
sensor.skip_frames(time=1000)

interface = pyb.SPI(2, SPI.MASTER,baudrate=40*1000*1000, polarity = 0, phase = 0)
cs_pin = pyb.Pin("P3",pyb.Pin.OUT_PP)
```

Código 7.6: Setup de imagen para la OpenMV Cam H7

Se importan las librerías necesarias para utilizar la cámara e interpretar los datos, así como el módulo SPI para la comunicación con el TinyS3.

Luego se reinicia el sensor de la cámara, que es un modelo OV7725, para establecer el formato de imagen y la resolución del fotograma a capturar. El formato utilizado es "GRAYSCALE" (escala de grises) y la resolución del fotograma se establece en "QQQVGA".

A continuación, se configura el pin "P4" como entrada, el cual se utilizará para el *handshake* entre dispositivos, y se enciende el LED rojo de la cámara. Este LED sirve como indicador visual de que la cámara se ha configurado correctamente. Para estabilizar la cámara, se descartan las primeras 1000 capturas realizadas.

Luego, se configura el módulo 2 del SPI en modo maestro con una velocidad de transferencia (baudrate) de 40 MHz y una polaridad y fase de 0. El pin CS de este módulo se asigna al pin "P3" y se configura como una salida de empuje, lo que permite controlar la línea y dictar su valor bajo demanda.

Con la configuración previamente realizada, se crea un bucle infinito como se muestra en el Código 7.7. En este bucle, el dispositivo responderá al *handshake* del TinyS3.

```
while(1):
    pyb.LED(RED_LED_PIN).off()
    if(handshake.value() == 0):
        envio = sensor.snapshot()
        cs_pin.low()
        interface.send(envio)
        cs_pin.high()
        #print("Enviado")
```

Código 7.7: Bucle para el envio de imagen

Cuando el pin de *handshake* toma un estado bajo, se captura el fotograma en ese momento y se coloca en nivel bajo el pin CS del SPI. Luego, se envía la imagen completa y se coloca en alto el pin CS para liberar el bus. La implementación del *handshake* agrega una capa de seguridad a la comunicación aumentando su efectividad y seguridad.

7.1.4. Obtención de imagen como cliente

Para obtener las imágenes como cliente es necesario conectarse a la red creada por el AP del TinyS3, asegurándose que la conexión sea exitosa como se observa en la Figura 20.

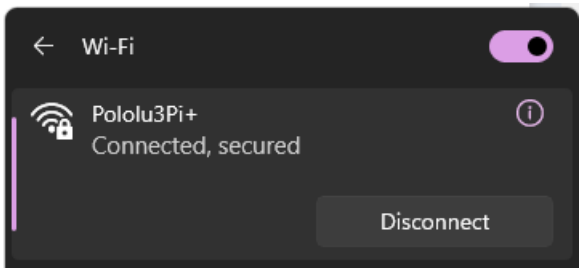


Figura 20: Conexión a la red creada por el TinyS3.

Al realizar la conexión, se cuenta con acceso al *socket* creado en el TinyS3. Se presenta un ejemplo de conexión y petición de imagen en Matlab, pero la obtención de imagen se puede realizar desde múltiples plataformas siguiendo la estructura planteada.

```
agente = tcpclient("192.168.4.1",3333);
write(agente,uint8('A'));
recov = read(agente,4800);
transf = transpose(reshape(recov,80,60));
imshow(uint8(transf));
clear tcpclient;
```

Código 7.8: Conexión y petición de imagen

Los paso a seguir son:

1. Se crea un objeto TCP que se conecta a la dirección IP y al *socket* configurados durante la creación del socket TCP en la función **tcp_socket_init()**.
2. Se envía el caracter 'A' como byte para solicitar la imagen.
3. Luego del envió se realiza una lectura del puerto esperando recibir 4800 bytes, que representan la imagen completa.
4. Al obtener la imagen se realiza una modificación en sus dimensiones. Dado que la imagen se recibe como un vector de una dimensión, se reorganiza como una matriz de 2 dimensiones de 80×60.
5. En caso de que la imagen aparezca espejada, se aplica una transposición para corregir la orientación.
6. Se muestra la imagen procesada en la interfaz gráfica.

7. Al terminar el proceso, se cierra el socket para finalizar la conexión con el TinyS3 y así liberar recursos para otros posibles clientes que deseen acceder a la información.

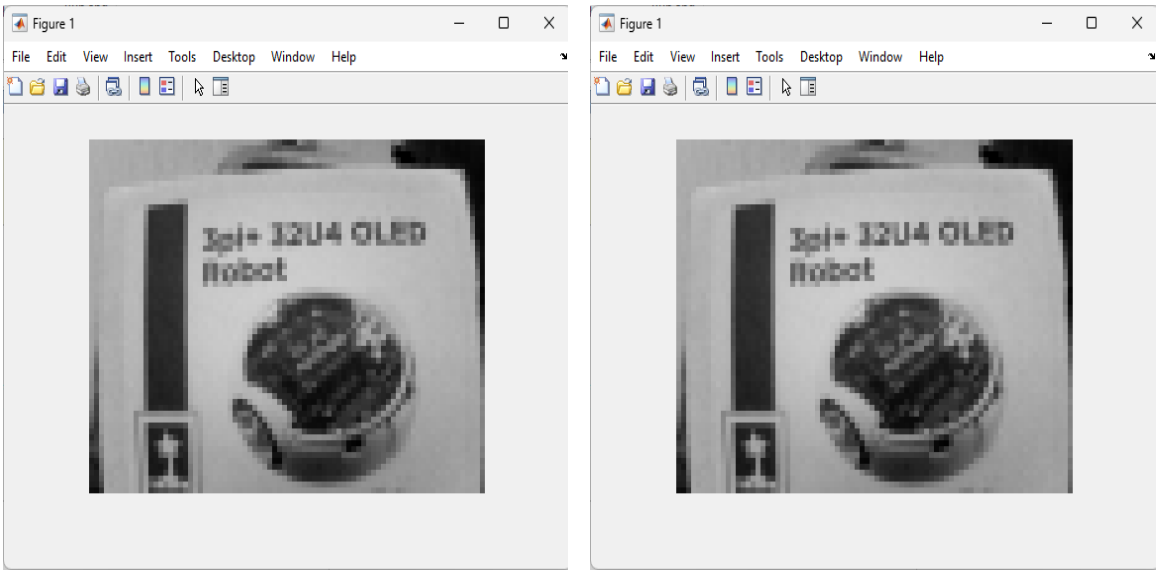
7.2. Resultados y discusión

Los valores de FPS obtenidos y velocidad de transferencia máxima, con UART y SPI, eran de:

Bus	FPS máximos	FPS promedio	Velocidad de transferencia máxima
UART	2.1	0.4	5 Mb/s
SPI	8.1	4.6	40 Mb/s

Cuadro 2: FPS obtenidos por bus de comunicación

Los valores máximos de transferencia reportados en el Cuadro 2 son aquellos que proporcionaron una imagen fiel a la capturada por la Open MV Cam H7, la menor cantidad de pausas o retrasos en la transmisión y la menor cantidad de errores en la imagen mostrada.



(a) SPI. (b) UART.

Figure 21: Comparación de capturas enviadas por UART y SPI.

Realizando las capturas de imagen con cada bus se puede observar en la Figura 21 que la calidad y detalles en ambas imágenes es el mismo, la diferencia son los FPS alcanzados y la fluidez de la transmisión (*livestream*).

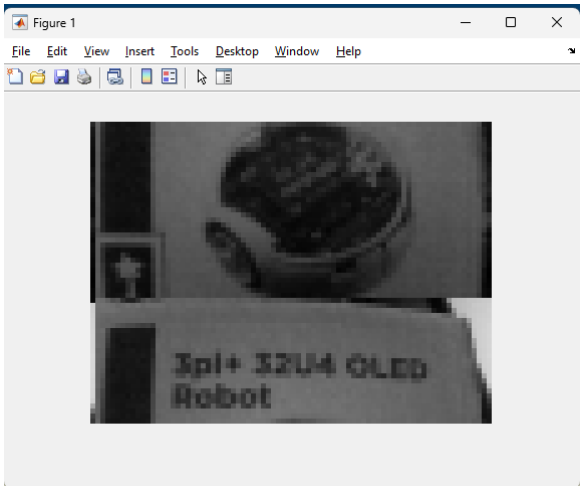


Figura 22: Artefacto presentado durante la transmision usando UART.

La transmisión generada con el bus SPI fue fluida y no presentó muchas pausas, distorsiones ni artefactos en la imagen. Por otro lado, la transmisión generada con UART mostró varias pausas, una fluidez de vídeo baja y presentó artefactos como se observa en la Figura 22, donde los datos de la imagen anterior se mezclaron con la actual.

Este artefacto o mezcla de imágenes fue causado por la pérdida de sincronía entre el TinyS3 y la OpenMV Cam H7. En algunas ocasiones, cuando este artefacto se presentaba, lograban resincronizarse y la transmisión podía continuar sin problemas. Sin embargo, cuando la falta de sincronía persistía durante un tiempo considerable el TinyS3 detenía la transmisión debido a un error en el envío hacia el cliente, lo que resultaba en el cierre de la conexión. Para resolver esto, era necesario reiniciar el agente para resincronizarlos.

Luego de estas pruebas se determinó que el bus a utilizar para comunicar el TinyS3 con la OpenMV Cam H7 era SPI, ya que alcanzó mayores FPS de forma estable y no presentó artefactos ante un prolongado uso.

Al realizar múltiples corridas de prueba con el agente (Pololu) alejándose del cliente en la plataforma de pruebas de Robotat y utilizando el bus SPI como medio de comunicación, se determinó que no existe pérdida de datos ni aumento de latencia en la transmisión, ya sea en el interior del aula donde se encuentra la plataforma de pruebas del Robotat o en cualquier ubicación del cliente y el agente. El tamaño de la plataforma es de aproximadamente 5×4 metros, y las mesas donde se pueden colocar los clientes están a unos 2 metros aproximadamente del borde de la plataforma de pruebas. Esto significa que la mayor distancia entre el cliente y el agente es de casi 7 metros, lo que mantiene la distancia de conexión efectiva entre el agente y el cliente dentro del rango de los 10 metros. El hecho de que no exista latencia, incluso estando muy cerca del rango máximo en algunos puntos, se debe a la falta de barreras físicas entre los agentes y los clientes.

CAPÍTULO 8

Control del agente robótico

En este capítulo, se introducen las funciones empleadas para dirigir el agente robótico mediante el TinyS3 y desde el cliente. Para controlarlo, aprovechamos la red WiFi preconfigurada que recibe directrices de los clientes. Además, establecemos una comunicación bidireccional con el agente mediante un bus UART. Los datos se transmiten por el bus codificados en CBOR desde el TinyS3, asegurando la integridad de la transacción. Desde el lado del cliente, se utiliza un conjunto de métodos desarrollados para un objeto personalizado que facilita la conexión con el agente y el control de los diversos módulos creados.

8.1. Metodología

Las funciones implementadas posibilitan el control de la velocidad individual de cada una de las ruedas del agente y facilitan la obtención de su posición estimada mediante odometría. Además, las funciones desarrolladas para gestionar el robot desde el cliente permiten ajustar las velocidades de las ruedas, controlar el manipulador serial, obtener los datos de odometria del agente y capturar los fotogramas de la OpenMV Cam H7.

8.1.1. Configuración del bus UART en el TinyS3

Para configurar el bus UART se utiliza la siguiente función:

```
1 void UART_init(void);
```

Código 8.1: Configuración del bus UART

En la función 8.1 se inicia creando una estructura del tipo **uart_config_t**, la cual recibe los valores para configurar el bus. Para la comunicación con el agente, se establece

un *baud rate* de 115,200, se configura el tamaño del mensaje en 8 bits, sin confirmación de paridad y con un bit de parada. Además, se deshabilita el control de flujo por hardware y se utiliza el reloj interno para la señal.

Con la estructura configurada, se procede a instalar el controlador con **uart_driver_install(...)**, donde se indica que se utilizará el UART 2 y se empleará solo un buffer de recepción. Posteriormente, se configuran los parámetros del controlador instalado con **uart_param_config(...)** y se especifican los pines que se utilizarán para este bus mediante **uart_set_pin(...)**, incluyendo únicamente pines RX y TX, ya que el resto de los pines no se utilizarán.

8.1.2. Envió de datos desde el TinyS3 al agente robótico

Luego de configurar el bus UART, se empleo un conjunto de funciones para codificar o decodificar las transmisiones realizadas con el agente. Las funciones son las siguientes:

```
1 void decodeCBOR(uint8_t* buffer, size_t len);
2 size_t encodeCBOR(void);
3
4 static void uart_rx(void * p_params);
5 static void uart_tx(void * p_params);
```

Código 8.2: Funciones para comunicarse con el agente

En el código 8.2, las primeras dos funciones se utilizan para decodificar y codificar los datos recibidos en el bus. Las otras dos funciones se utilizan para enviar y recibir datos del bus, estas se calendarizan en el TinyS3.

La función **decodeCBOR(...)** recibe un búfer de datos, el cual se espera que sea un contenedor de CBOR, y el tamaño del contenedor. Luego, inicializa el *parser* para el contenedor y determina si es o no un contenedor. Si no es un contenedor, la función sale, sin embargo, si es un contenedor, ingresa en este e itera sobre los valores contenidos, determinando si son o no flotantes. Luego, estos valores se decodifican y se asignan a un arreglo temporal de flotantes. Al terminar de decodificar 3 flotantes, estos se pasan a las variables para guardar la posición actual por del agente. Esta función se utiliza para recibir los valores de posición en X, Y y Theta del agente calculados mediante odometría.

La función **encodeCBOR(...)** es lo opuesto a la función anterior y codifica un búfer de datos para enviarlo como un contenedor de CBOR. Primero, se inicializa el *encoder* de CBOR, indicándole el búfer a utilizar y su tamaño. Luego, se crea un arreglo de tipo float y tamaño 2 para el encoder, el cual recibe las velocidades de las dos ruedas codificadas como flotantes, y se termina cerrando el contenedor. Al realizar estas tareas con éxito, se retorna el tamaño del contenedor creado.

Al contar con ambas funciones para codificar y decodificar en CBOR, se continúa con las funciones para transmitir los datos al agente.

Primero, se tiene la función **uart_tx(...)**, la cual transmite de manera periódica los datos al agente por UART. Se crea una variable para medir el tiempo que ha pasado desde la última lectura y se establece un tiempo de control, garantizando un envío regular de datos

al agente a su sistema de control. También se crea una variable para almacenar el tamaño del contenedor y se entra en un ciclo infinito donde se realiza el envió de datos. En cada envío de datos, se espera hasta que haya pasado el tiempo de control, se codifican las velocidades del agente y se guarda el tamaño del contenedor en su variable correspondiente. Al haber codificado las velocidades, estas se envían utilizando **uart_write_bytes(...)**, pasando el contenedor como mensaje y la variable del tamaño de contenedor como la cantidad de datos a escribir.

Luego, está la función **uart_rx(...)**, encargada de recibir los mensajes del agente, en este caso, la posición del agente calculada por odometría. En esta función se crea una variable para almacenar la cantidad de bytes recibidos en el búfer y se entra en un ciclo infinito. Dentro de este ciclo, se lee el bus constantemente y se compara si los bytes leídos son suficientes para determinar que se recibió un contenedor CBOR. Si los bytes son suficientes, se ejecuta la función **decodeCBOR(...)**, pasando el mensaje leído por el bus como el contenedor y el tamaño del mensaje. Luego, se limpia el bus y se queda a la espera de recibir un nuevo mensaje.

8.1.3. Control del agente desde el cliente

Para realizar el control desde el cliente se utiliza un objeto de la clase **PololuAgent**, esta es una clase personalizada y creada en el script "FuncionesControlPololu.py", esta clase contiene los siguientes métodos:

```
1  PololuAgent.connect()
2  PololuAgent.disconnect()
3  PololuAgent.on()
4  PololuAgent.off()
5  PololuAgent.arm_raw_config()
6  PololuAgent.arm_coord()
7  PololuAgent.battery_update()
8  PololuAgent.get_battery_voltage()
9  PololuAgent.get_battery_level()
10 PololuAgent.set_motor_speed()
11 PololuAgent.get_image()
12 PololuAgent.get_odo()
```

Código 8.3: métodos para el control del agente.

Si bien todos los métodos y la clase presentada están escritos en Python, al tratarse de una biblioteca simple para la comunicación con el agente, es posible traducirla a lenguajes como Matlab fácilmente.

Primero, al inicializar el objeto con la clase PololuAgent, el constructor recibe la dirección IP del agente y el puerto que utilizará, asignándolos a variables internas. También crea un búfer para el envío de datos, una variable para almacenar los valores de voltaje y el nivel porcentual de la batería, y las variables requeridas para la odometría del agente.

El método **connect()** se encarga de crear el socket con una dirección IPV4 y realiza los intentos de conexión al AP hasta lograrlo, o despliega el error encontrado en el proceso. El método **disconnect()** cierra el socket y envía el mensaje terminador para indicar a la placa que se cierra la conexión.

Ahora están los métodos **on()** y **off()**, estos envían un mensaje con el carácter requerido para que **handle_socket()** encienda o apague el agente. Este activa el switch digital en la placa principal del agente y coloca en estado alto o bajo el pin de CTRL del agente.

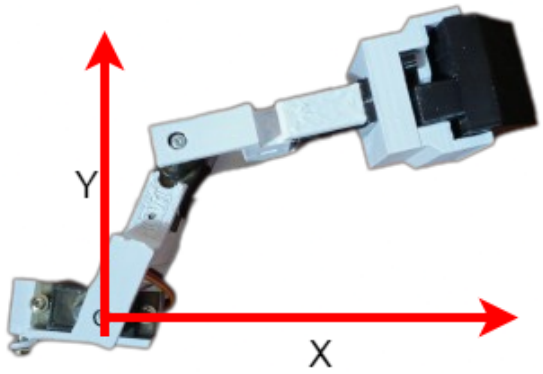


Figura 23: Ejes de referencia para el manipulador serial.

Para controlar el manipulador serial montado en el agente se crearon los métodos **arm_raw_config()** y **arm_coords()**. El primero permite enviar la configuración de las 2 juntas y el efector final en bruto, brindando una mayor libertad en el movimiento del manipulador serial. Con esta funcion se puede lograr posiciones que la cinemática inversa implementada no logra calcular o generar rutinas de movimiento fácilmente replicables a futuro. La segunda envía coordenadas para mover el manipulador serial, estas son interpretadas por el agente y, mediante cinemática inversa, coloca el efector final del manipulador serial en la posición solicitada. Los ejes de referencia para el manipulador serial se ven en la Figura 23.

Para leer el nivel de la celda utilizada para alimentar la placa principal, módulos y el agente robótico, se puede utilizar el método **battery_update()**. Este método envía el carácter de control al agente y recibe el valor en bruto de la celda al ser leído por el ADC. Luego, este valor se almacena en una variable interna del objeto creado y se divide para obtener una medida de voltaje o una medida porcentual. Para acceder al voltaje medido en la celda, se utiliza **get_battery_voltage()**, y para medir el voltaje como un porcentaje de la carga completa de la celda, se puede utilizar **get_battery_level()**. Estos métodos retornan los valores solicitados y pueden ser utilizados para determinar si es o no necesario regresar al agente a su estación de carga.

El método **set_motor_speed()** recibe las velocidades individuales de cada motor como un flotante y las convierte en bytes. Luego, estas velocidades se agregan al bufér del objeto junto con el carácter de control para enviarlo al agente, de modo que este pueda reconstruir los bytes recibidos como flotantes y asignarlos como nuevas velocidades en el agente.

Para obtener el último fotograma capturado por el agente se utiliza **get_image()**. Este método envía el carácter de control para solicitar la imagen y recibe 4800 bytes que describen el último fotograma capturado. Luego, estos valores se reordenan en un arreglo de NumPy para ser procesados por el cliente. Usualmente, después de la solicitud, se puede reorganizar el vector como una matriz de 60x80 y graficarla usando Matplotlib, lo que permite reconstruir una imagen desde el punto de vista del agente.

Para el seguimiento de trayectorias, el agente utiliza su odometría interna. Se puede acceder a los valores de esta utilizando el método **get_odo()**. Este método envía el carácter de control para solicitar los valores de odometría y recibe 12 bytes que representan 3 floats. Los valores obtenidos corresponden al valor de X, Y y theta del espacio de trabajo del agente. Estos valores se reconstruyen y se asignan a atributos internos que se pueden acceder utilizando **.odoX**, **.odoY** o **.odoTheta**, dependiendo de qué valor se requiera. Los valores almacenados por el agente están en metros, y el ángulo está en radianes.

8.1.4. Resultados y discusión

Los resultados presentados en esta sección se basan en los vídeos tomados para realizar las pruebas del manual de ensamblaje de las placas, el cual se encuentra en el repositorio con el nombre "ManualEnsamble.pdf". Pueden encontrarse los vídeos en la carpeta de Videos > Pruebas.

En el vídeo "Prueba_02_01", se observa al agente realizando una rutina establecida. Primero, rota en dirección antihoraria y luego en dirección horaria, se detiene y se mueve hacia adelante y, posteriormente, regresa, deteniéndose en su posición inicial. Esta rutina se llevó a cabo utilizando el método **set_motor_speed()**. Para la rotación, se establecieron velocidades de (-40, 40) y luego (40, -40) durante 3 segundos. Luego, se detiene, y se observa cómo se mueve hacia adelante y hacia atrás, utilizando velocidades de (30, 30) y (-30, -30) durante 5 segundos. Al final, el agente termina en la misma posición de partida. Estos resultados demuestran la capacidad del agente para moverse con precisión al recibir comandos desde el cliente, gracias a la baja latencia y la alta tasa de refresco de las variables de control.

En el vídeo "Prueba_02_03", se muestra el envío en tiempo real de lo que ve el agente. En el script de esta prueba, se solicitan los fotogramas para alcanzar de 8 a 10 FPS, los cuales son mensajes que contienen 4800 bytes para formar la imagen y se solicitan utilizando **get_image()**. Aunque existen pérdidas en la transmisión, estas se manejan adecuadamente, permitiendo retomar la captura luego de unos cuadros. Tanto la configuración de la red como del *socket* se ha realizado de manera adecuada, ya que permite retomar de manera segura el envío de múltiples mensajes.

En el siguiente video o en el siguiente enlace `https://youtu.be/6JNj5-_CVWA`, se puede observar al agente realizando transmisiones en vivo desde su punto de vista, además de seguir una trayectoria utilizando la odometría interna como referencia. Para obtener la imagen y actualizarla en tiempo real, se utiliza el método **get_image()**; para solicitar los valores de la odometría, se utiliza **get_odo()**, y para actualizar las velocidades de las ruedas con el controlador utilizado, se utiliza **set_motor_speed()**. Como se aprecia en el video, el agente es capaz de enviar la imagen y la odometría interna mientras recibe las velocidades de control para realizar la trayectoria parcialmente circular. Este último script requiere sincronización y envío como se presenta en los scripts de prueba incluidos en el repositorio, ya que es una combinación de los métodos implementados.

CAPÍTULO 9

Diseño electrónico y manufactura de la placa de control.

En este capítulo, se presentan los pasos para el diseño de la placa de control del Pololu 3PI+, la cual permite integrar el TinyS3, la OpenMV Cam H7 y un manipulador serial.

La selección de componentes para la placa se basó principalmente en la disponibilidad regional, y luego se buscaron alternativas que se ajustaran a los requerimientos del proyecto. El tamaño y la forma de la placa fueron determinados por el Pololu 3PI+, y los anchos de pista se tuvieron en cuenta en función de los materiales disponibles en el MakerLab, que es el taller del Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica donde se fabricaron las placas. Se utilizo Altium Designer para el diseño de los esquemáticos y placa de este proyecto.

9.1. Metodología

La conexión general entre módulos se presenta en la Figura 24. Estas conexiones comprenden alimentación, señales de control y buses de comunicación.

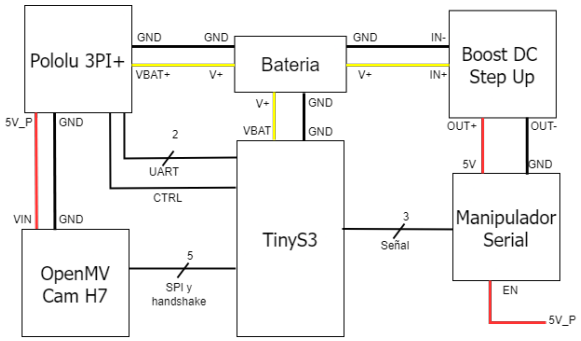


Figura 24: Diagrama general de conexiones entre módulos.

9.1.1. Selección de componentes

En el trabajo de graduación de Alejandro Gonzalez [12], se presentó una comparación entre la ESP32 Cam, la JeVois Cam y la OpenMV Cam H7. El análisis realizado tuvo en cuenta la arquitectura, la potencia consumida y las características de reconocimiento de imagen implementables. La OpenMV Cam H7 obtuvo la mejor puntuación en el análisis, por lo que se eligió como la opción a implementar en el proyecto. A pesar de que este modelo de cámara no se encuentra disponible a nivel nacional, el Departamento de Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala (UVG) dispone de varias de estas cámaras, lo que cubre la demanda necesaria para su implementación.

Al ser un proyecto destinado a aumentar las capacidades de agentes robóticos móviles y de enjambre, la mejor vía de comunicación es inalámbrica utilizando WiFi. Por ello, se tomó en consideración un microcontrolador central basado en ESP32, que contiene un módulo WiFi/Bluetooth y diversos periféricos en una misma placa de desarrollo.

Dado que el Pololu 3PI+ tiene poco espacio superior para colocar una placa de expansión sin modificar demasiado su centro de masa, se estableció como restricción el uso de una placa de pequeño formato. Inicialmente se consideró la TinyPico por sus capacidades de procesamiento, su módulo de carga y alimentación por batería LiPo, y sus 14 pines disponibles. Sin embargo, al momento de llevar a cabo este trabajo el Departamento de Mecatrónica solo contaba con el TinyS3 disponible para realizar las pruebas y el desarrollo de este trabajo. Dado que el proyecto busca utilizar equipo disponible a nivel nacional, se comparó esta placa con la propuesta inicial.

	TinyS3	**TinyPICO**
MCU	ESP32-S3FN8	ESP32-PICO-D4
Núcleos	2x Xtensa LX7	2x Xtensa LX6
Velocidad	hasta 240MHZ	hasta 240MHz
GPIO	17	14
WiFi	2.4Ghz b/g/n	2.4Ghz b/g/n
SRAM	512k	520k
FLASH	8 MB	4 MB
PSRAM	8 MB	4 MB

Cuadro 3: Comparación del TinyPico y el TinyS3

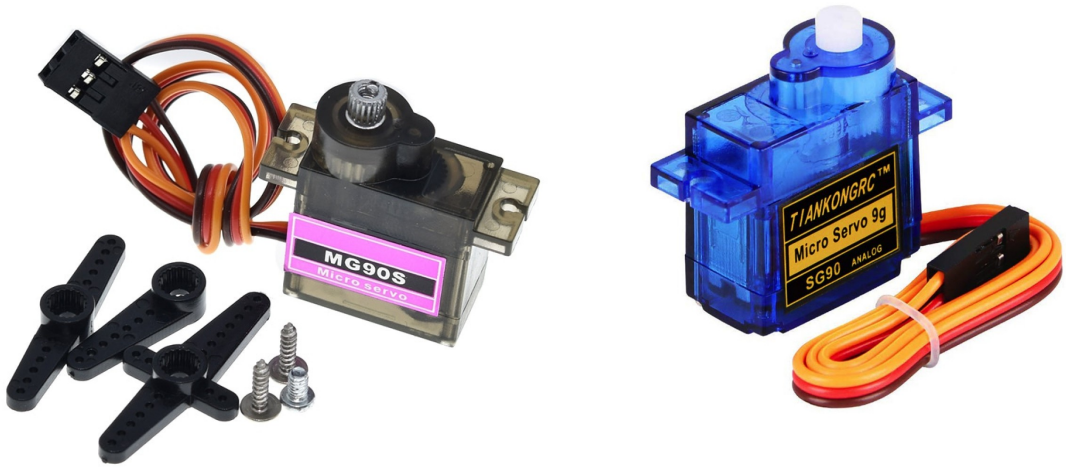
Observando el Cuadro 3, que es un fragmento del cuadro comparativo encontrado en [22], se puede notar que el TinyS3 presenta características similares o superiores al TinyPico, con el mismo tamaño. Por lo tanto, se optó por implementarlo en el trabajo. La elección de utilizar una placa de desarrollo en lugar de sólo el microcontrolador se basa en la facilidad de reparación de la placa de expansión y en la versatilidad que ofrece para su uso en otros proyectos del departamento.

Para el manipulador serial se consideró inicialmente el uso de motores a pasos o servomotores. En el siguiente Cuadro se presentan los parámetros considerados de ambas propuestas:

	Motores a pasos	**Servomotores**
Disponible localmente	No	Sí
Complejidad de uso	Alta	Baja
Precisión de movimiento	<0.01°	1°
Requiere de componentes extra	Si	No
Alimentación	3.9V - 5V	4.5V - 5V

Cuadro 4: Comparación entre motores a pasos y servomotores.

El manipulador serial implementado apunta a ampliar las capacidades del Pololu 3Pi+ de manera básica, permitiendo interactuar con objetos de bajo peso. Entre las interacciones esperadas están recoger objetos, empujarlos y moverlos a una posición definida, por lo cual no se consideró importante una alta precisión de movimiento. Al tomar en consideración los parámetros presentados en el Cuadro 4, se optó por implementar servomotores en el diseño final.



(a) MG90 [23]. (b) SG90 [24].

Figura 25: Servomotores considerados para el manipulador serial.

Al considerar a los servomotores de la Figura 5 como los actuadores del manipulador serial, a nivel local existen dos opciones con alta disponibilidad que son los SG90 y los MG90. Estos servomotores son de tamaño reducido y bajo peso. Las características consideradas para los mismos fueron:

	SG90	MG90
Voltaje de operación	4.8V - 5V	4.8V - 5V
Tipo de engranes	Plasticos.	Metálicos.
Dimensiones	22.2 x 12.2 x 28.5 mm	22.2 x 11.8 x 31 mm
Torque detenido	1.8 kgf-cm	3.0 kgf-cm
Peso	9 g	13.6 g

Cuadro 5: Comparación entre servomotor SG90 [24] y MG90 [23]

Al observar las características presentadas en el Cuadro 5, la opción seleccionada para el proyecto fue los servomotores MG90. Estos son ligeramente más grandes y más pesados pero poseen un mayor torque de salida y en estado detenido. Además, el uso de engranajes metálicos en lugar de plásticos le da peso a esta alternativa, ya que su desgaste con el tiempo será menor, permitiendo un uso prolongado antes de tener que cambiarlos.

Al seleccionar la celda que alimentará al agente y los distintos módulos, se optó por una 18650 ya que cuentan con una gran corriente de descarga base (superior a 5.2 A), además de tener un peso y forma que facilitan el equilibrio del centro de masa del agente. También son compatibles con el circuito de carga y descarga del TinyS3 y el voltaje que entrega es capaz de encender al Pololu 3Pi+ sin problema. Para seleccionar el modelo, se relacionó la capacidad con el costo por celda.

Capacidad nominal	Corriente de descarga máxima	Costo	mAh/Q
2600 mAh	5.2 A	Q59.00	44 mAh/Q
2500 mAh	7.5 A	Q69.00	36 mAh/Q
2500 mAh	30 A	Q75.00	33 mAh/Q
3000 mAh	7.5 A	Q75.00	40 mAh/Q
3500 mAh	7.5 A	Q159.00	22 mAh/Q

Cuadro 6: Análisis de costo por mAh en baterías 18650

Observando la relación mAh/Quetzal presentada en el Cuadro 6, se notó que las dos mejores alternativas son la versión de 3 Ah y la de 2.6 Ah, ya que ofrecen una mayor autonomía por Quetzal en los proyectos. Teniendo eso en cuenta, se analizó la mayor capacidad de descarga y la capacidad nominal de la celda, por lo cual se seleccionó la versión de 3000 mAh. La elección se debe a que su descarga máxima es mayor, lo que permite tener un factor de seguridad más amplio ante alguna eventualidad con la celda; y su capacidad nominal es mayor, lo que proporcionará una mejor autonomía al proyecto.

Luego de seleccionar la celda, se procedió a elegir el sistema de manejo y protección para la batería (BMS). Los requisitos establecidos para la selección del BMS incluyeron la capacidad de controlar una celda, entregar hasta 3 A en su salida y tener un formato compacto. A nivel local solo se encontró un modelo que cumplía con estas características y se muestra en la Figura 26.

Figura 26: BMS para una celda con salida de hasta 3 A [25].

Por último, se seleccionó el módulo elevador de voltaje para alimentar los servomotores utilizados en el manipulador serial. Si bien el Pololu 3Pi+ cuenta con una salida regulada de 5V, esta tiene una corriente limitada de aproximadamente 0.5 A. Dado que la OpenMV Cam es alimentada con este regulador, la corriente real para una operación segura es menor a 0.3 A lo que lo hace insuficiente para alimentar todo el manipulador serial.

Figura 27: Módulo elevador de voltaje a 5V.

Por ello, se tomó en consideración un módulo de salida fija a 5V, no regulable, con capacidad de entregar al menos 1 A en su salida y ser de pequeño formato. A nivel local, solamente se encontró un modelo[26] que cumple con estas especificaciones y tan solo mide 20×15 mm. El modelo seleccionado se presenta en la Figura 27.

Al tener todos los módulos seleccionados, se buscó en las librerías en línea de Altium Designer si estaban disponibles con *footprint* y modelo 3D. Sin embargo, ninguno de los módulos adquiridos a nivel local contaba con estas características. Por lo tanto, se creó una librería integrada de componentes donde se incluyeron todos los módulos adquiridos.

9.1.2. Requerimientos de potencia

Al contar con todos los módulos y componentes a utilizar, se continúa con los requerimientos de potencia de cada módulo a utilizar. Este análisis se realizó para determinar la autonomía del agente y la corriente máxima de descarga, con el objetivo de evitar sobrepasar

los niveles máximos soportados por la celda.

Módulo	Voltaje de alimentación	Corriente consumida	Potencia
MG90 (3)	5 V	1.5 A	7.50 W
TinyS3	3.3 V	0.34 A	1.12 W
Pololu	3.8 V	1.2 A	4.56 W

Cuadro 7: Análisis de potencia máxima por módulo

En el Cuadro 7, se observa el análisis de potencia y corriente consumida por cada módulo. La OpenMV Cam no se incluyó en el análisis ya que se alimenta con el regulador de 5V del Pololu 3Pi+, por lo cual la potencia consumida por este se incluye solo en el Pololu 3Pi+. Tanto el TinyS3 como el Pololu se alimentan con la celda 18650, por lo cual solamente se analizará su consumo de corriente para la autonomía y descarga máxima de la celda.

Por otro lado, los MG90 son alimentados por un módulo elevador de voltaje, presentado en la Figura 27, el cual debe entregar 7.5 W en su salida de 5V para los 3 servomotores. Con una eficiencia del 96 %, su entrada es de 7.82 W. Tomando un voltaje de batería promedio de 3.8 V, la corriente demandada por el módulo es de 2.06 A. Este valor se tomó en cuenta para calcular la autonomía y descarga máxima de la celda.

El total de corriente consumida entre los módulos es de 3.6 A, este siendo el valor máximo si todos los módulos estuvieran funcionando al máximo y al mismo tiempo. Sin embargo, esto es algo que difícilmente sucederá en la práctica, ya que la OpenMV Cam y el manipulador serial rara vez trabajarán en conjunto y la potencia reportada por los brazos es en condiciones de máxima capacidad. Para reflejar un uso más cercano a la realidad, se aplicaron factores a las corrientes consumidas, quedando:

Módulo	Corriente máxima	Factor	Corriente en la práctica
MG90 (3)	2.06 A	0.4	0.82 A
TinyS3	0.34 A	0.8	0.272 A
Pololu	1.2 A	0.75	0.9 A
Pololu + OpenMV Cam	1.2 A	0.85	1.02 A

Cuadro 8: Análisis de potencia práctica por módulo.

El factor aplicado al manipulador serial (MG90) se debe a que solo uno de ellos estará funcionando a media capacidad, siendo la junta de base del manipulador, mientras que la junta del codo y el efector final trabajarán en baja capacidad. Además, se tomó en consideración que los objetos manipulados son esponjas o pelotas de ping pong, por lo tanto, el cambio en la capacidad de los servomotores es mínimo. Luego, al aplicar el factor al TinyS3 se consideró que este únicamente aumenta su consumo de corriente al crear el servidor AP y enviar paquetes por WiFi. Por lo tanto, se asumió que el 80 % del tiempo demandará esa corriente, mientras que el resto del tiempo se utilizará para modificar datos en el cliente. Por último, el factor del Pololu 3Pi+ se aplicó teniendo en cuenta el hecho que se opera a bajas velocidades, no mueve elementos pesados y no utiliza la mayoría de periféricos que contiene. El factor aumenta al utilizar la cámara con el Pololu 3Pi+, el cambio no es drástico ya que esta consume menos de 0.15 A al estar en operación.

9.1.3. Requerimientos de tamaño y reglas de diseño

Para el diseño y consideraciones de tamaño se tomó como base los planos de la placa de control del Pololu 3Pi+ versión OLED, con revisión 3pi03b.

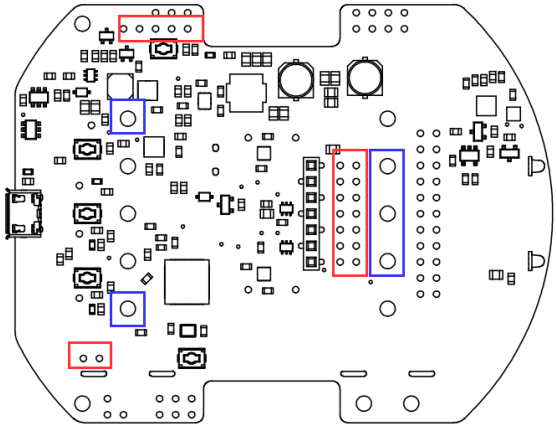


Figura 28: Vista superior de la placa de control del Pololu 3Pi+ [27].

Se consideraron múltiples formas para la placa de expansión, como rectángulos u óvalos. Al final, se optó por replicar la placa base del Pololu 3Pi+ [27] ya que esta queda montada sobre la carcasa exterior y disminuye los esfuerzos en los pines, además de aumentar la robustez del montaje.

Los conectores utilizados en esta placa se presentan en la Figura 28, resaltados en color rojo, mientras que los agujeros de montura y para futuras ampliaciones se encuentran enmarcados en azul. Con esto en mente, la disposición final quedó como se muestra en la Figura29. Este diseño 3D se exportó a formato STEP para generar la disposición y forma de la placa en Altium a partir de su forma 3D.

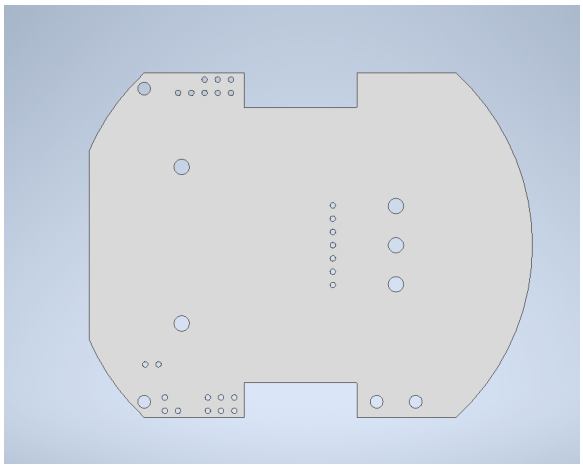


Figura 29: Disposición de la placa en vista superior.

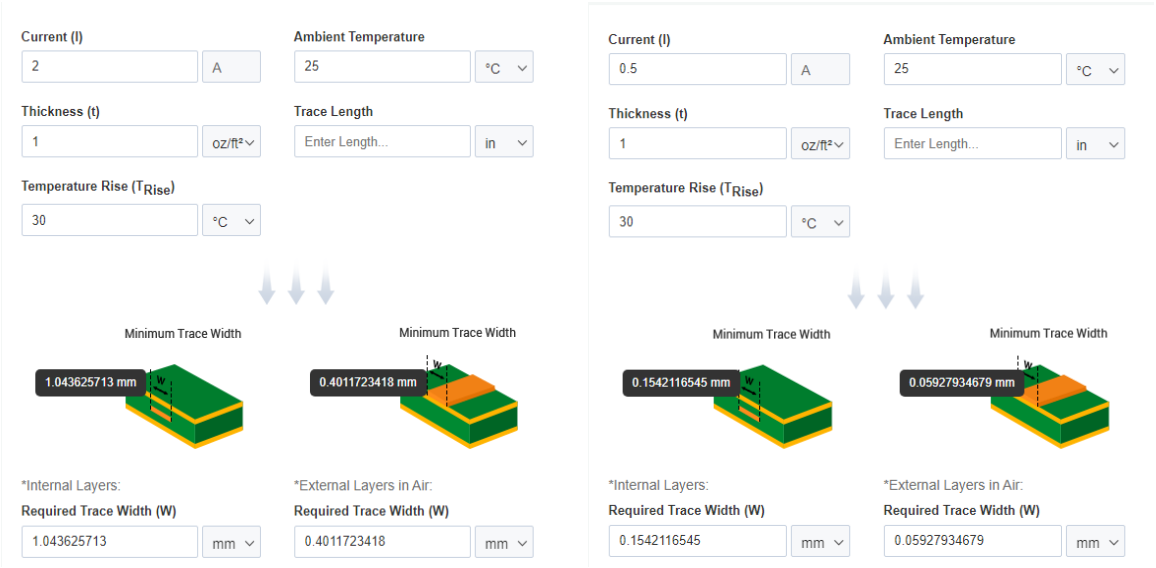
Para manufacturar esta placa, se tomaron en consideración los requerimientos de diseño proporcionados por el MakerLab que permite la fabricación de placas de cobre simples o de doble lado. Estas placas contienen 1 oz de cobre en cada cara y tienen un tamaño máximo de 9×12 pulgadas. Los requisitos proporcionados por el MakerLab son los siguientes:

Requisito	**mills**	**mm**
Ancho mínimo de pista	10	0.254
Ancho mínimo recomendado de pista	20	0.508
Espacio mínimo entre pistas	10	0.254
Ancho mínimo recomendado entre pistas	20	0.508
Tamaño mínimo de pad	agujero + 20	agujero + 0.508
Tamaño mínimo entre pads	10	0.254
Tamaño mínimo de pad para Vias	60	1.524

Cuadro 9: Requisitos del MakerLab para la manufactura de PCBs

Los requisitos presentados en el Cuadro 9 se colocaron en las reglas de diseño del proyecto creado en Altium.

Luego de definir las reglas para el diseño se procedió a calcular los anchos de pista requeridos. Se consideraron dos casos que engloban la mayoría de las pistas a utilizar. El primero involucra las pistas de alimentación para el Pololu 3Pi+ y los servomotores, en las cuales se espera conducir hasta 2 A de corriente. El segundo caso abarca las pistas de alimentación de módulos pequeños y señales de control, en las que se espera conducir hasta 0.5 A de corriente. Utilizando la calculadora en línea de PCB Way [28], que se basa en el estándar IPC-2221, se determinaron los siguientes anchos para las líneas:



(a) Ancho para 2 A [28]

(b) Ancho para 0.5 A [28]

Figura 30: Referencias de ancho de pista para el diseño [28].

Los valores obtenidos en la Figura 30 se utilizaron como referencia para modificar las reglas ya establecidas. Se conservó el valor mínimo de ancho de pista, separación entre pistas y *pads* propuesto por el MakerLab, ya que, como se observa en la Figura 30a, estos valores son mayores a los calculados. Los valores obtenidos en la Figura 30b se utilizaron para determinar el máximo admitido en las reglas. Dado que en Altium Designer se coloco el ancho sugerido en las reglas, a las líneas de potencia se les modificó el ancho manualmente al valor calculado en la Figura30b.

9.1.4. Modularidad y disposición de componentes

Al contar con los módulos seleccionados, además del tamaño y forma de la placa se realizó la colocación de los elementos en la placa. Para la colocación se siguieron las siguientes restricciones:

- El TinyS3 debe colocarse en la parte posterior del agente y centrado.
- La OpenMV Cam es un módulo que se puede montar y desmontar.
- El manipulador serial es un módulo que se puede montar y desmontar.
- La celda se debe instalar en la parte posterior del agente.
- Los conectores marcados en la Figura 28 no se pueden mover.

Tomando estas restricciones en mente se llevó a cabo una primera iteración.

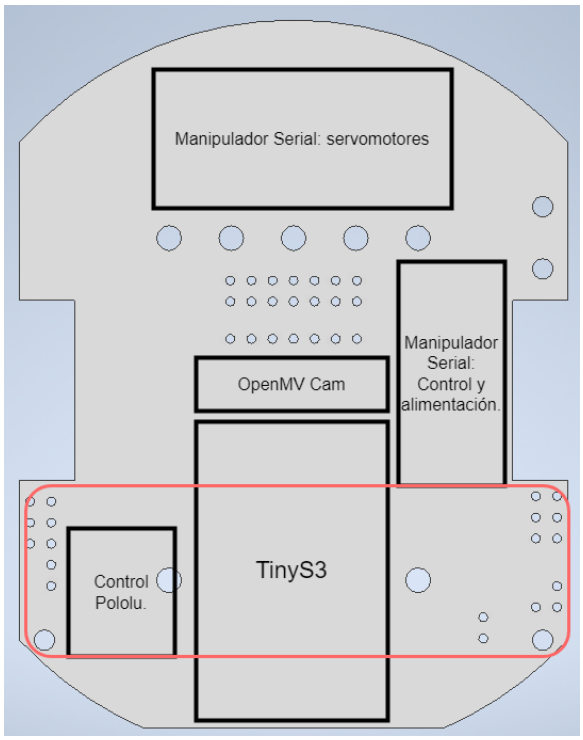


Figura 31: Disposición de los módulos en la placa usando tecnologia THT.

En la Figura 31 se muestra en negro la disposición de los distintos módulos colocados o conectados directamente a la placa principal. El manipulador serial se divide en dos partes: los servomotores, que forman el manipulador completo a montar en el Pololu 3Pi+, y el sistema de control y alimentación, que incluye el elevador de voltaje y las líneas de señal. En el módulo de control para el Pololu 3Pi+, se encuentran las conexiones para encender o apagar el agente además del circuito para medir la carga de la batería. En el módulo de la OpenMV Cam se colocan los headers en los cuales se conectará la cámara, que se monta sobre una placa que permite la conexión de alimentaciones y el bus de comunicación con el agente a 90°.

En rojo, se observa el módulo de la celda que incluye el BMS y el conector de la celda. Debido a la falta de espacio, este se colocó como una placa en cascada montada sobre el TinyS3 y conectada a la placa principal mediante un cable de 2 líneas.

Esta disposición inicial consideraba elementos pasivos y semiconductores THT (*Through-Hole Technology*). Sin embargo, al realizar las primeras versiones y ajustar las conexiones, se optó por cambiar a tecnología SMD (*Surface Mount Device*). Esto dio como resultado una nueva disposición.

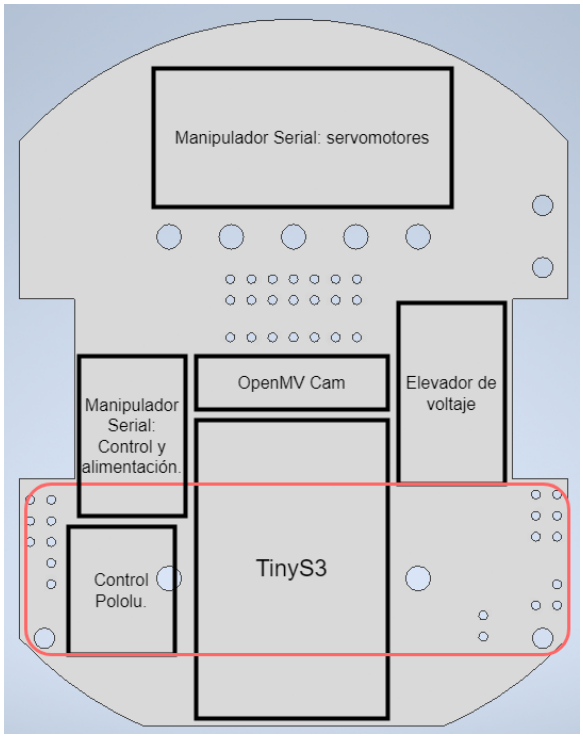


Figura 32: Disposición de los módulos en la placa usando tecnología SMD.

Al realizar el cambio a SMD, como se puede observar en la Figura 32, se distribuyeron de manera uniforme los componentes en el área central de la placa, lo que permitió dejar la zona frontal para un montaje seguro del manipulador serial.

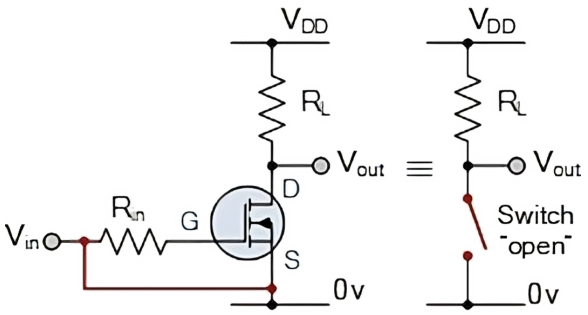


Figura 33: Configuración del MOSFET para utilizarlo como interruptor digital [29].

Los módulos de control en la Figura 33 son MOSFETs digitales que permiten encender o apagar los módulos. Tanto para los servomotores como para el Pololu se utilizó el MOSFET configurado como se ve en 33. La diferencia entre estos módulos es la señal para activarlos Vin. El interruptor para el Pololu 3Pi+ es manejado con un pin del TinyS3, mientras que los interruptores de los servomotores (manipulador serial) son manejados con la salida de 5V del regulador del Pololu 3Pi+.

9.1.5. Pines utilizados

Luego de determinar los módulos a utilizar, la placa y el bus de comunicación se continuó con la selección de los pines a utilizar. Se realizó una propuesta inicial previo a comenzar el ruteo y colocación de los elementos, pero esta era ineficiente y requería de pistas muy largas para conectar algunas señales de control. Por ello, con la disposición final de los módulos se adaptaron los pines de los módulos quedando:

Número de Pin	**Uso**
1	Manejo servomotor 1
2	Manejo servomotor 2
3	Manejo servomotor 3
4	Encendido agente
5	Lectura nivel de bateria
35	MOSI
36	MISO
37	SCK
34	SS
9	*Handshake* SPI
8	TX a Pololu
7	RX a Pololu

Cuadro 10: Pines utilizados en el TinyS3.

Número de Pin	Uso
0	MOSI
1	MISO
2	CLK
3	SS
3	*Handshake* SPI

Cuadro 11: Pines utilizados en la OpenMV Cam H7.

Tanto para la OpenMV Cam como para el TinyS3 quedaron libres los pines que no se enlistaron en los Cuadros anteriores, permitiendo expandir sus capacidades para futuros proyectos.

9.2. Resultados y discusión

Al tomar en consideración los factores empleados en el Cuadro 8 se calculó la autonomía esperada ante distintas combinaciones de módulos, dando como resultado:

Combinación	Consumo	Autonomía	Autonomía real
Pololu + TinyS3 (Mínimo)	1.172 A	2.56 Horas	1.92 Horas
Pololu + TinyS3 + OpenMV Cam	1.292 A	2.32 Horas	1.74 Horas
Pololu + TinyS3 + MG90	1.99 A	1.5 Horas	1.125 Horas

Cuadro 12: Autonomía del agente ante distintas combinaciones de módulos.

Los valores de autonomía presentados en el Cuadro 12 son ligeramente mayores a los que tendrá el agente. Esto se debe a que las condiciones para recargar el agente consideran un nivel de batería bajo cuando llega al 25 % de su capacidad. Por ello, se realizó el ajuste presentando una autonomía real que es el 75 % de la autonomía esperada. Los resultados de autonomía real obtenidos indican que se puede hacer uso de los agentes durante al menos 1 hora antes de suspender las pruebas y colocarlos a cargar.

Al realizar pruebas con todos los módulos funcionando al mismo tiempo, se observó que el BMS entraba en corto el circuito. Se realizaron múltiples pruebas con las combinaciones de módulos presentadas y solamente el Pololu 3Pi+ junto con el TinyS3 encendían con el BMS, cualquier otro elemento agregado no permitía que encendiera nada. Al revisar la corriente consumida y los niveles de voltaje durante el uso, se obtuvieron los valores esperados por combinación y estos no se disparaban al encender o apagar el agente por completo.

Al observar este fenómeno, se buscaron alternativas para el BMS, pero a nivel local solamente ese modelo estaba disponible para los requerimientos del agente. Por lo cual se analizaron las medidas de protección que este ofrece, siendo:

- Protección contra sobre voltaje: no exceder los 4.2 V al cargar la celda.
- Protección contra bajo voltaje: no permitir que la celda caiga por debajo de 3.2 V.

- Protección contra sobre corriente: no permitir que fluyan más de 3 A.

Observando las protecciones que este circuito ofrece, se logró determinar que los componentes previamente seleccionados ya implementaban estas medidas de seguridad. La protección contra sobre voltaje la brinda el sistema de carga del TinyS3, ya que deja de cargar la batería cuando se acerca a los 4.2 V. La protección contra bajo voltaje se implementa con el lector analógico del nivel de batería para el agente, este en inicio se dimensionó para detectar un nivel que permita al agente llegar seguro a su estación de carga, pero se le implementó un apagado de emergencia al estar por debajo de cierto nivel menor al nivel establecido para el retorno automático. Para la protección contra sobre corriente, en vez de implementar el BMS en la placa de la celda, se colocará un fusible de 3 A de acción rápida.

Al eliminar el BMS, se trabajó en las versiones finales de las placas en cada módulo. Para la placa principal de control se manufacturaron 4 versiones en total, para la placa de batería fueron 3 versiones, y la placa de la OpenMV Cam se siguió utilizando en su primera versión propuesta. Los modelos de estas placas vistos en Altium Designer quedaron de la siguiente manera:

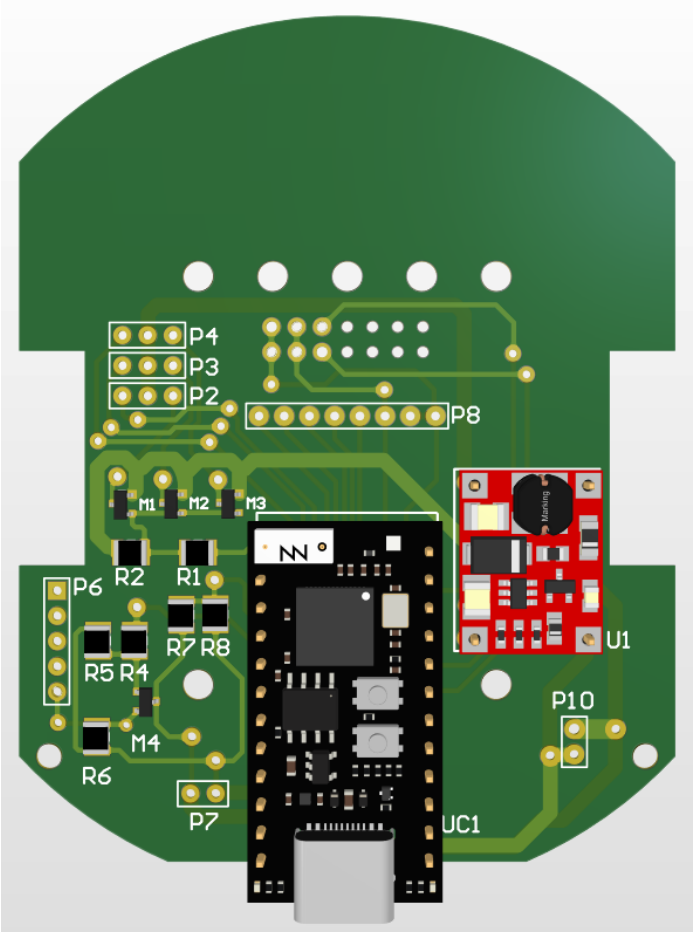
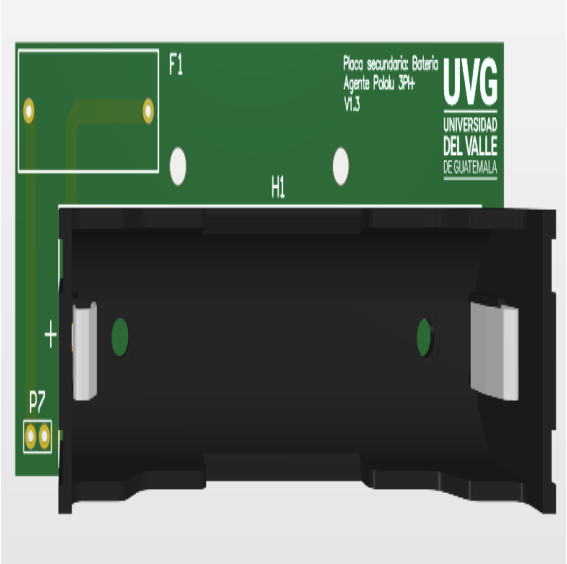
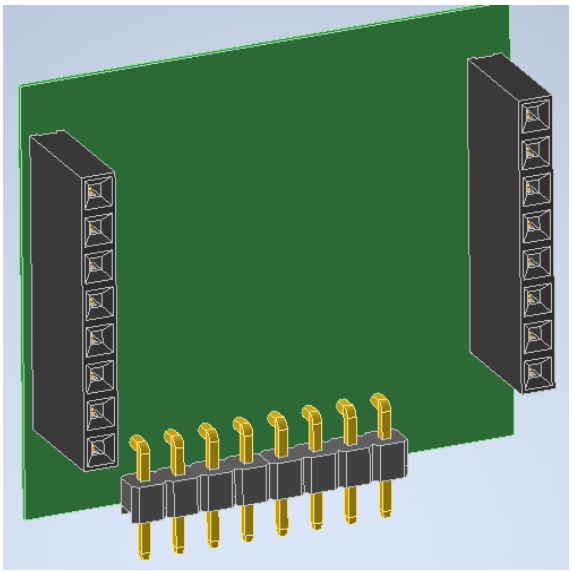


Figura 34: Diseño 3D final de la placa principal.



(a) Modelo 3D en Altium de la placa de la batería.



(b) Modelo 3D de la placa de la OpenMV Cam.

Figura 35: Diseños 3D finales de las placas de los submódulos.

Los modelos ya manufacturados y con sus componentes montados fueron:

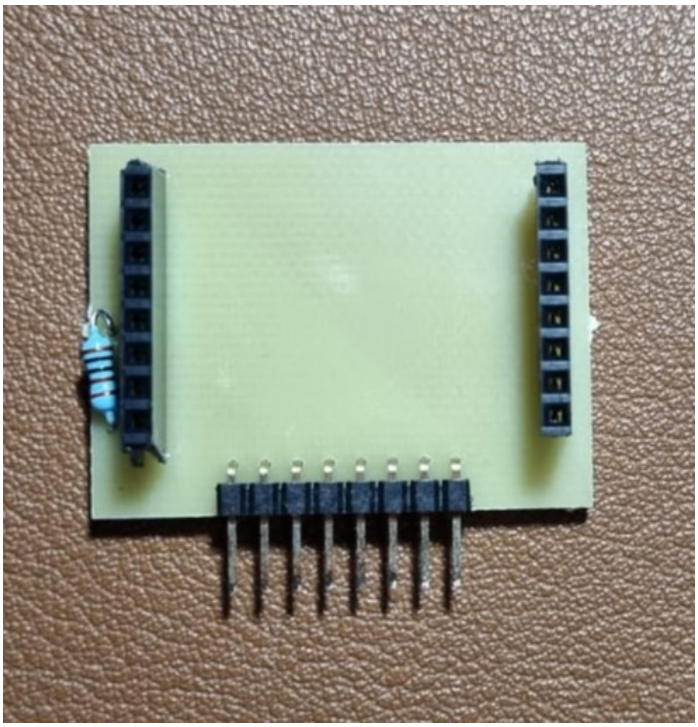


Figura 36: Placa de la cámara manufacturada y montada.

Figura 37: Placa de la celda manufacturada y montada

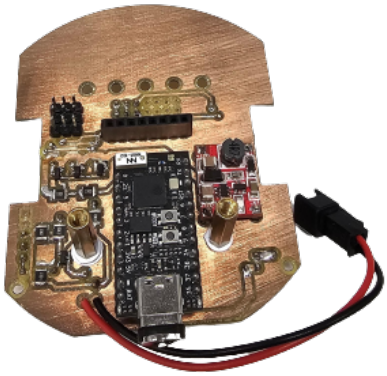


Figura 38: Placa principal manufacturada y montada.

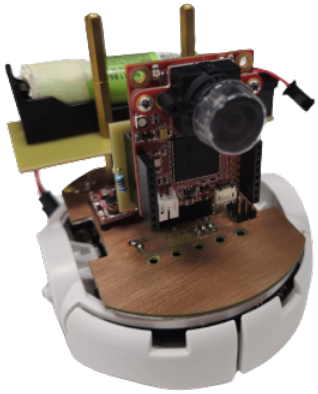


Figura 39: Agente con su placa principal y de cámara montadas.

En la Figura 39 se observa el agente con el módulo de cámara y batería ya montado. En la parte frontal de la placa principal se monta el manipulador serial al quitar la cámara.

CAPÍTULO 10

Diseño e implementación del manipulador serial

En este capítulo se presentan los pasos para configurar el módulo MCPWM en el TinyS3, que se utiliza para controlar la posición de los servomotores que componen el manipulador serial. También se incluye el proceso de diseño 3D del manipulador serial.

El control del manipulador serial recibe la señal de control y los parámetros a través de WiFi mediante el punto de acceso (AP) creado para el control de la OpenMV Cam H7. Estos valores pueden incluir la configuración en bruto de las articulaciones, así como las coordenadas a las cuales se busca mover el agente. Para el diseño del manipulador serial se buscó una estructura modular que permita realizar modificaciones en el futuro.

10.1. Metodología

Para el diseño del manipulador serial, se comenzó por configurar el módulo MCPWM. Una vez configurado, se procedió a desarrollar las funciones necesarias para controlar el manipulador serial, permitiendo tanto el envío de valores en bruto como la especificación de coordenadas. La implementación de esta última función se realizó en paralelo con el diseño del modelo 3D del manipulador serial.

10.1.1. Configuración del módulo MCPWM en el TinyS3

```
1 static void mcpwm_initialize(void);
```

Código 10.1: función para inicializar el MCPWM

Para iniciar la configuración del MCPWM, como se muestra en la Figura 10.1, se inició configurando los pines para que sean utilizados con el PWM utilizando la función **mcpwm_-**

gpio_init(...). En esta función se asocian los canales de salida de la unidad 0 de los módulos de PWM y se indica que el pin asociado se utilizará como salida PWM.

A continuación, se crearon tres estructuras para definir los parámetros iniciales de cada servomotor. Estos parámetros incluyen la frecuencia del PWM (establecida en 50 Hz), el ciclo de trabajo inicial y el modo del contador asociado, configurado como un PWM de alta velocidad de respuesta.

Con las estructuras creadas, se pasaron como argumentos a la función **mcpwm_init(...)**, que se encarga de inicializar los canales del PWM con los parámetros establecidos. Es importante destacar que estos parámetros son modificables después de la inicialización y conservan los cambios aplicados.

10.1.2. Actualización de los valores de configuración

```
1 static void updatePositionSCH(void *arg);
```

Código 10.2: función para actualizar la configuración de los servomotores.

Después de haber inicializado el MCPWM, como se muestra en la Figura 10.2, se asigna esta configuración a una tarea la cual inicia con un bucle infinito. Al comenzar este bucle, se establecen los límites superiores e inferiores para cada articulación y se verifica si los valores deseados están dentro de este rango. En caso de que no lo estén, se redondean a los valores límite correspondientes.

Después de asegurarse de que los valores deseados estén dentro del rango permitido, se almacena la posición actual en variables que servirán como referencia para el movimiento en un bucle de control. Al entrar en este bucle de control se compara si las posiciones deseadas son mayores o menores que las actuales. Esto se hace para aumentar o disminuir gradualmente el valor de la posición temporal almacenada y actualizar el ciclo de trabajo en función de este valor.

Para ajustar los valores, primero se calcula el ciclo de trabajo correspondiente a la nueva posición corregida en un paso. Luego, se establece este ciclo de trabajo para cada canal utilizando la función **mcpwm_set_duty_in_us(...)**. Después de actualizar el valor temporal en un paso, se almacena en la posición actual y se espera durante 15 milisegundos lo que permite suavizar el movimiento entre pasos. El paso configurado para este bucle de control es de 1° cada 15 ms.

Fuera del bucle de control, se presenta un retraso de 10 milisegundos para evitar problemas con la activación del *Watchdog timer*.

Esta función se utiliza para configurar los valores brutos de las juntas cuando se reciben por WiFi o se calculan a partir de coordenadas.

10.1.3. Configuración con base en coordenadas

```
1 static void CalcConfig(void *arg);
```

Código 10.3: función para calcular la configuración de las juntas con base en coordenadas.

La función mostrada en la Figura 10.3 se asigna a una tarea y se ejecuta al recibir el byte de control adecuado a través de WiFi. En caso de no recibir el byte de control correcto, se activa un retraso de 10 milisegundos para reiniciar el *Watchdog Timer*.

Cuando se ingresa en el bucle, se realiza el proceso de conversión de la coordenada en Y solicitada, que se recibe como un entero, a un valor de punto flotante. Para el cálculo de la configuración, se utiliza la ecuación 37:

$$\begin{aligned} Q_2 &= -\arccos\left(\frac{X^2 + Y^2 - L_1^2 - L_2^2}{2 * L_1 * L_2}\right) \\ Q_1 &= \arctan(Y/X) - \arctan\left(\frac{L_2 * \sin(Q_2)}{L_1 + L_2 * \cos(Q_2))}\right) \end{aligned} \tag{37}$$

Donde X y Y son las coordenadas solicitadas, L_1 y L_2 representan la longitud de los eslabones. Después de determinar la configuración se redondea el valor obtenido y se convierte a grados. Luego se verifica que los valores obtenidos no estén fuera de los límites establecidos. Si están dentro de los límites permitidos, se asigna el valor obtenido a la configuración deseada y se finaliza asignando la bandera que activa este bucle a 0.

10.1.4. Diseño del manipulador serial

El manipulador serial se diseñó utilizando Autodesk Inventor 2024, empleando mediciones directas del servomotor SG90 como referencias iniciales para luego validarse con un modelo 3D del servomotor al realizar un primer ensamble parcial.

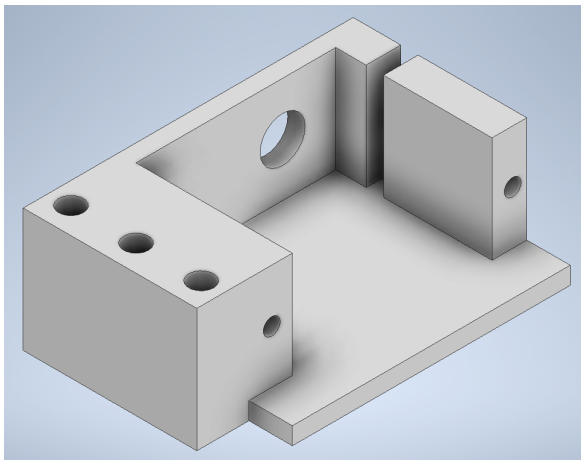
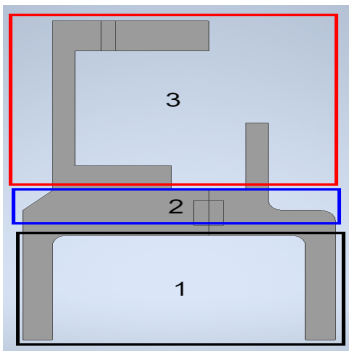
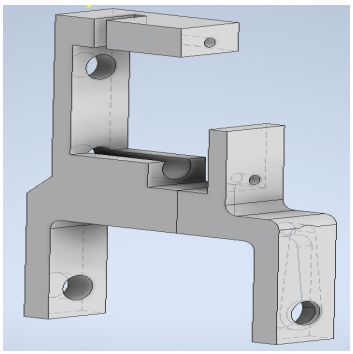


Figura 40: Modelo 3D de la base utilizada para el manipulador serial.

Se comenzó con el diseño de la base, como se muestra en la Figura 40. Esta base se diseñó con el menor tamaño posible y cuenta con un punto de anclaje que se acopla a los conectores físicos frontales de la placa de expansión del agente. El sistema de montaje utilizado en esta base implica el uso de pines que atraviesan completamente la placa de expansión y la base. Después del montaje, se coloca un anillo a presión en el segmento que sobresale del pin para evitar que se caiga y mantener la base en su posición.



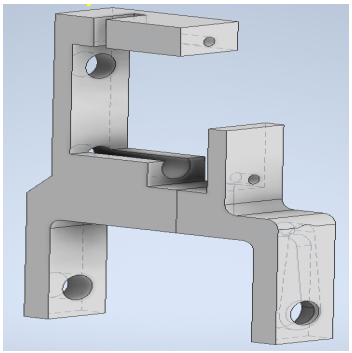
(a) Segmentos del eslabón.



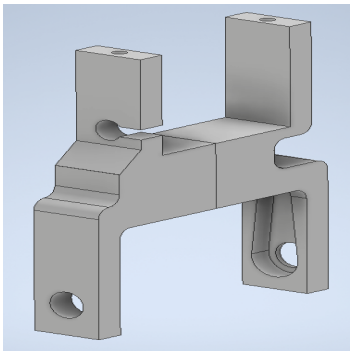
(b) Vista lateral del eslabón.

Figura 41: Modelo 3D del eslabón utilizado en el manipulador serial.

Luego, al diseñar el eslabón de la Figura 41, se consideraron tres elementos cruciales mostrados en la Figura 41a: el pivote (1), la sección media (2) y el conector (3). El pivote se utiliza para montar el actuador $n-1$ en el eslabón, y en él se incluye un espacio para el cuerno o brazo atornillable, como se muestra en la Figura 41b, lo que permite que su movimiento se transmita al eslabón completo. Luego está la sección media, que se utiliza para alargar o acortar el eslabón, modificando su longitud. Por último, se encuentra el conector donde se monta el actuador n y cumple la función de fijarlo directamente al eslabón, preparándolo para montar el siguiente eslabón. Este ultimo segmento puede ser modelado de distintas formas, como se observa en la Figura 42, dependiendo de la orientación deseada para el siguiente actuador.



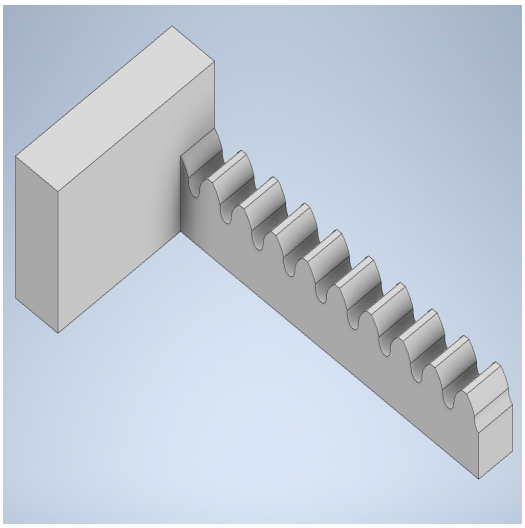
(a) Eslabón 1.



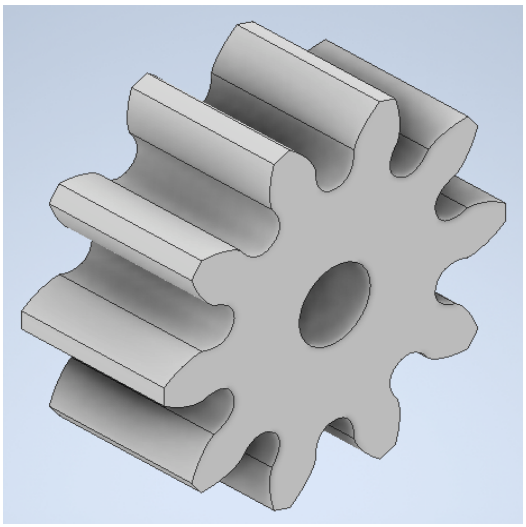
(b) Eslabón 2.

Figura 42: Eslabones utilizados en el manipulador serial.

Dado que el manipulador serial propuesto utiliza únicamente 3 actuadores, se diseñaron 2 tipos de eslabones y un efector final. Los eslabones se diseñaron como se ve en la Figura 42. El eslabón 1, observable en la Figura (42a), se diseñó para que permitiera montar el segundo actuador con su eje de rotación alineado al actuador inicial de la base y la misma orientación. El eslabón 2, observable en la Figura (42b), se diseñó para colocar el eje de rotación del segundo actuador perpendicular al actuador donde se monta, lo que permite manipular el efector final de la manera deseada.



(a) Cremallera.



(b) Piñón.

Figura 43: Piñón y cremallera utilizados en el efector final.

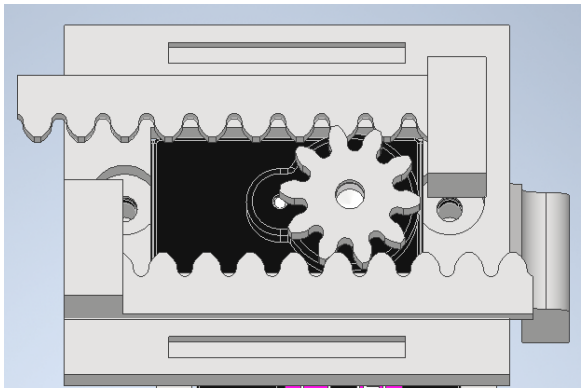


Figura 44: Montaje del piñón y la cremallera en el efector final.

Luego de contar con los eslabones diseñados, se continuó con el diseño del efector final. Para este se tomó como base un sistema de engranaje y doble cremallera, como se muestra en la Figura 43. El engranaje se montó en el segundo efector final y el movimiento rotatorio de este accionaría las cremalleras, las cuales se montarían a presión en el piñón y en los extremos del efector final, como se ilustra en el montaje del efector final en la Figura 44. Este montaje genera una garra simple con un mecanismo de movimiento compacto, ideal para el agente.

10.2. Resultados y discusión

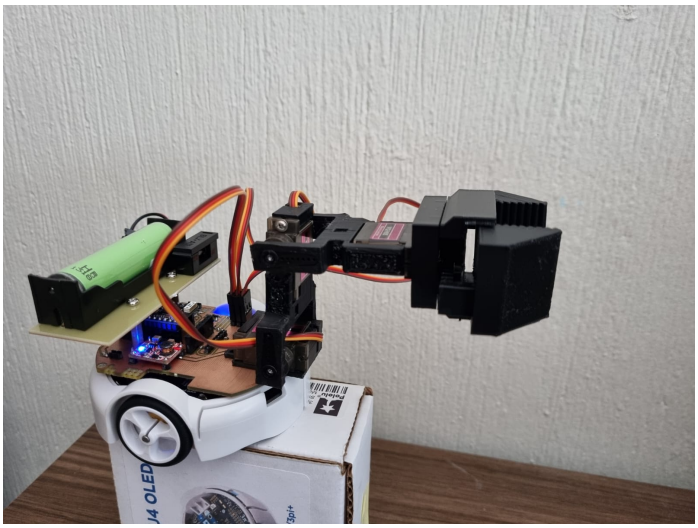


Figura 45: Manipulador serial montado en el agente y la placa de expansión.

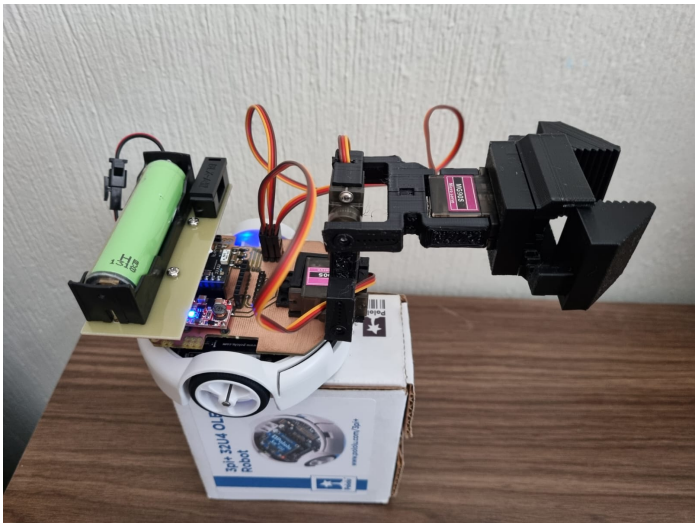
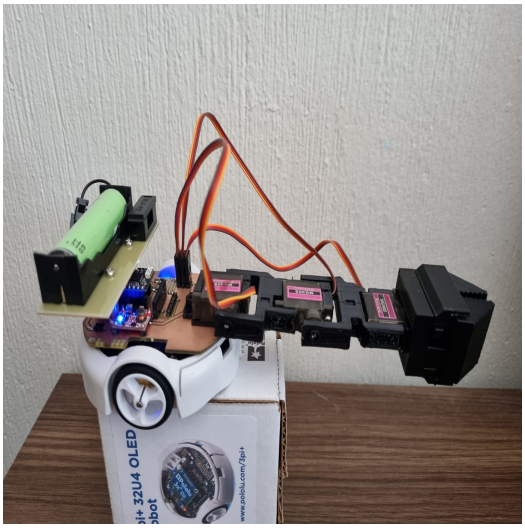
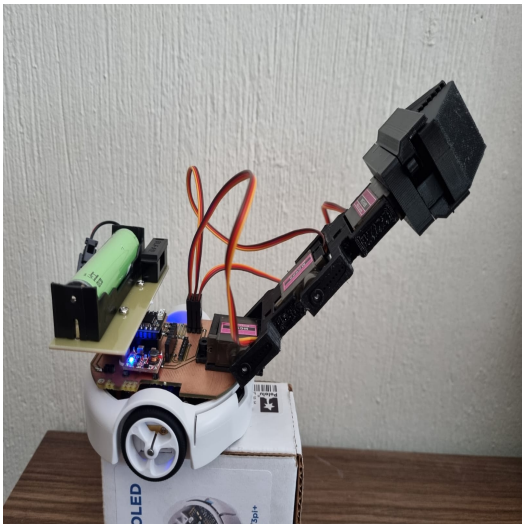


Figura 46: Garra del manipulador serial abierta.

En la Figura 45 se puede observar el agente con el manipulador serial montado. A lo largo del diseño, no se propuso un tamaño inicial basado en la observación del agente sino que su tamaño se determinó utilizando el ensamblaje en el modelo 3D y las primeras iteraciones del modelo 3D impreso. Las medidas finales para el eslabón 1 fueron de 47 mm y para el segundo eslabón fueron de 68 mm, con un piñón de 12 mm de diámetro. Las cremalleras cuentan con un largo efectivo de 35 mm. La apertura máxima del efector es de 30 mm con esta configuración.



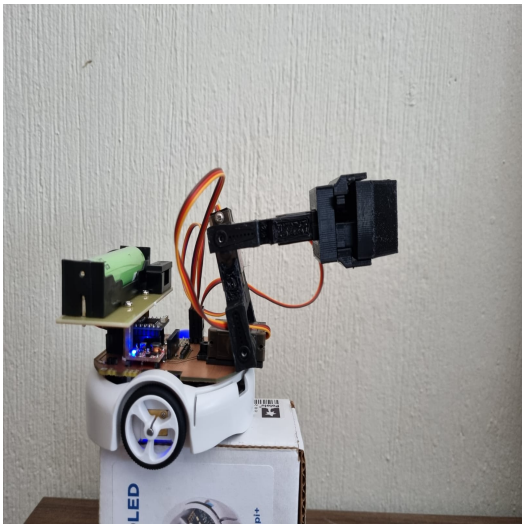
(a) Manipulador extendido a 0°.



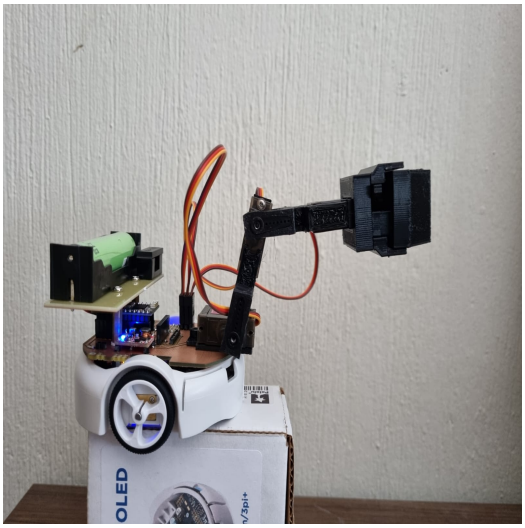
(b) Manipulador extendido a 45°.

Figura 47: Distintas configuraciones del manipulador serial.

En la Figura 47, se pueden observar dos de las múltiples configuraciones que puede presentar el manipulador serial. Estas configuraciones se alcanzaron al enviar las coordenadas en bruto por medio de WiFi. Luego de haber realizado diversas pruebas, se determinó que la mayor precisión del manipulador es de 0.2°. Por debajo de este valor, la mayoría de las veces, el servomotor no responde y es necesario aumentar en otro paso su posición para que responda correctamente.



(a) Coordenadas (50,50)



(b) Coordenadas (80,50)

Figura 48: Configuración del manipulador serial en base a coordenadas.

En la Figura 48 se puede observar el cambio en la configuración del agente al enviar un

grupo de coordenadas. Las coordenadas probadas en este ejemplo son movimientos a (50,50) y luego a (80,50). Se observa en las Figuras que se conserva la altura entre movimientos, pero la distancia desde la base aumenta. Al realizar las pruebas, se determinó que el mínimo movimiento permitido es de 3 mm. Por debajo de este valor se presenta el mismo problema en el cual los actuadores no se activan hasta que se logra el cambio mínimo requerido.

CAPÍTULO 11

Algoritmo de parqueo automático

En este capítulo, se presentan los pasos para definir la función que permite el estacionamiento automático del agente bajo demanda o cuando detecta un nivel bajo de carga.

El estacionamiento bajo demanda se utiliza al finalizar las pruebas con uno o múltiples agentes. Esta función se ejecuta al concluir las pruebas y dirige a los agentes a sus puestos de carga, donde recargarán sus celdas y estarán en espera para una nueva conexión. El estacionamiento al detectar un nivel bajo de carga, de igual manera, regresa a los agentes a sus puestos de carga. A diferencia del estacionamiento bajo demanda, este se activa automáticamente por el agente para evitar niveles muy bajos en su celda durante las pruebas.

11.1. Metodología

La función se desarrolló en Matlab, ya que esta plataforma cuenta con múltiples *Toolboxes* y funciones que facilitan el cálculo de trayectorias de manera eficiente.

Se comenzó declarando un grupo de constantes: el radio de las llantas, la distancia de las llantas desde el centro y el desfase para ajustar la orientación de los agentes. El radio de las llantas y la distancia desde el centro son utilizados por todos los agentes, mientras que el desfase presenta un valor distinto dependiendo del agente que solicite el retorno a su estación de carga.

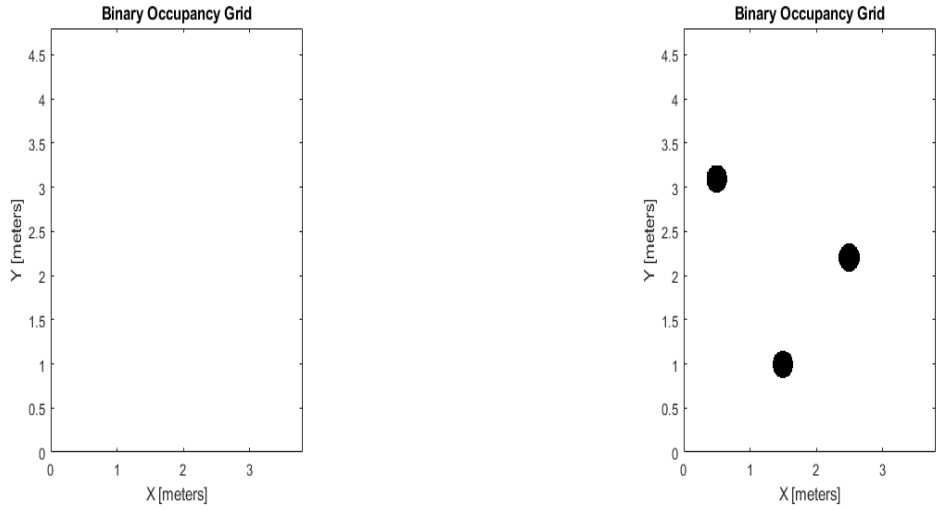
Luego de haber declarado las constantes, se obtuvieron las poses de cada uno de los marcadores en la plataforma del Robotat, a excepción del agente que activó el comando, si aplica. Estas poses se obtuvieron mediante la representación de ángulos de Euler. Una vez obtenidas las poses, se registraron las coordenadas XY en un arreglo y se conectó al agente en la red AP que crea.

11.1.1. Definición del espacio de trabajo y obstáculos en Matlab

Al contar con las coordenadas de los objetos, se creó un objeto de la clase "mapa binario" utilizando la función de Matlab llamada **binaryOccupancyMap(...)**. A esta función se le proporcionaron los valores de ancho, largo y resolución del mapa. El ancho es de 3.8 metros y el largo de 4.8 metros, con una resolución de 100 unidades, lo que resulta en un mapa con una precisión de 1 centímetro.

El mapa creado en la Figura 49a tiene su origen en la esquina inferior izquierda, mientras que el sistema de captura Optitrack tiene su origen en el centro de la plataforma. Por esta razón, se realizó un ajuste en las coordenadas. En el eje X, se sumaron 1.9 unidades a todas las coordenadas, y en el eje Y, se sumaron 2.4 unidades. Esto colocó los datos con referencia en la esquina inferior izquierda de la plataforma, representándolos como valores positivos.

Una vez realizado este ajuste, se procedió a agregar obstáculos al mapa, como se observa en 49b, utilizando la función **setOccupancy(...)**. A esta función se le proporcionó el mapa y las nuevas coordenadas ajustadas, junto con un vector unitario y lineal de tamaño N, donde N es el tamaño de los obstáculos a representar. Para garantizar una representación más precisa, se inflaron los obstáculos ya que originalmente se detectaban como elementos puntuales. El inflado se realizó mediante la función **inflate(...)**, a la cual se le proporcionó el mapa con los obstáculos ya agregados y un radio de inflado de 15 cm o 0.15 m.



(a) Mapa binario creado sin obstáculos.

(b) Mapa binario con obstáculos inflados.

Figura 49: Mapas binarios creados inicialmente.

11.1.2. Generación de trayectoria (RRT)

Después de crear el mapa con obstáculos, se construyó un modelo de espacio de estados de Dubin utilizando la función **stateSpaceDubins(...)** de la *Robotics Toolbox* de Matlab. A esta función se le proporcionaron los límites inferiores y superiores del plano XY (en metros), así como las coordenadas máximas de giro para el agente. Las coordenadas de giro

proporcionadas fueron $[-\pi, \pi]$, para asegurar que el rango de giro en el espacio de estados estuviera dentro de los valores esperados por el controlador (como se muestra en la Figura 9).

Después de definir los límites para el espacio de estados, se especificó el radio mínimo de giro aceptado, asignándole un valor de 10 cm.

Una vez que se había definido el espacio de estados, se creó un validador para las trayectorias calculadas utilizando la función **validatorOccupancyMap(...)**. Al validador se le proporcionó el espacio de estados a utilizar, el mapa binario con sus obstáculos y una distancia máxima de validación. El valor máximo de validación asignado fue de 0.1 m, lo cual es ligeramente mayor que el diámetro del agente para evitar colisiones.

Luego, se procedió a crear el planificador de trayectorias utilizando la función **plannerRRT(...)**. Esta función se basa en el algoritmo RRT (Rapidly-Exploring Random Trees). Al planificador se le proporcionó el espacio de estados de Dubin configurado y el validador de trayectorias.

Una vez que el planificador fue creado, se establecieron varios parámetros. Se definió la máxima distancia de conexión como 0.5 metros y el número máximo de iteraciones como 100,000. Además, se indicó la función utilizada para determinar si se alcanzó o no la meta. La función seleccionada es la presentada en el ejemplo de aplicación de [30], pero con un umbral reducido a 0.1 cm o 0.001 m para minimizar el error entre la posición final y la meta.

Con el planificador configurado, se obtiene la posición del agente que activó la función o del agente se retornara a su estación de carga. Además, se obtiene la posición de su estación de carga. La pose de la estación de carga se utiliza para calcular el ángulo con el cual el agente debe llegar al punto deseado.

Una vez definida la meta, se planificó la trayectoria utilizando la información proporcionada mediante la función **plan(...)**. En esta función se le proporcionó el planificador, la pose inicial y la pose final deseada. Como resultado, esta función retorna un arreglo de dimensiones $2 \times N$ con las coordenadas de los puntos de la trayectoria planificada.

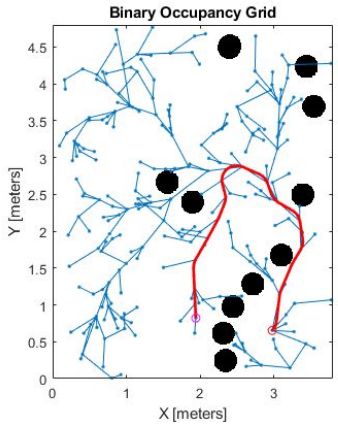


Figura 50: Mapa binario con obstáculos inflados y la trayectoria utilizada.

Después de obtener los puntos de la trayectoria, se retornan sus valores a coordenadas utilizables en la plataforma del Robotat, se resta 1.9 unidades al eje X y 2.4 unidades al eje Y. Esto se realizo para ajustar las coordenadas al sistema de referencia de la plataforma del Robotat.

11.1.3. Seguimiento de la trayectoria y parqueo en su estación.

Una vez ajustadas las coordenadas, se procede con el envío de la trayectoria al agente. Se declara una variable para determinar el siguiente punto al cual debe dirigirse el agente y se entra en un ciclo while. Este ciclo no se completa hasta que el agente haya llegado al último punto en la trayectoria.

Al inicio de este ciclo, se obtiene la pose del agente, ajustando su orientación en función del desfase previamente establecido. Luego, se selecciona el punto de la trayectoria basado en la variable previamente establecida. Se calculan los errores de orientación y posición con respecto a este punto. Si el agente no está lo suficientemente cerca del punto (por ejemplo, a menos de 1 cm), se mantiene el mismo punto objetivo. Si está lo suficientemente cerca, se aumenta en uno la variable para tomar el siguiente punto en la trayectoria.

Con los errores calculados, se utiliza un controlador PID con acercamiento exponencial para calcular la velocidad lineal "V" del agente (como se muestra en la ecuación 11). Además, se utiliza un controlador PID estándar (como se muestra en la ecuación 10) para calcular la velocidad rotacional "W". Posteriormente, se calculan las velocidades de rotación de cada rueda según las ecuaciones en 25, y estas velocidades se envían al agente a través de la conexión WiFi.

Al salir de este bucle que permite al agente llegar a su posición final, se le indica que detenga por completo su marcha y se obtiene su pose final.

Con la pose final obtenida, se inicia un bucle para orientar al agente de manera que quede de espaldas al punto final. Dentro de este bucle, se verifica periódicamente la orientación del agente hasta que se encuentre parcialmente alineado con el eje de la estación de carga, permitiendo una variación absoluta de hasta 2 grados. Una vez que el agente se encuentra dentro de este rango establecido, se detiene.

Luego, se ingresa a un último bucle en el cual el agente retrocede a baja velocidad. Durante este retroceso, se calcula el error entre la posición actual del agente y la posición de la estación de carga. Cuando el error es menor a 0.01 cm, se detiene por completo, concluyendo así la rutina de estacionamiento.

11.1.4. Optimización de las variables del sistema

Los valores utilizados en el controlador PID que determina el valor de w eran:

Parámetro	Valor
Kp	1
Kd	0.1
Ki	0

Cuadro 13: Valores iniciales para el PID que calcula el valor de w.

Mientras que los valores para el PID de acercamiento exponencial que determina el valor de v fueron:

Parámetro	Valor
V_o	20
α	0.9

Cuadro 14: Valores iniciales para el PID de acercamiento exponencial para el calculo de de v.

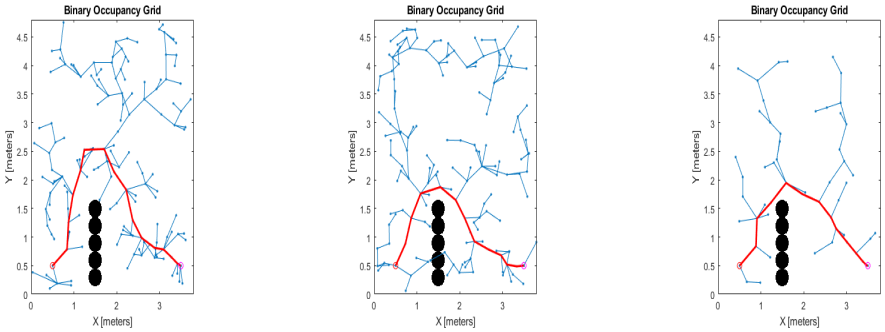
Estos valores iniciales se propusieron de manera conservadora para permitir la observación del comportamiento del agente en las primeras iteraciones, aunque su movimiento no sea eficiente. Dado que este algoritmo se utilizará para situaciones como el retorno ante baja carga, es importante que presente un movimiento fluido y controlado. Estos valores iniciales son empíricos y se propuso un enfoque de dos etapas para optimizarlos.

La primera etapa se centra en la optimización de los parámetros utilizados en los controladores PID, manteniendo constantes el radio de giro y el valor máximo de validación propuestos. En estas pruebas, el algoritmo se ejecutará varias veces, primero optimizando los parámetros del PID para la velocidad de rotación y, una vez que los giros sean más suaves, se continuará con la optimización de los parámetros del PID para la velocidad lineal. El objetivo de esta etapa es lograr un movimiento suave y sin cambios bruscos en el agente, además de alcanzar una velocidad adecuada a lo largo de la trayectoria.

En la segunda etapa, se modificaron los valores del radio mínimo de giro y la distancia de validación. Al ajustar estos parámetros, se busco encontrar el valor mínimo de radio de giro que evite que el agente diverja de la trayectoria y la distancia máxima de validación entre nodos, de manera que los cambios de velocidad del agente no sean bruscos.

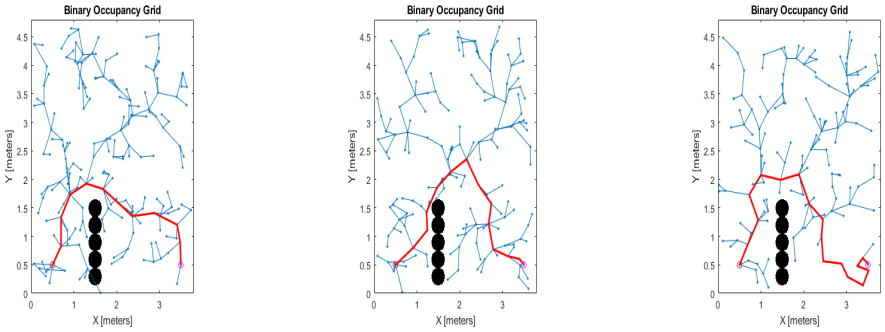
11.2. Resultados y discusión

En las primeras pruebas realizadas se colocaron marcadores a una distancia aproximada de 1.5 metros del agente a lo largo del eje X, espaciados cada 30 centímetros desde el borde, utilizando un total de 5 marcadores. Se consideraron dos orientaciones diferentes para el estacionamiento. La primera consistió en orientar al agente de manera que su eje X final fuera perpendicular al eje X del Robotat, y la segunda orientación fue con el mismo eje del agente perpendicular al eje Y del Robotat. A continuación, se presentan los resultados obtenidos colocando al agente en la esquina inferior izquierda y la meta en la esquina inferior derecha.



(a) (b) (c)

Figura 51: Primeras iteraciones del algoritmo de planificación, orientando al agente perpendicular al eje Y de la plataforma.

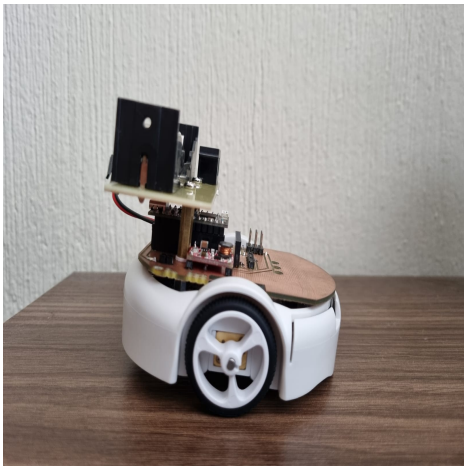


(a) (b) (c)

Figura 52: Primeras iteraciones del algoritmo de planificación, orientando al agente perpendicular al eje X de la plataforma.

Como se puede observar en las Figuras 51 y 52, el agente fue capaz de llegar a la meta evitando los obstáculos. Aunque en algunas iteraciones, como se muestra en la Figura 52b, pasa ligeramente por encima del obstáculo. Este comportamiento se consideró aceptable después de haber aumentado el tamaño de inflado del obstáculo más de lo necesario. Sin embargo, es importante destacar que, si bien el agente llega a la meta, no siempre se orienta completamente perpendicular al eje deseado como se puede apreciar en las Figuras 51 y 52. Esto se considera aceptable en este punto, dado que los parámetros del planificador aún no se han optimizado. Por lo tanto, se procede con las pruebas para ajustar y optimizar las variables de control del agente.

En las primeras pruebas, el agente experimentó cambios abruptos de velocidad, como se observa en la Figura 53a, los cuales causaban su inclinación. Estos cambios se producían al acercarse el agente al punto de destino y justo antes de cambiar al siguiente punto en la trayectoria. Cuando ocurría el cambio de punto, el agente aceleraba de golpe y luego volvía a su velocidad normal. Este comportamiento ejercía una carga adicional en los motores y, con el tiempo, podría llevar a un desgaste acelerado de las cajas reductoras. Por otro lado, la orientación del agente era bastante suave y presentaba un error mínimo. Por lo tanto, se ajustaron los parámetros de la orientación después de haber ajustado los parámetros de



(a) Agente inclinado debido al freno abrupto.

(b) Agente sin inclinación por movimiento constante.

Figura 53: Comportamiento del Pololu 3Pi+ antes distintos perfiles de movimiento.

velocidad lineal.

Después de realizar múltiples iteraciones, se determinaron los parámetros que permiten un movimiento suave, con una velocidad constante y evitan cambios bruscos al detenerse en el último punto. El valor final determinado para V_o fue de 80 y para α fue de 0.95.

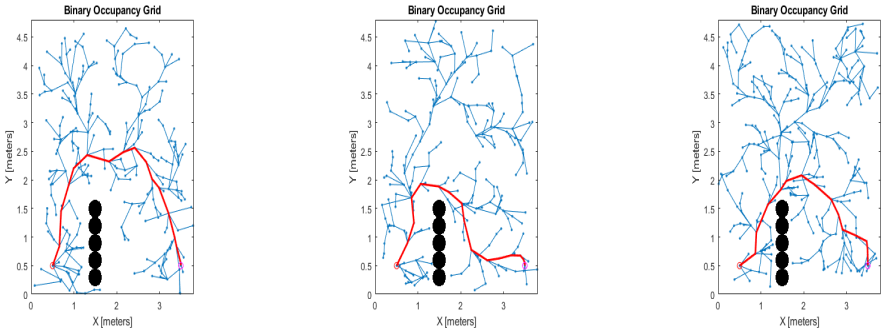
Después de optimizar los valores para el movimiento lineal, se realizaron ligeras modificaciones en los valores del controlador de orientación. Tras realizar 10 iteraciones, se encontró una combinación de parámetros que resultó en un error de orientación visualmente nulo y suavizó los cambios de orientación mayores a 45° entre puntos. Estos valores optimizados evitan que el agente se desvíe fácilmente de su trayectoria y logra una orientación final que se acerca lo más posible a la esperada. Los valores finales son para Kp de 10, para Kd de 0.0001 y Ki de 0.

Después de haber determinado los parámetros para lograr un movimiento fluido en el agente, se procedió a ajustar los parámetros del planificador y validador con el objetivo de obtener una trayectoria suave, sin movimientos bruscos y que permita que la orientación final sea lo más cercana posible a la deseada. Después de realizar múltiples iteraciones, se obtuvieron los siguientes resultados:

Estos valores se obtuvieron al cambiar el radio mínimo de giro a 20 cm y la distancia de validación a 5 cm. Al comparar estos resultados con los obtenidos en las primeras iteraciones, se observa que la trayectoria es más suave, presenta menos variaciones y la orientación final al llegar al punto deseado se asemeja más a la deseada.

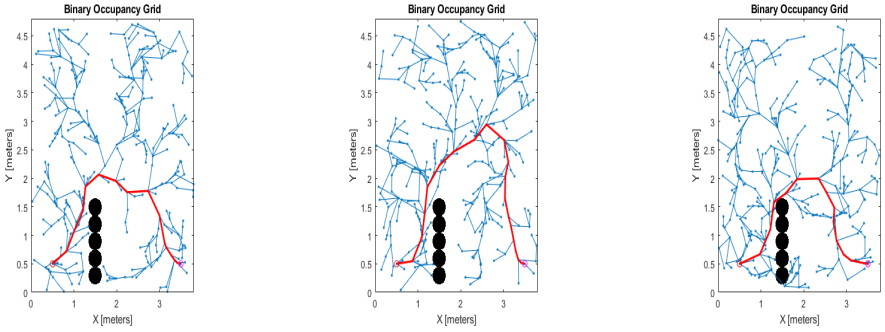
Por último, se muestra la forma en que el agente llega a un marcador utilizado como objetivo para las pruebas. La elección de utilizar un marcador se debe a la falta de una plataforma física donde montar el cargador y el circuito de carga del agente.

En el video **AgenteEstacionandose** del repositorio, se puede observar el algoritmo de estacionamiento automático desde el inicio hasta llegar al marcador objetivo, orientarse



(a) (b) (c)

Figura 54: Iteraciones finales para una orientación perpendicular respecto al eje X del Robotat.



(a) (b) (c)

Figura 55: Iteraciones finales para una orientación perpendicular respecto al eje Y del Robotat.

y estacionarse de regreso. En la última parte del video, se puede observar la rutina de estacionamiento del agente. Primero, el agente se acerca al marcador y lo encara desde el ángulo deseado. Luego, comienza a rotar hasta orientarse y dar la espalda al marcador, con una precisión de 4° respecto a la orientación inicial. Por último, inicia el retroceso hasta posicionarse sobre el marcador. Toda esta rutina simula el movimiento esperado para el estacionamiento del agente y su colocación en la estación de carga.

CAPÍTULO 12

Conclusiones

- La implementación de espacios de estados de Dubin en el algoritmo de RRT para la planificación de trayectorias permitió orientar al agente con una precisión de hasta $\pm 2°$ a lo largo de la rutina de parqueo, al mismo tiempo que se mantuvo la velocidad de generación que ofrece el algoritmo.
- La velocidad de transmisión de datos entre el TinyS3 y la OpenMV Cam H7 es aceptable, ya que en conjunto con el *socket* TCP permiten crear un *webstream* que alcanza hasta 8 FPS, con una imagen de baja resolución.
- El uso de un *socket* TCP para la comunicación entre el agente y un cliente, en conjunto con paquetes estandarizados de bytes para los mensajes, permite que múltiples dispositivos con capacidades WiFi puedan controlar y acceder a la información brindada por el agente.
- La implementación de diseño mecánicos de tipo modular mejora la flexibilidad y portabilidad del diseño, permitiendo el uso de múltiples métodos de manufactura y montura al incorporar distintas características a los elementos.
- El uso de una celda 18650 con una capacidad de 3000 mAh proporciona al agente una autonomía de hasta 2 horas sin disminuir la carga de la celda a niveles inferiores a los recomendados. Estos valores reflejan un estimado confiable, ya que tienen en cuenta los porcentajes de uso de cada módulo en el agente.

CAPÍTULO 13

Recomendaciones

- Se recomienda implementar las matrices de cinemática inversa para el posicionamiento por coordenadas, sustituyendo el mapeo por trigonometría. Esto permitirá eliminar algunas de las restricciones geométricas en el manipulador.
- Se recomienda explorar el envío de la imagen en formato JPG, permitiendo el envío de la imagen a color y una mayor resolución utilizando la menor cantidad de bytes durante la transmisión.
- Se recomienda el implementar la conexión de los agentes a la red del Robotat, lo que facilitará el uso de robótica de enjambre con los agentes.

CAPÍTULO 14

Bibliografía

[1] C. P. Montoya, "Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica," Tesis doct., 2021.

[2] S. Wilson, P. Glotfelter, L. Wang et al., "The Robotarium: Globally Impactful Opportunities, Challenges, and Lessons Learned in Remote-Access, Distributed Control of Multirobot Systems," *IEEE Control Systems Magazine*, vol. 40, n.º 1, págs. 26-44, 2020. DOI: 10.1109/MCS.2019.2949973.

[3] D. Pickem, P. Glotfelter, L. Wang et al., "The Robotarium: A remotely accessible swarm robotics research testbed," en, en *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, Singapore: IEEE, mayo de 2017, págs. 1699-1706, ISBN: 978-1-5090-4633-1. DOI: 10.1109/ICRA.2017.7989200. dirección: http://ieeexplore.ieee.org/document/7989200/ (visitado 19-04-2023).

[4] P. N. Beuchat, G. J. Bradford y G. Buskes, "Challenges and opportunities of using differential-drive robots with project-based learning pedagogies," *IFAC-PapersOnLine*, vol. 55, n.º 17, págs. 186-193, 2022, 13th IFAC Symposium on Advances in Control Education ACE 2022, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2022.09.277. dirección: https://www.sciencedirect.com/science/article/pii/S2405896322015130.

[5] P. Gonçalves, P. Torres, C. Alves et al., "The e-puck, a Robot Designed for Education in Engineering," *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, ene. de 2009.

[6] Q. Kong, L. Zhang y X. Xu, "Control algorithm of path planning and path tracking for automated valet parking," *Journal of Physics: Conference Series*, vol. 2232, n.º 1, pág. 012 018, mayo de 2022, ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/2232/1/012018. dirección: https://iopscience.iop.org/article/10.1088/1742-6596/2232/1/012018 (visitado 19-04-2023).

[7] H. Kwon, D. Cha, J. Seong, J. Lee y W. Chung, "Trajectory Planner CDT-RRT* for Car-Like Mobile Robots toward Narrow and Cluttered Environments," en, *Sensors*, vol. 21, n.º 14, pág. 4828, jul. de 2021, ISSN: 1424-8220. DOI: 10.3390/s21144828. dirección: https://www.mdpi.com/1424-8220/21/14/4828 (visitado 20-04-2023).

[8] OpenMV, *OpenMV cam H7*, 2018. dirección: https://openmv.io/collections/cams/products/openmv-cam-h7.

[9] MicroPython, *MicroPython - python for microcontrollers*, 2015. dirección: https://store.micropython.org/product/PYBLITEv1.0.

[10] pixycam, *Documentation*, mayo de 2022. dirección: https://docs.pixycam.com/wiki/doku.php?id=wiki%5C%3Av2%5C%3Astart.

[11] JeVois, *Welcome to the documentation of the Jevois Smart Embedded Machine Vision Toolkit*. 2015. dirección: http://jevois.org/doc/.

[12] K. Gonzalez y H. Alejandro, "Desarrollo e implementación de algoritmos de visión por computadora clásicos empleando OpenCV en sistemas embebidos.," Tesis doct., 2022.

[13] K. M. Lynch y F. C. Park, *Modern robotics: mechanics, planning, and control*, en. Cambridge, UK: Cambridge University Press, 2017, OCLC: ocn983881868, ISBN: 978-1-107-15630-2 978-1-316-60984-2.

[14] D. Freeman y K. Chen, *Integral linear quadratic regulator (LQR) control*, abr. de 2023. dirección: https://introcontrol.mit.edu/_static/spring23/lectures/lec11a-handout.pdf.

[15] T. M. Inc., *binaryOccupancyMap*, 2015. dirección: https://www.mathworks.com/help/nav/ref/binaryoccupancymap.html.

[16] S. M. LaValle, *Rapidly Exploring Random Trees: A New Tool for Path Planning*, 1998.

[17] I. Noreen, A. Khan y Z. Habib, oct. de 2016. dirección: https://www.uhb.edu.sa/Lists/Books%5C%20and%5C%20Articals/Attachments/1129/BookFile-%5C%20A%5C%20Comparison%5C%20of%5C%20RRT,%5C%20RRTstar%5C%20and%5C%20RRTstar-Smart%5C%20Path%5C%20Planning%5C%20Algorithms.pdf.

[18] M. E. Shacklett, A. Novotny y K. Gerwig, *TCP/IP: What is TCP/IP and how does it work?* Jul. de 2021. dirección: https://www.techtarget.com/searchnetworking/definition/TCP-IP.

[19] IBM, abr. de 2021. dirección: https://www.ibm.com/docs/es/aix/7.2?topic=protocol-tcpip-protocols.

[20] Pololu, *Pololu - 3PI+ 32U4 OLED robot - Standard edition (30:1 MP motors), assembled*. dirección: https://www.pololu.com/product/4975.

[21] TinyPICO, *Tinypico*. dirección: https://www.tinypico.com/.

[22] U. Maker, *Introducing the new TinyS3 from Unexpected Maker*, 2022. dirección: https://esp32s3.com/tinys3.html.

[23] Dirección: https://laelectronica.com.gt/motor-servo-mg90s-con-engranajes-de-metal?search=mg90&description=true.

[24] Dirección: https://laelectronica.com.gt/motor-servo-mg90s-con-engranajes-de-metal?search=mg90&description=true.

[25] Dirección: https://laelectronica.com.gt/modulo-protector-de-baterias-bms-1s-3a.

[26] Dirección: https://laelectronica.com.gt/modulo-elevador-de-voltaje-de-5v---1a?search=step+up&description=true.

[27] 2021. dirección: https://www.pololu.com/file/0J1825/3pi-plus-32u4-oled-control-board-dimensions.pdf.

[28] Dirección: https://www.pcbway.com/pcb_prototype/trace-width-calculator.html.

[29] W. Storr, *MOSFET as a switch - using power MOSFET switching*, ago. de 2022. dirección: https://www.electronics-tutorials.ws/transistor/tran_7.html.

[30] Dirección: https://www.mathworks.com/help/nav/ref/plannerrrt.html?s_tid=doc_ta.

CAPÍTULO 15

Anexos

15.1. Listado de componentes por placa.

Descripción	Identificador	Cantidad
Mosfet AO3402.	M1,M2,M3,M4	4
Step up 5V - 1A.	U1	1
Header macho, 1-Pin	U1	4
Header macho, 3-Pin	P2,P3,P4,P6	6
Header macho, 5-Pin	P6	1
Header macho, 2-Pin	P10	2
Header hembra, 8-Pin	P8	1
Resistor SMD 1210 1K Ohm	R2,R5,R6,R7,R8	5
Resistor SMD 1210 330 Ohm	R1,R4	2
Header hembra 11-Pin	UC1	1
Header hembra 12-Pin	UC1	1

Cuadro 15: Listado de componentes utilizados en la placa principal.

Descripción	Identificador	Cantidad
Porta fusible	F1	1
Porta batería 1S para 18650	H1	1
Header, 2-Pin	P7	1

Cuadro 16: Listado de componentes utilizado en la placa secundaria de batería.

Descripcion	Identificador	Cantidad
Header, 8-Pin Hembra	P1	2
Header, 8-Pin Macho a 90°	P1	1
Resistor SMD 1210 1K Ohm	R3	1

Cuadro 17: Listado de componentes utilizados en la placa secundaria de la OpenMV Cam H7

15.2. Enlace al repositorio del proyecto.

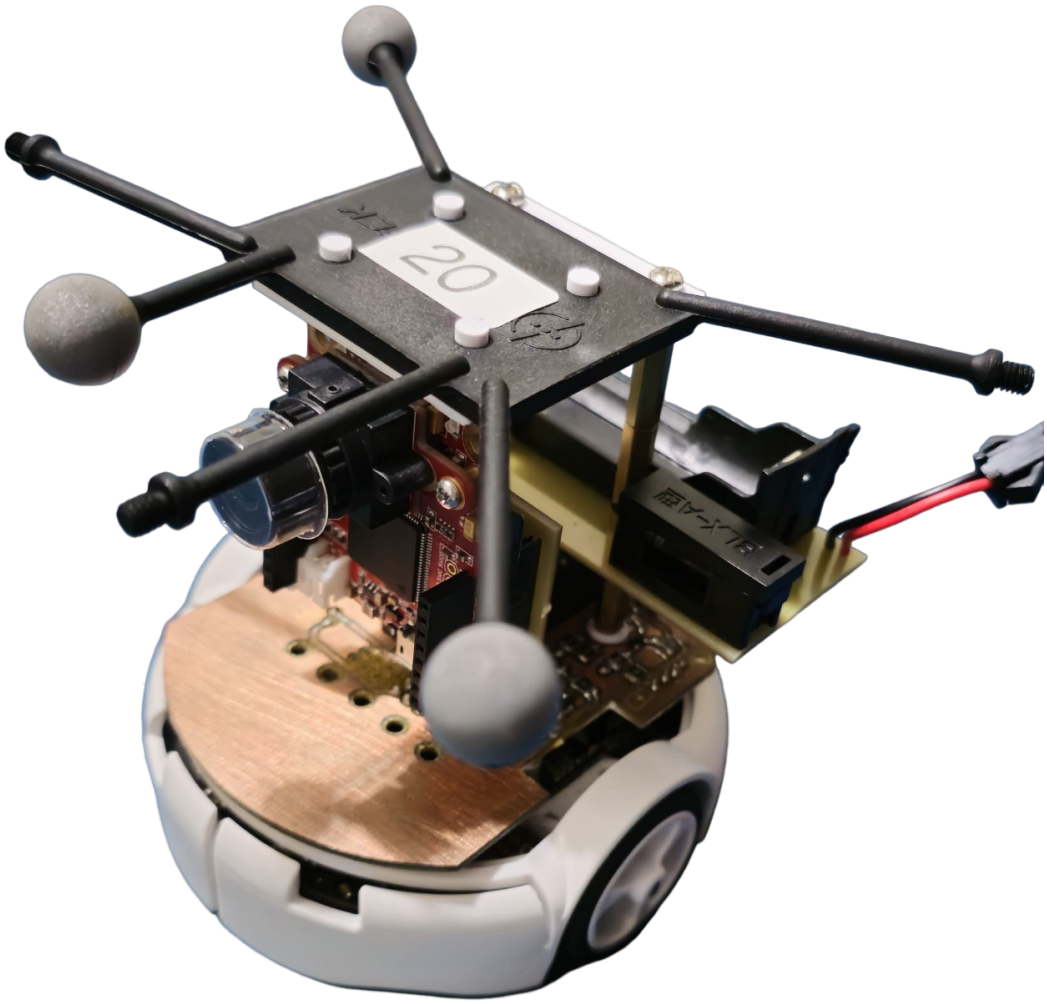
En el siguiente enlace se encuentra el repositorio del **proyecto**.La descripción y estructura del repositorio, así como las explicaciones de las porciones de código, se encuentran desglosadas en los archivos README por carpeta. Estos proporcionan explicaciones breves de las distintas partes del código, ejemplos de aplicación, descripciones de funciones y detalles sobre cómo preparar el entorno para realizar modificaciones.

15.3. Manual de usuario para el proyecto

A continuación, se incluye el manual de usuario desarrollado para poder ensamblar las placas de manera eficiente, permitiendo pruebas a lo largo de su ensamblaje. Además, se indica cómo ensamblar el manipulador serial y se proporcionan consideraciones para utilizar las placas, así como para su ensamblaje.

Manual de usuario para las placas de expansión del agente Pololu 3PI+.

Creado por:
José Luis Alvarez Pineda



Índice

1. Introducción.

En este manual se presenta una manera de ensamblar los módulos diseñados para el agente 3Pi+. Se explica cómo funcionan estos módulos y se detallan las pruebas que se pueden realizar para verificar su correcto funcionamiento durante el ensamblaje, asegurando así un despliegue sin problemas en la plataforma.

El proceso comienza con una breve introducción al funcionamiento de las placas y módulos creados, indicando el propósito de cada uno y la distribución de pines en la placa principal para su uso futuro. Luego, se describe el proceso de ensamblaje de las placas secundarias del agente, como las de la cámara y la batería, con el objetivo de realizar pruebas al ensamblar la placa principal. Después de ensamblar las placas secundarias, se detalla el proceso para ensamblar la placa principal, junto con un conjunto de pruebas para verificar su correcto funcionamiento y garantizar un despliegue seguro en la plataforma.

Una vez completado el ensamblaje de las placas, se proporcionan ejemplos de cómo montarlas en el agente y se ofrecen indicaciones para ensamblar el manipulador serial, seguido de su montaje. Finalmente, se describen las pruebas mencionadas y se incluye una sección para observaciones, donde se pueden anotar los cambios realizados en futuras iteraciones.

2. Módulos de expansión

2.1. Placa principal.

Esta placa desempeña la función de comunicar el TinyS3 con la OpenMV Cam H7, el agente Pololu 3Pi+ y el manipulador serial. En el agente, actúa como una capa intermedia de comunicación al enlazar el microprocesador A32U4 del agente con el cliente a través de una capa inalámbrica vía WiFi.

La placa cuenta con lo siguiente:

1. Bus de comunicación UART entre el TinyS3 y el agente.
2. Bus de comunicación SPI con pin de handshake para la comunicación entre el TinyS3 y el agente.
3. Módulo elevador de voltaje a 5V, para alimentar al manipulador serial.
4. Pines de control y alimentación para los servomotores del Manipulador serial.
5. Pin de control para encender y apagar al agente, en conjunto con la cámara y el Manipulador serial.
6. Pines para alimentar al agente directo con la celda externa conectada.
7. Pines para conectar la celda externa a la placa.

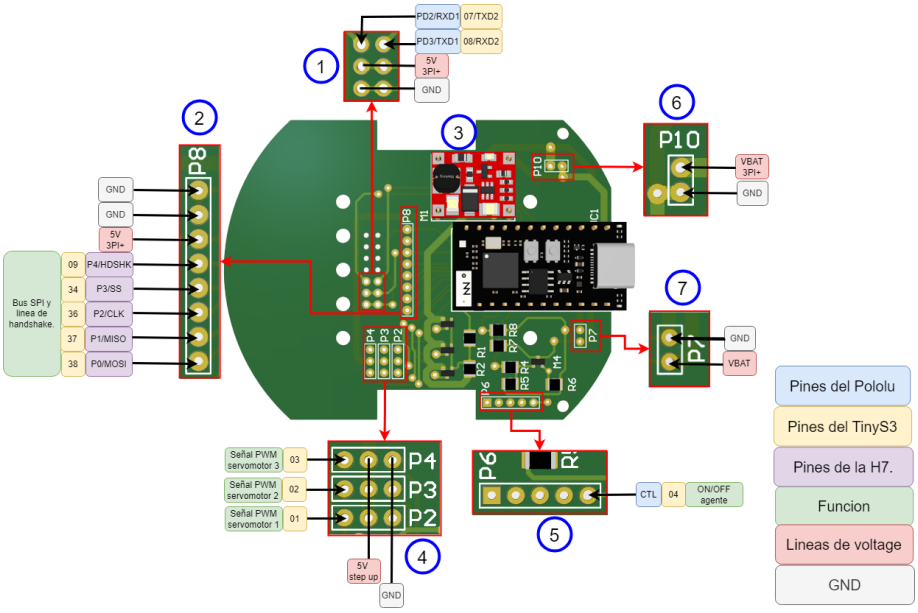


Figura 1: Identificación de los módulos y pines utilizados en la placa principal.

En la Figura 1 se puede observar la ubicación de cada uno de los componentes previamente mencionados, así como los pines utilizados por el TinyS3, la OpenMV Cam H7 y el agente 3Pi+. En la placa también se encuentran circuitos basados en MOSFETs como interruptores digitales, que permiten encender o apagar el agente y los servomotores, asi como un divisor de voltaje para leer el nivel de la celda.

Además de encargarse de comunicar los distintos componentes que integrarán al agente, esta placa también sirve como punto de montura para otros elementos, como el manipulador serial y el módulo de batería.

2.2. Manipulador Serial.

Este módulo físico está compuesto por 3 servomotores y una armazón impresa en 3D utilizando tecnología FDM. El material recomendado para la armazón es el PLA, el ABS o el PETG, sin embargo, se sugiere el uso de PLA debido a su facilidad de impresión y su bajo costo. En cuanto a los servomotores, este modelo utiliza los MG90 debido a su alto torque y engranes metálicos. No obstante, la junta del efector final puede ser reemplazada por un SG90, si bien este último cuenta con engranajes de plástico y un torque más bajo, sus especificaciones son suficientes para el propósito previsto.

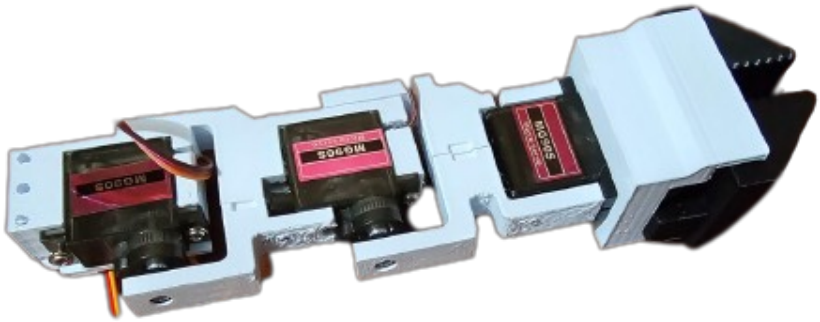


Figura 2: Manipulador serial ensamblado.

El módulo puede ser montado o desmontado del agente utilizando tornillos M3 con una longitud mínima de 15 mm. Se recomienda utilizar tornillos M3x20.

Por otro lado, se trata de un manipulador serial de 3 grados de libertad, incluyendo su efector final, que es controlado y alimentado desde la placa principal. Puede ser operado ya sea mediante coordenadas en 2 dimensiones o mediante las configuraciones en bruto de cada una de las juntas.

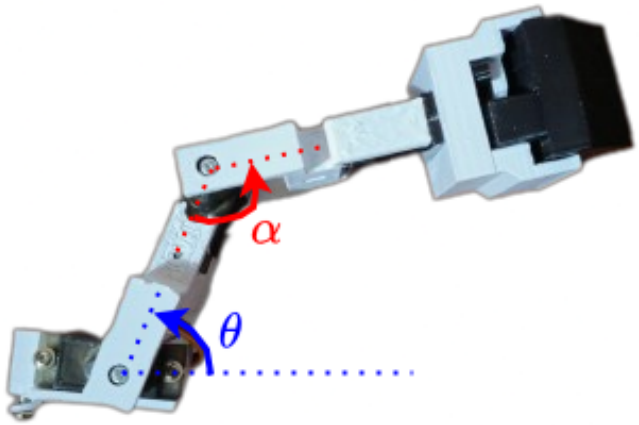
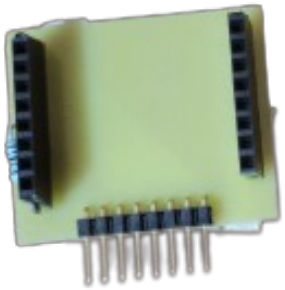


Figura 3: Ángulos de movimiento para el manipulador.

Como se puede observar en la Figura 3, para el servomotor de la base, el ángulo (θ) de configuración tiene un valor de 0° con respecto a la horizontal y aumenta en dirección opuesta a las manecillas del reloj. El segundo servomotor tiene un ángulo (α) de 0° con respecto al primer eslabón y aumenta de la misma manera en dirección opuesta a las manecillas del reloj. Ambos servomotores están limitados por software para evitar colisiones entre sí mismos, con el propio agente o elementos montados. El último servomotor, el del efector final, puede tomar valores intermedios entre 0° (totalmente abierto) y 180° (totalmente cerrado) y no se encuentra limitado ya que no presenta riesgo de colisión en su operación.

2.3. Placa para la OpenMV Cam H7.

Este módulo consiste en una placa que permite montar la OpenMV Cam H7 de forma vertical con respecto al agente, conectándola al puerto de comunicación y alimentación indicado en la placa principal. En esta placa se incluyen los headers hembra para montar la OpenMV Cam H7, una resistencia de 1KOhm para el handshake y headers macho a 90° para su montaje en la placa principal.



(a) Placa de montura.

(b) Placa con la OpenMV montada.

Figura 4: Módulo para montar la OpenMV Cam H7 en la placa principal.

Al montar la OpenMV Cam H7, se puede optar por usarla tanto con tarjeta SD como sin ella, según el propósito de su uso. Si se utiliza el script base empleado para las pruebas del agente, no es necesaria la tarjeta SD.

2.4. Placas para la celda y protección.

Esta placa únicamente se utiliza para montar la celda 18650 utilizada para alimentar al agente, placa principal y manipulador serial. Dado que la celda se le dará un uso no demandante, en esta placa se implementa un fusible de 3A que en conjunto con la lectura analógica de la celda por el TinyS3 sirven como un BMS, al evitar que la batería se descargue por completo y si se excede el consumo de corriente por algún mal funcionamiento el fusible protege tanto la celda como los módulos.

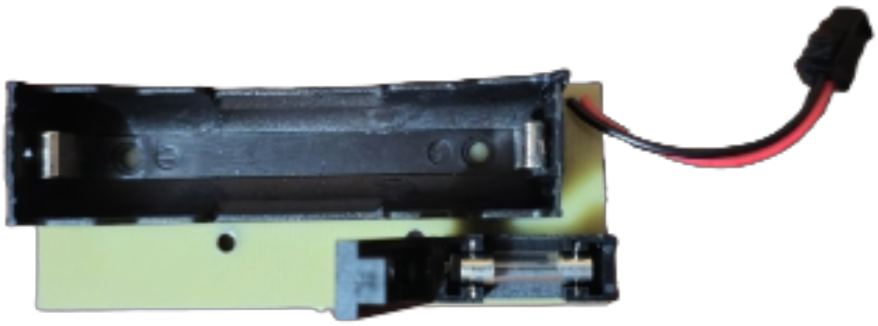


Figura 5: Placa con fusible y porta celda 18650.

3. Ensamble de la placa de batería y cámara.

3.1. Herramientas requeridas.

Entre los elementos requeridos para ensamblar las placas estan:

- Cautin, de preferencia de punta fina.
- Estaño.
- Corta alambre y pinzas.

3.2. Componentes requeridos.

Descripción	Identificador	Cantidad
Porta fusible	F1	1
Porta batería 1S para 18650	H1	1
Header, 2-Pin	P7	1

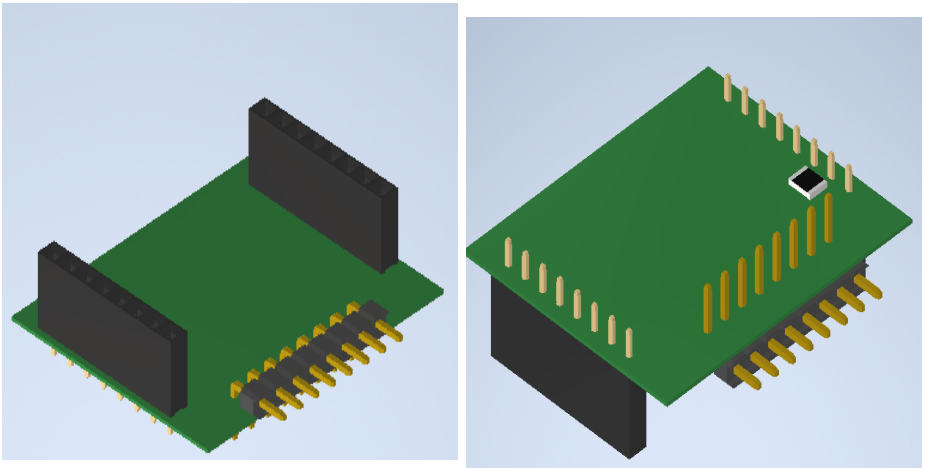
Cuadro 1: Listado de componentes utilizado en la placa de batería.

Descripción	Identificador	Cantidad
Header, 8-Pin Hembra	P1	2
Header, 8-Pin Macho a 90°	P1	1
Resistor SMD 1210 1K Ohm	R3	1

Cuadro 2: Listado de componentes utilizados en la placa de la OpenMV Cam H7

Dado que estas placas son de un lado no requieren de soldar vías en caso de ser manufacturadas en el MakerLab de la UVG.

3.3. Cómo montar los componentes en la placa para la OpenMV Cam H7.



(a) Vista superior de la placa para la OpenMV.

(b) Vista inferior de la placa para la OpenMV.

Figura 6: Vistas de la placa para la OpenMV Cam H7.

Ya que esta placa es sencilla, en la Figura 6 se observan las posiciones de los componentes. En la vista superior se pueden observar 3 filas de headers, donde en los laterales se deben colocar las hembras de 8 pines y en el centro la fila de headers macho a 90°. Observando la parte inferior de la placa, se encuentra un espacio para colocar una resistencia SMD de 1 KOhm; en esta zona solo debe soldarse la resistencia.

Al contar con todos los componentes montados, esta placa estará lista para montar la OpenMV Cam H7 y realizar las pruebas después de ensamblar la placa principal.

3.4. Cómo montar los componentes en la placa para la batería.

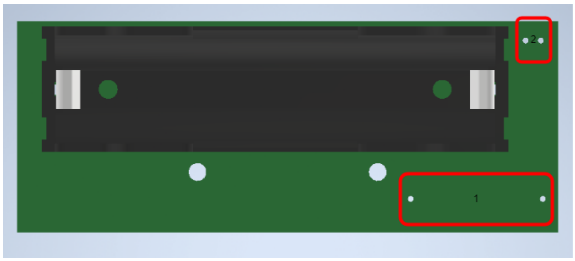


Figura 7: Componentes a soldar en la placa de batería.

Dado que esta placa es sencilla, se indican los componentes que se colocan en los espacios presentados en la Figura 7. En el indicador 1 se coloca el porta fusible, y en este caso se dejarán algunos en la Universidad, pero se puede modificar la placa y colocar cualquier porta fusible que sea necesario. En el indicador 2 se coloca el cable de conexión seleccionado para la placa principal, que puede ser similar al mostrado en 8.

4. Ensamble de la placa principal

4.1. Herramientas requeridas.

Entre los elementos requeridos para ensamblar la placa estan:

- Cautin, de preferencia de punta fina.
- Estaño.
- Corta alambre y pinzas.

4.2. Componentes requeridos.

Descripción	**Identificador**	**Cantidad**
Mosfet AO3402.	M1,M2,M3,M4	4
Step up 5V - 1A.	U1	1
Header macho, 1-Pin	U1	4
Header macho, 3-Pin	P2,P3,P4,P6	6
Header macho, 5-Pin	P6	1
Header macho, 2-Pin	P10	2
Header hembra, 8-Pin	P8	1
Resistor SMD 1210 1K Ohm	R2,R5,R6,R7,R8	5
Resistor SMD 1210 330 Ohm	R1,R4	2
Header hembra 11-Pin	UC1	1
Header hembra 12-Pin	UC1	1
Pareja de cable con conector	P7	1

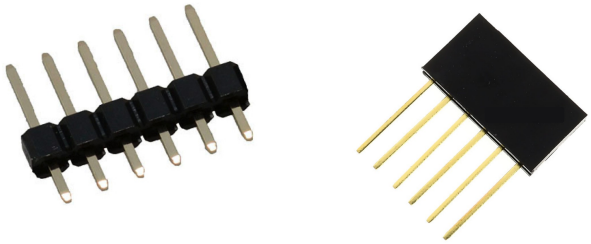
Cuadro 3: Listado de componentes utilizados en la placa principal.

Aparte de los componentes listados en el Cuadro 3, al manufacturar la placa en el MakerLab de la UVG se requiere de 1 metro de cable de protoboard para realizar los puentes necesarios y tanto el TinyS3 como el agente Pololu 3PI+ deben de tener el firmware cargado.

A continuación se presentan imágenes de los componentes requeridos para la placa principal:

Figura 8: Cables con conector para la batería.

Los cables que se utilicen deben tener un calibre de 20-22 AWG y contar con un conector que evite una conexión invertida de manera sencilla. Por lo tanto, se puede utilizar otro tipo de cable disponible al momento de ensamblarlos.



(a) Header macho. (b) Header hembra.

Figura 9: Headers a utilizar en la placa principal

Los headers a utilizar deben de contar con un espaciado de 2.54 mm, para el header hembra se recomienda una altura de conector de 9 mm.

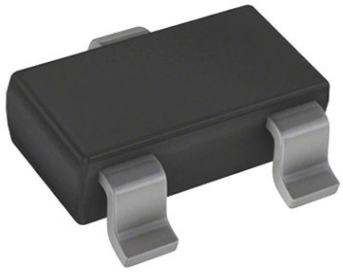


Figura 10: MOSFET AO3402.

En el caso de no encontrar el modelo específico de MOSFET disponible al momento de ensamblar otra placa, se puede utilizar un modelo equivalente que posea las mismas características y empaquetado.

4.3. Orden para montar los componentes

El orden presentado en esta sección es meramente una recomendación, la placa puede ser ensamblada en cualquier orden. No obstante, al seguir este orden, se logró probar cada uno de los componentes y elementos soldados por etapas, lo que facilitó el proceso de análisis y revisión en caso de fallos.

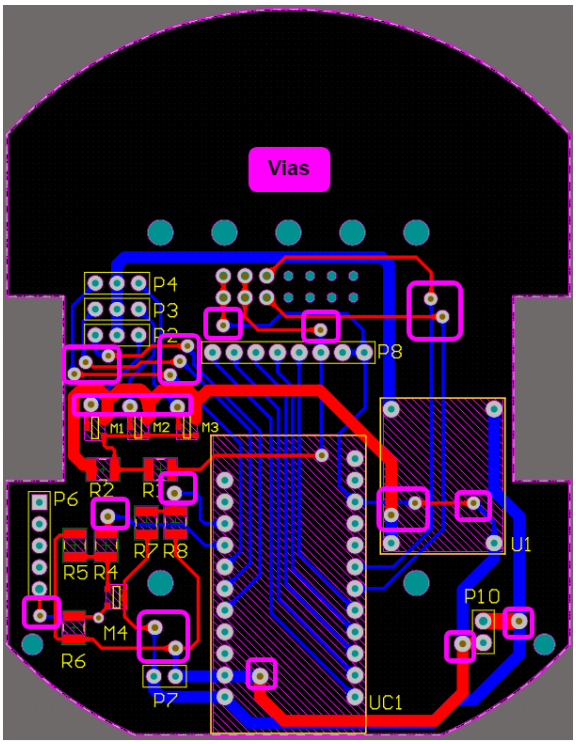


Figura 11: Vías presentes en la placa.

En caso de contar con placas manufacturadas en el MakerLab de la UVG, se recomienda comenzar soldando las vías entre capas y verificar su continuidad, las cuales se muestran en la Figura 11. Si las placas no son manufacturadas en el MakerLab y son producidas de manera industrial, este paso se puede omitir.

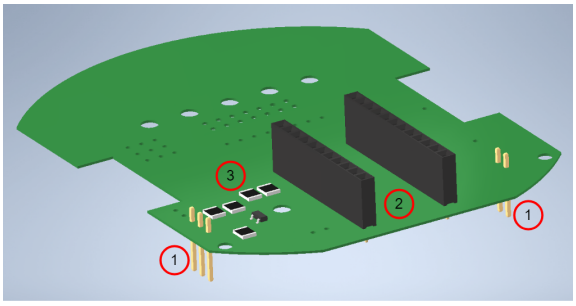


Figura 12: Orden para colocar los componentes de alimentación, control y montar el TinyS3.

Con la placa conectada en ambas capas, se procede con el ensamblaje de los componentes para alimentar la placa, el agente y los circuitos para encender y apagar el agente, así como el divisor de voltaje para leer el nivel de la batería.

En primer lugar, como se muestra en la Figura 12, se deben soldar los conectores P10 y P6 con headers macho orientados hacia la capa inferior, ya que estos se conectarán con el agente, y en P7 se conectará el

cable para la batería. Luego, se deben colocar los headers hembra de 11 y 12 pines en UC1 para conectar el TinyS3 al agente. Por último, se deben soldar la resistencia R4 de 330 Ohm y las resistencias R5, R6, R7 y R8 de 1KOhm, junto con el MOSFET M4.

Con esta etapa finalizada, se pueden realizar las pruebas de la **Primera etapa** (véase 8.1) para verificar el funcionamiento correcto de la placa y poder avanzar a la siguiente etapa.

Luego de haber realizado las pruebas mencionadas anteriormente y confirmado que el agente funciona de manera adecuada, se puede proceder con la siguiente etapa de ensamblaje. En esta etapa se ensamblarán los componentes necesarios para comunicarse con la OpenMV Cam H7 utilizando el bus SPI y con el agente mediante el bus UART.

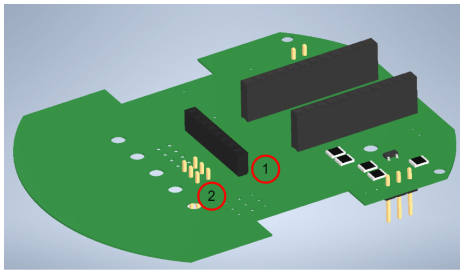


Figura 13: Colocando los componentes para comunicarse con la OpenMV y el agente.

Como se observa en la Figura 13, se coloca un header hembra de 8 pines en P8, orientado hacia arriba. Es crucial que este quede bien soldado y recto, ya que en él se montará la cámara. Luego, se deben soldar 2 tiras de header macho de 3 pines orientados hacia la parte inferior de la placa como en 2 en la Figura 13, ya que estos se conectarán al agente.

Al terminar de soldar estos elementos se pueden realizar las pruebas de **Segunda etapa** (vease 8.2), en estas se verifica el correcto funcionamiento de los buses de comunicación y alimentación a la Camara OpenMV.

Luego de haber realizado las pruebas y corroborado que la etapa anterior funciona de manera adecuada se procede a soldar los componentes que controlan y alimentan el manipulador serial.

En esta última etapa se soldaran los MOSFETS M1 al M3, las resistencias R2 y R3, los headers de 3 pines P2 a P4 y el módulo step up a 5V en U1 utilizando los headers de macho de 1 pin.

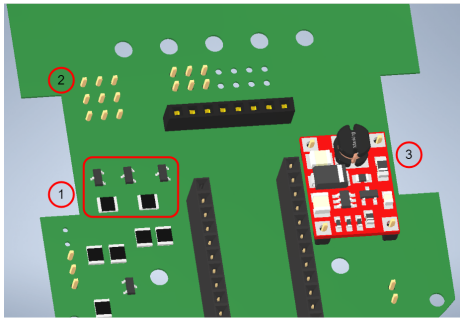


Figura 14: Colocando los componentes para el control y alimentación del manipulador serial.

En primer lugar, se colocan los MOSFETs en la posición 1 y las resistencias R1 de 330 Ohms y R2 de 1 KOhm, que forman parte del circuito para encender o apagar los servomotores. Luego, se continúa soldando los headers macho de 3 pines en la posición 2, los cuales alimentan los servomotores y transmiten la señal de control. Por último, se coloca el módulo step-up. Dado que este utiliza headers macho de 1 pin en sus 4 puntos de conexión, se recomienda colocar los pines con el extremo más corto hacia la placa. Posteriormente, se monta el step-up y se sueldan los pines, dejando para último lugar la soldadura en la placa. Este orden evita someter a esfuerzos innecesarios la soldadura al montarlos, considerando la tolerancia significativa de estos módulos.

Al terminar de montar estos componentes se pueden realizar las pruebas de **Tercera etapa** (véase 8.3), pero, para ello se requiere de ensamblar el manipulador serial y montarlo, por lo cual en capítulos posteriores se volver a mencionar esta prueba.

5. Montando las placas en el agente

5.1. Materiales a utilizar.

Entre los materiales a utilizar para ensamblar las placas entre si están:

Descripción	Cantidad
Placa principal.	1
Placa de batería.	1
Placa de la cámara.	1
Tira de 40 headers hembra	1
Espaciadores de cobre hembra - hembra	2
Espaciadores de cobre hembra - macho	4

Cuadro 4: Listado de materiales a utilizar para ensamblar el manipulador serial.

También se debe de contar con cautin, estaño y pinzas, en caso el agente no tenga los pines minimos soldados.

5.2. Cómo montar las placas.

Previo a montar la placa en el agente, primero se debe de asegurar que el agente cuente con los siguientes pines soldados:

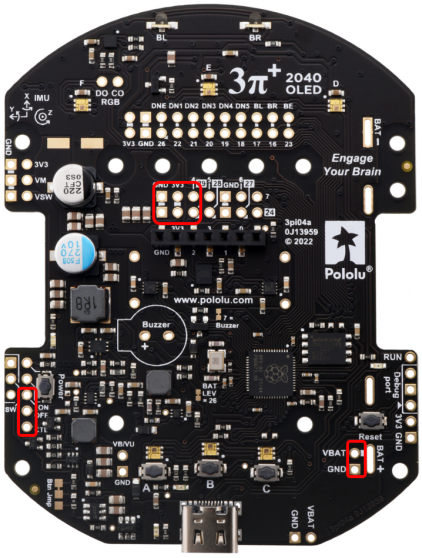


Figura 15: Pines que debe de tener soldado el agente.

Encerrados en el área resaltada en rojo en la Figura 15, se observan los pines mínimos que deben soldarse en los agentes para poder utilizar las placas de expansión. Con estos pines soldados, se puede continuar con el proceso de montado.

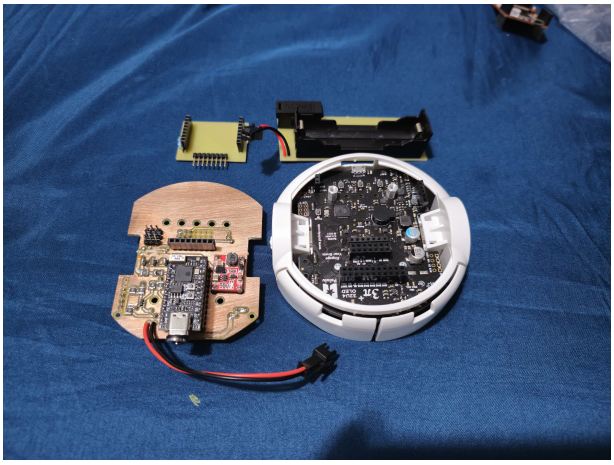


Figura 16: Placas de expansión y el agente.

En la Figura 16 se presentan las placas de expansión que se pueden montar en el agente. A continuación, se describe el orden correcto para ensamblarlas entre sí.

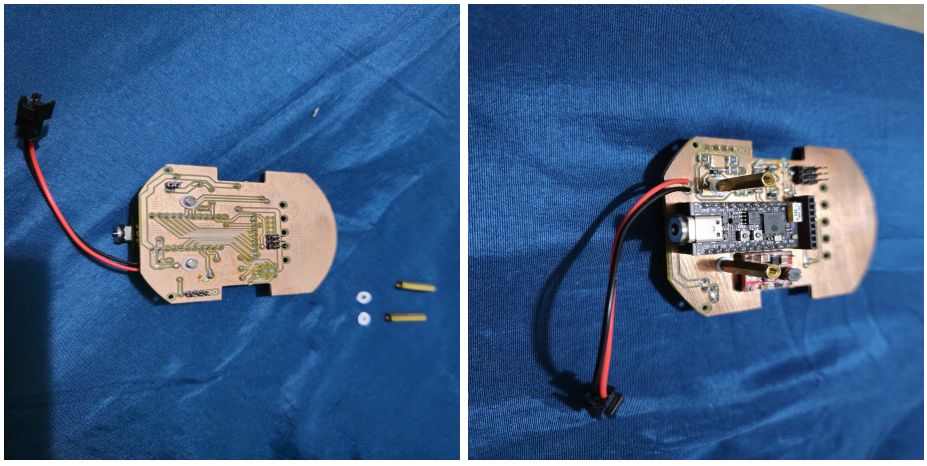


Figura 17: Tornillos, espaciadores plásticos y espaciadores de cobre.

En la Figura 17 se muestra la ubicación para colocar los tornillos y las arandelas de plástico necesarios para montar los espaciadores de cobre hembra - hembra. Una vez que los tornillos están en su lugar, se procede a fijar los espaciadores de cobre en el lado opuesto, colocando arandelas plásticas en el lado que entra en contacto con la placa y se aseguran atornillando.

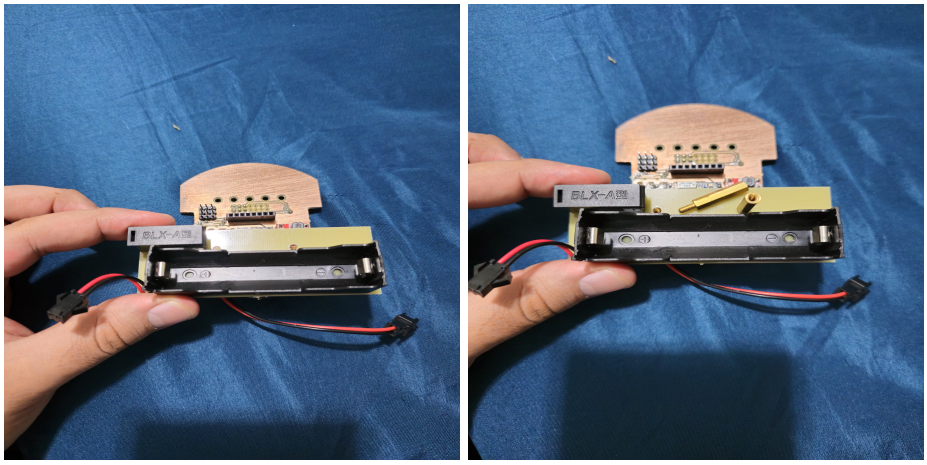


Figura 18: Espaciadores y placa de batería.

Con los espaciadores ya en su lugar, el siguiente paso consiste en montar la placa de la batería, tal como se muestra en la Figura 18. Posteriormente, se asegura fijando los espaciadores hembra-macho atornillándolos.

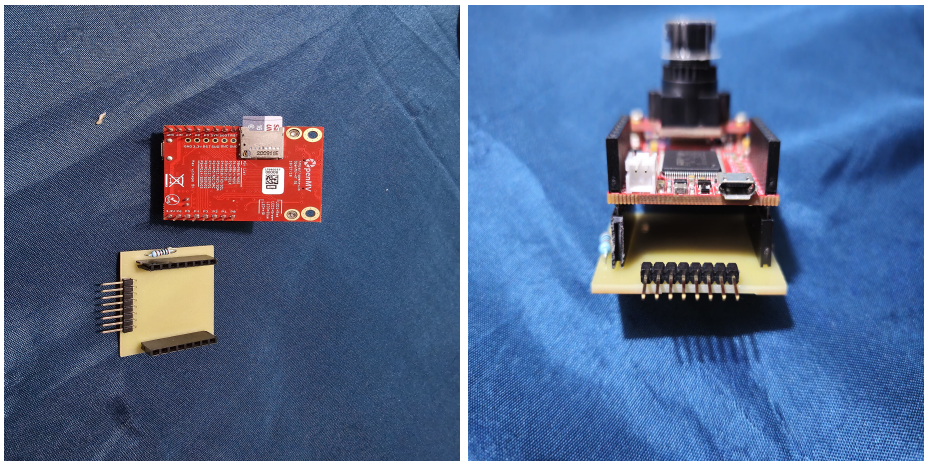


Figura 19: Montando la cámara en su placa.

Con la placa de la batería instalada, el siguiente paso es montar la cámara en su placa, como se ilustra en la Figura 19. Este procedimiento implica simplemente insertar la cámara en la placa, asegurándose de seguir la orientación indicada en la imagen.

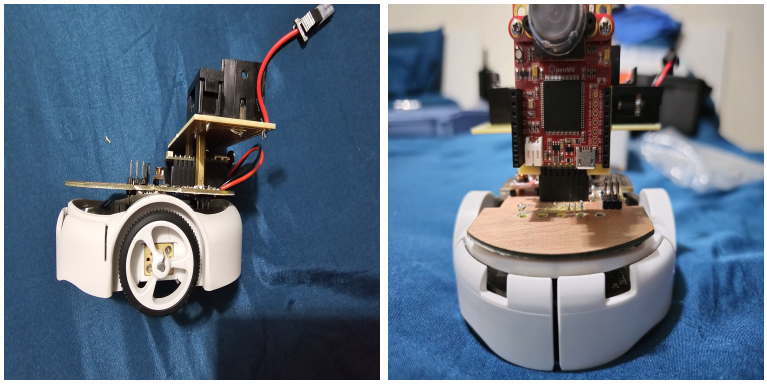


Figura 20: Montando la placa principal y la cámara en su placa.

Después de tener la placa de la batería y la cámara correctamente montadas, el siguiente paso consiste en instalar la placa principal en el agente. Este proceso implica simplemente insertar la placa principal con la orientación adecuada, como se muestra en la Figura 20. Es fundamental asegurarse de que los pines estén correctamente alineados para evitar cualquier posible deformación al presionar la placa. Una vez que la placa principal está montada, se puede proceder a fijar la cámara en su lugar siguiendo la orientación observada en 20.

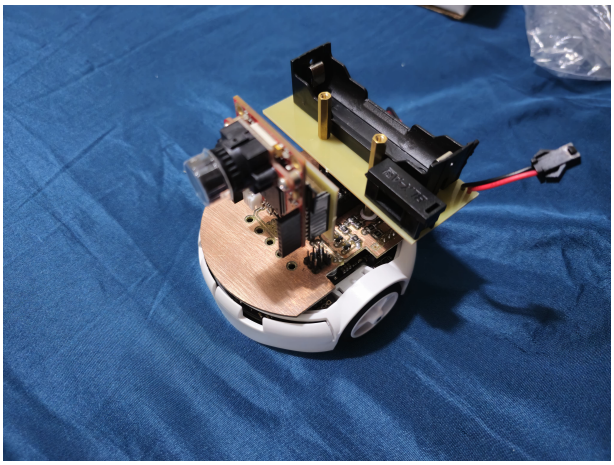


Figura 21: Montaje de la placa principal y la cámara en la placa principal.

La Figura 21 muestra el agente con la placa principal, la placa de la batería y la cámara correctamente montadas. Si se desea incorporar un marcador del Robotat en el agente, se puede realizar el siguiente montaje.

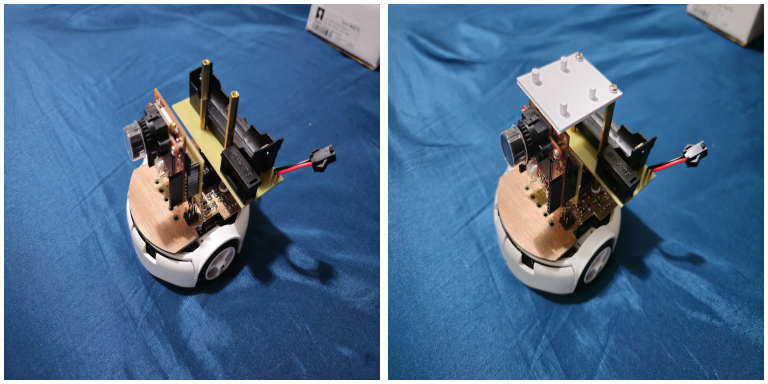


Figura 22: Cómo ensamblar el marcador.

Al buscar agregar el marcador del Robotat cómo se ve en 22, primero se deben agregar espaciadores de cobre hembra-macho en los espaciadores de la batería. Con los espaciadores montados, se añade el soporte para el marcador impreso en 3D, y este se puede asegurar con tornillos M3 en los últimos espaciadores montados.

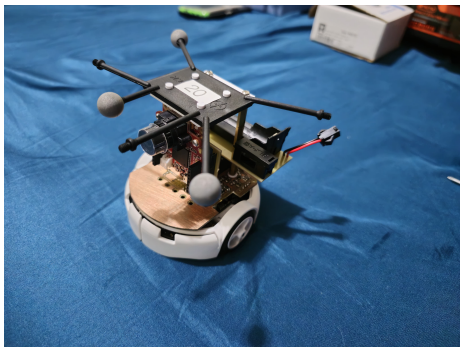


Figura 23: Agente final con todas las placas montadas y el soporte para marcador.

En la Figura 23 se puede observar al agente ensamblado y con el marcador del Robotat montado.

6. Ensamble del manipulador serial

6.1. Piezas a utilizar.

Descripción	Cantidad
Kit de eslabones y base.	1
Kit del efector final.	1
Servomotores MG90.	3
Tornillos M3X10	2

Cuadro 5: Listado de materiales a utilizar para ensamblar el manipulador serial.

Figura 24: Piezas para ensamblar los eslabones del manipulador serial.

Como se puede observar en la Figura 24, se necesitan 7 componentes para ensamblar la estructura de los eslabones y la base. Cinco de estos componentes están incluidos en el kit de eslabones y base, el cual se debe imprimir, y los archivos STL para la impresión se encuentran disponibles en el repositorio. Los otros 2 componentes son los brazos individuales que vienen incluidos con los servomotores al momento de la compra.

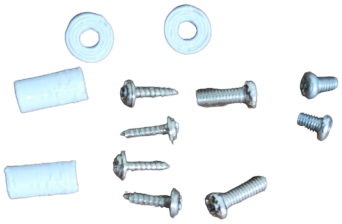


Figura 25: Tornillos a utilizar en el ensamble de los eslabones del manipulador.

En 25, se pueden observar los tornillos, arandelas y pines que se utilizarán para ensamblar el manipulador serial. Los pines y arandelas se imprimirán. Se utilizan 4 tornillos de 2mm incluidos con los servomotores, 2 tornillos de 2.5 mm cortos incluidos en los servomotores y 2 tornillos M3X10 comprados por separado.

Figura 26: Servomotores a utilizar en el manipulador serial.

Los actuadores a utilizar en este manipulador serial son servomotores MG90 como se ve en 26, son necesarios 3.

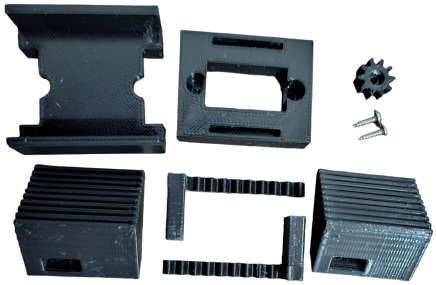


Figura 27: Piezas para ensamblar el efector final.

Para ensamblar el efector final del manipulador se requiere del kit del efector final, el cual debe imprimirse y los archivos STL se encuentran disponibles en el repositorio. Además, se necesitan 2 tornillos de 2 mm de longitud que se utilizan para montar los servomotores. En la Figura 27 se pueden ver los componentes que se utilizan en este proceso.

6.2. Cómo ensamblar los eslabones y el efector final.

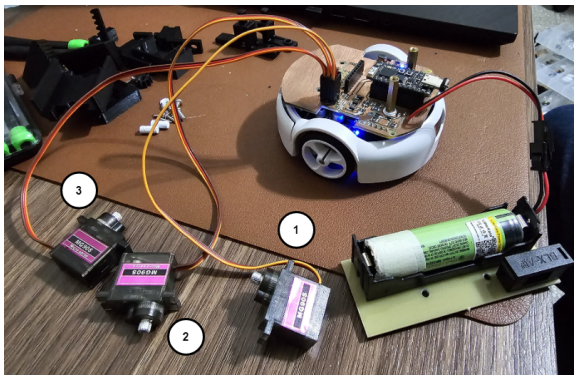


Figura 28: Piezas para ensamblar el efector final.

Al contar con todos los elementos impresos e identificados los componentes a utilizar, se comienza preparando los servomotores como se ve en 28 corriendo el script "PreparacionManipulador.py"del repositorio. Esto se realiza con la placa montada y los servomotores conectados. El script colocará a los servomotores en una configuración que permite montar el manipulador serial en posición horizontal, facilitando el proceso de ensamblaje.

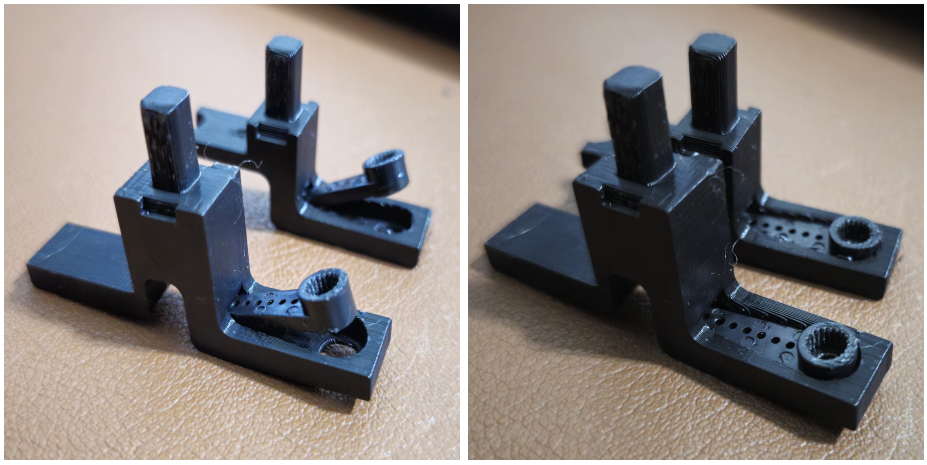


Figura 29: Como insertar los brazos del servomotor en los eslabones.

Con los servomotores en posición, se continúa con los pasos de la Figura 29. En esta se indica cómo insertar los brazos para luego presionarlos y así quedar en posición en los eslabones.

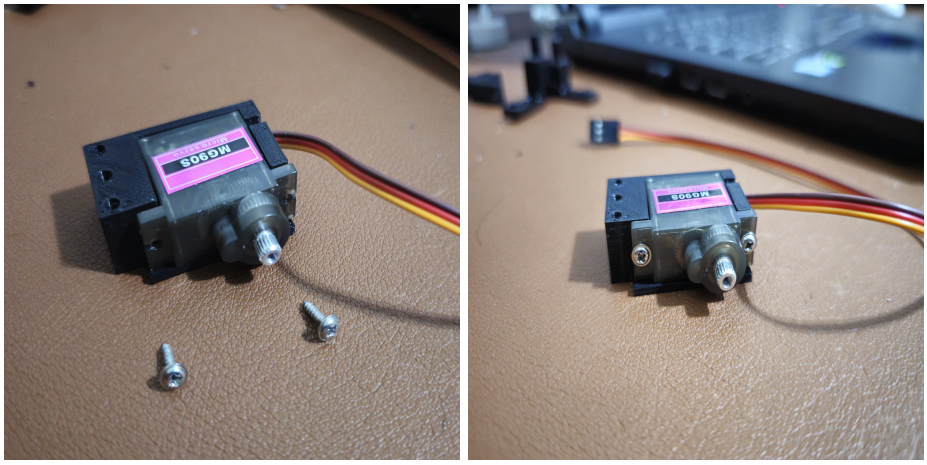


Figura 30: Colocando el primer servomotor.

Al haber colocado los brazos en los eslabones, se debe montar el servomotor marcado con 1 en la base, como se ve en 30, y ubicar los tornillos de 2 mm largos, que se montarán en los costados del servomotor para sujetarlo a la base.

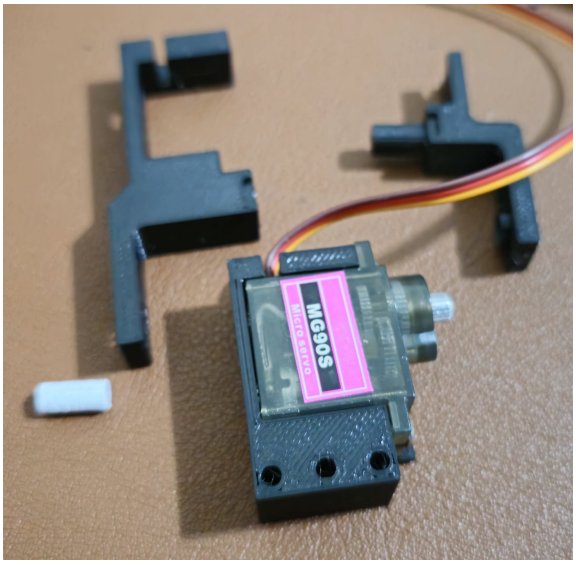


Figura 31: Piezas para el primer eslabón

Con el primer servomotor montado en la base, se continúa identificando los elementos para el manipulador serial del primer eslabón, estos se pueden ver en la Figura 31. Estas son las 2 partes del eslabón, una de ellas con el brazo del servomotor montado y un pin para dar otro punto de anclaje al manipulador.

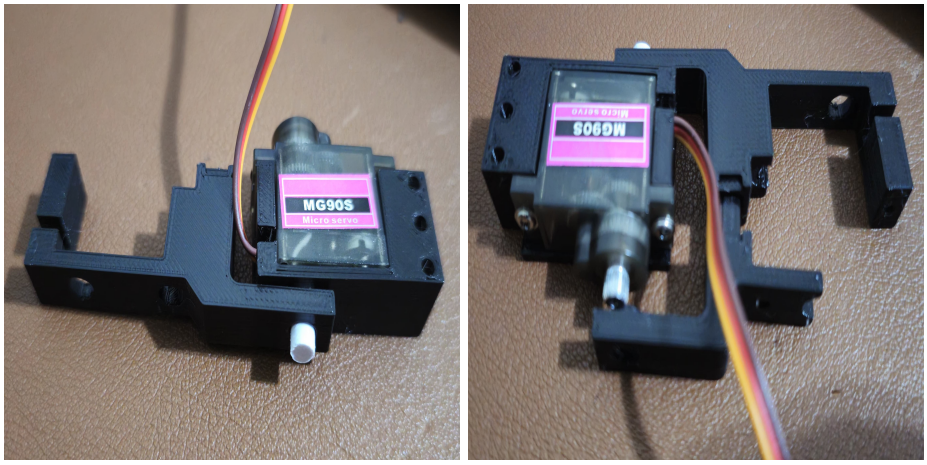


Figura 32: Montando el primer eslabón.

Al haber identificado los componentes del eslabón, se proceden a montar como se ve en la Figura 32, colocando primero el elemento más grande del eslabón y el pin en la base, luego montando de manera horizontal la otra parte que contiene el brazo del servomotor y conectar ambas partes con algo de presión.

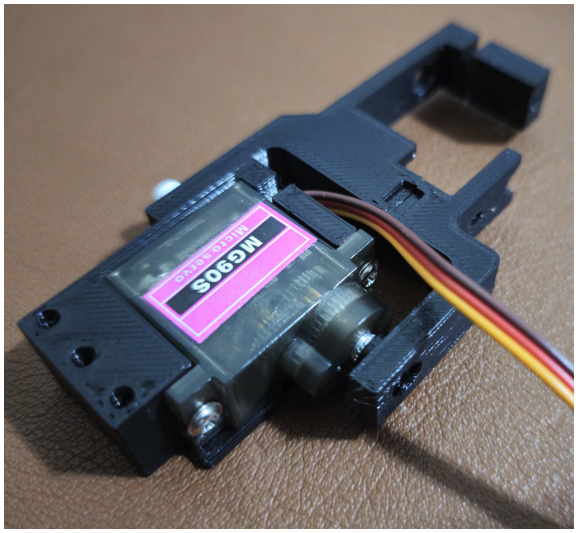


Figura 33: Primer eslabón montado.

Al tener el primer eslabón montado como se ve en la Figura 33, se termina por colocar el tornillo para el brazo del servomotor y así terminar de ensamblarlo.

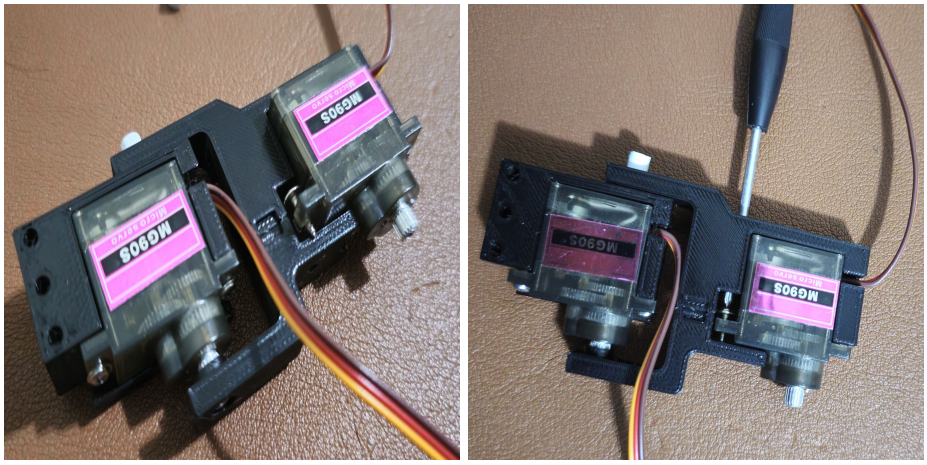


Figura 34: Cómo insertar el segundo servomotor.

Con el primer eslabón armado, se continua colocando el servomotor 2 de la Figura 28 en el eslabón 1. A este primero se le coloca el tornillo M2 largo en el orificio más alejado del conector del servomotor y se inserta en el primer eslabón como se ve en la Figura 34, luego, cuando ya está insertado utilizando el canal dejado en el primer eslabón se puede atornillar y fijar al primer eslabón. Al terminar de apretar ese tornillo se puede colocar otro M2 largo en el otro extremo del servomotor y así terminar de fijarlo al primer eslabón.

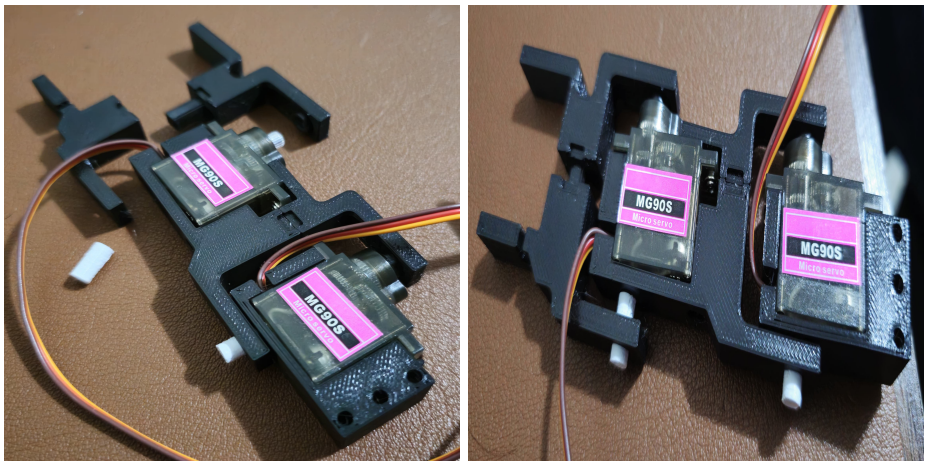


Figura 35: Piezas y como ensamblar el segundo eslabón.

Luego de terminar de armar el primer eslabón se deben ubicar las partes para el segundo eslabón, siendo estas las que se ven en la Figura 35. Este eslabón se puede ensamblar como el anterior, por lo cual solo se comparte cómo debería quedar montado en la Figura 35. Al terminar de montarlos solo se debe de colocar el tornillo en el brazo del servomotor y quedaría completa la montura de todos los eslabones.

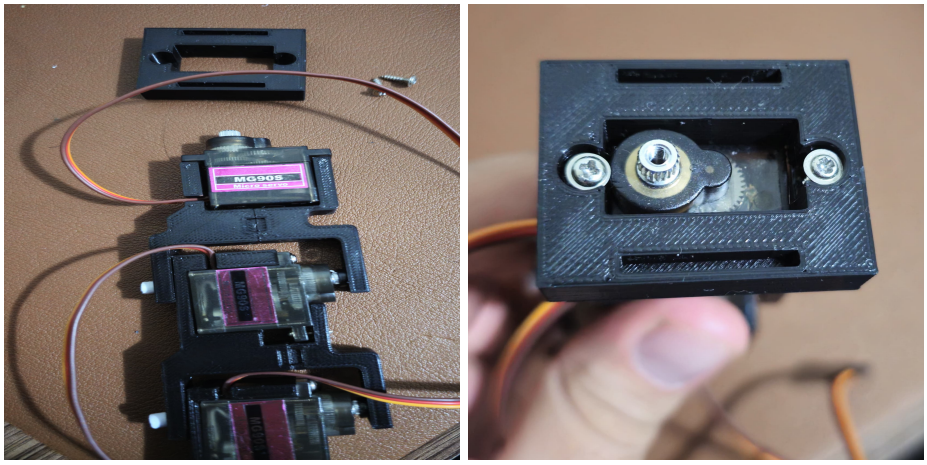


Figura 36: Cómo ensamblar la base del efector final.

Ahora se sigue con el ensamble del efector final, primero se debe montar el servomotor 3 de la Figura 28, ubicándolo como se ve en la Figura 36 en el eslabón 2 y solo mantenerlo en su posición. Luego, con el servomotor montado se debe montar la base del efector final en este último servomotor como se ve en la Figura 36, con esto colocado se pueden colocar los 2 tornillos largos de 2mm para los servomotores en sus costados y apretarlos, lo que dejará fija la base con el segundo eslabón.

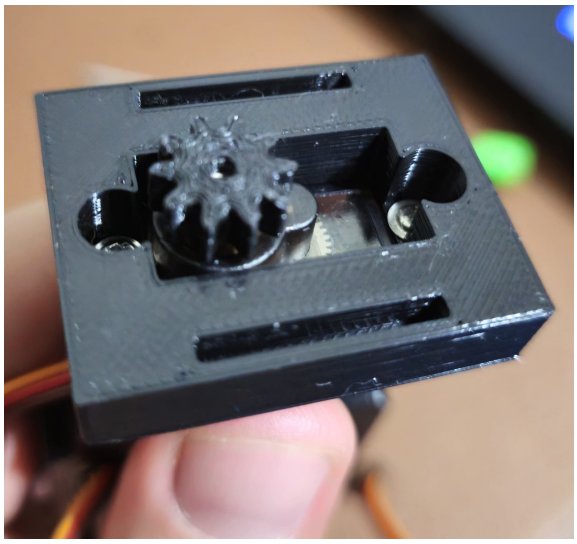


Figura 37: Engrane para el efector final.

Con esta base montada se puede montar el engrane en el ultimo servomotor, como se ve en 37. Este solamente entra a presión.

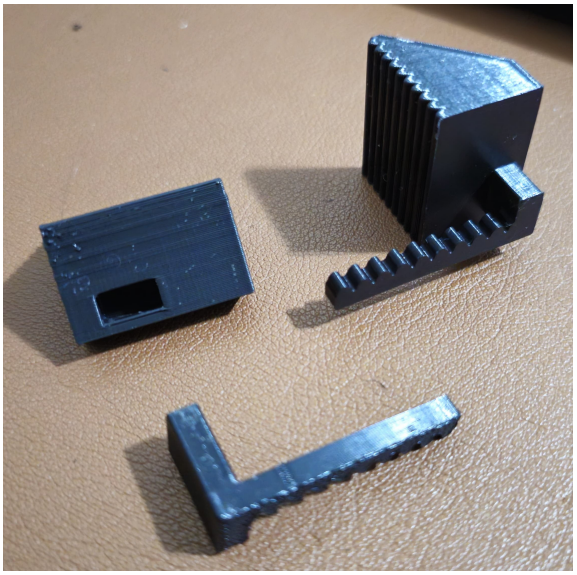


Figura 38: Garras del efector final.

Al montar el engranaje en el servomotor, se puede continuar montando las garras con las cremalleras para el efector final, como se ve en la Figura 38. Estas de igual manera entran a presión y se debe asegurar de que la cremallera quede orientada hacia la superficie con relieve de la garra para asegurar un correcto funcionamiento.

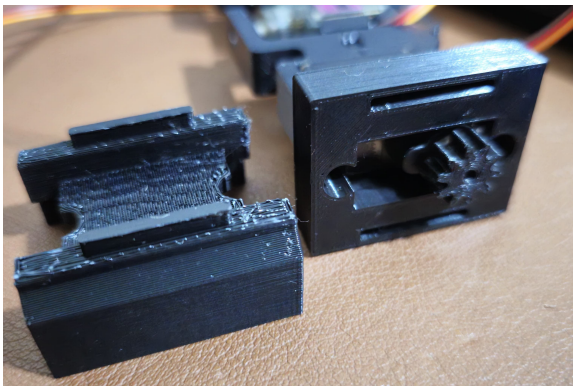


Figura 39: Parte superior efector final.

Al terminar de ensamblar la base y las garras del efector final, se debe ubicar la parte superior, cómo se ve en la Figura 39. Esta entra a presión en la base y mantiene la posición de las garras.

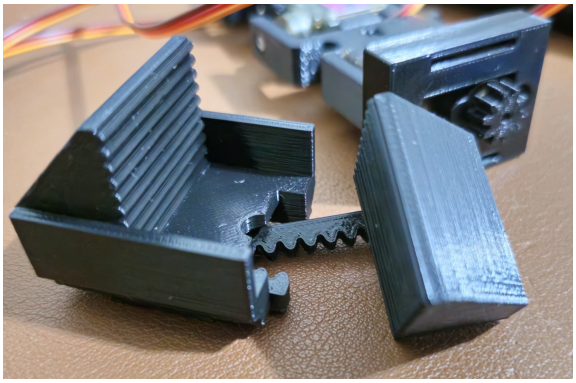


Figura 40: Montando las garras en la parte superior.

Con esta parte ubicada, los pines que entran en la base del efector final se pueden orientar hacia abajo, como se ve en 40. Con la parte superior ya orientada, se puede deslizar las garras con las cremalleras montadas en el hueco inferior de esta parte, como se ve en 40. Se debe asegurar que los dientes de la cremallera queden orientados hacia el interior. Después de haberlos introducido, se deben juntar las garras y, agarrando toda la pieza, montarla en la base del efector final. En este momento, se requiere mover ligeramente las garras hasta que toda la pieza encaje y quede en su lugar con las garras total o parcialmente cerradas.

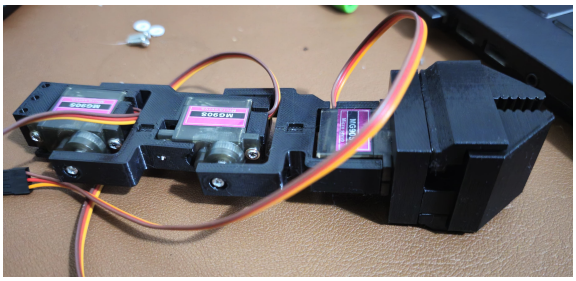


Figura 41: Manipulador terminado.

Con el último paso se terminó de ensamblar el manipulador serial y este quedaría como se ve en la Figura 41. Al tener el manipulador serial ensamblado, en el capítulo de montura se mencionará la prueba a realizar para su correcto funcionamiento.

7. Cómo montar el manipulador serial

7.1. Materiales a utilizar.

Entre los elementos requeridos para montar el manipulador serial están:

- 2 tornillos M3X20
- 2 arandelas de plástico.
- El manipulador serial ensamblado.

7.2. Cómo ensamblarlo.

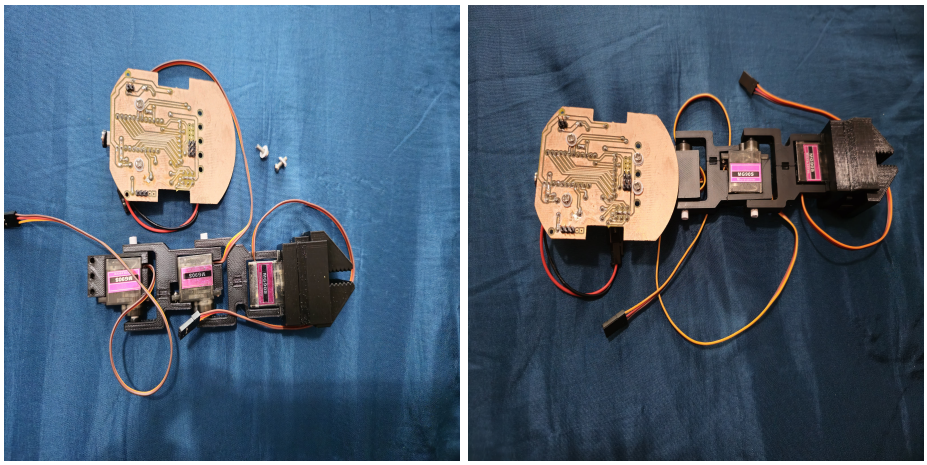


Figura 42: Montando el manipulador serial en la placa principal.

Como se observa en 42, para ensamblar el manipulador serial en la placa principal, primero se debe desmontar la placa para poder colocarlo. Con la placa afuera, se debe orientar como se ve en la Figura 42, y a la vez, alinear los agujeros de la base del manipulador serial con los 5 agujeros frontales, procurando alinearlos con sus centros. Al tener esto alineado, con los tornillos M3 y las arandelas plásticas, se comienza a apretar hasta que ambas piezas estén juntas. Luego, solo es montar la base de regreso y conectar en orden los servomotores como se ve en 1.

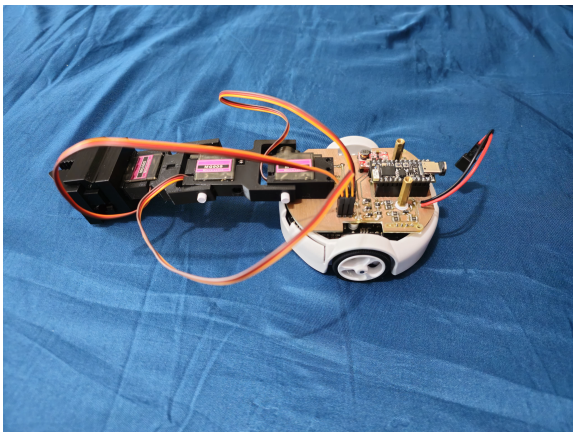


Figura 43: Manipulador serial montado

En la Figura 43 se observa el manipulador serial montado y conectado a la principal, junto con el manipulador serial se puede montar el marcador del robotat para el seguimiento de trayectorias. Ya que se cuenta con el manipulador ensamblado en la placa principal, se puede correr la última prueba **Tercera etapa** (véase 8.3).

8. Pruebas de funcionamiento.

Todos los vídeos mencionados en las siguientes secciones se encuentran en el repositorio en "Videos >Pruebas", los nombres utilizados en este documento son los mismos nombres que se encuentran en la carpeta, con la diferencia que acá se omitirá la extensión. Todos los scripts presentados en este documento se encuentran en el respositorio en ÇontrolCliente >Pruebas".

Para todas las pruebas se requiere conectar a la red WiFi del agente.

8.1. Primera etapa

En esta prueba se usará el script **Script_01** en ella el agente se encenderá durante 5 segundos y luego se apagará, la manera en que se puede ver si está encendido o apagado es con la luz azul que enciende en su placa u observando la luz azul que se refleja en la parte inferior del agente. El vídeo para observar este comportamiento es **01_PrimeraEtapa**.

8.2. Segunda etapa

En esta segunda prueba, la cual se divide en tres scripts, se busca confirmar el correcto funcionamiento de la comunicación entre el agente y la cámara. Para llevar a cabo estos escenarios, es necesario tener montada la placa principal en el agente, junto con la cámara.

El primer script, denominado **Script_02_01**, induce al agente a girar sobre sí mismo durante 3 segundos en ambas direcciones. Posteriormente, lo mueve hacia adelante y hacia atrás en intervalos de 5 segundos, finalizando con la acción de apagado. Puedes observar este comportamiento en el archivo de vídeo **02_01_SegundaEtapa**.

El segundo script, **Script_02_02**, se centra en la captura de una imagen de la vista actual del agente. Este resultado gráfico se visualiza y luego el agente se apaga. Este comportamiento se puede apreciar en el vídeo **02_02_SegundaEtapa**.

Finalmente, el tercer script, **Script_02_03**, proporciona una transmisión en tiempo real de la visión del agente, mostrándola en la pantalla durante 20 segundos, luego se cierra la pantalla y se apaga el agente. Este comportamiento se muestra en el vídeo **02_03_SegundaEtapa**.

8.3. Tercera etapa

En esta tercera prueba, se utilizará el script **Script_03**. Este script permite que el manipulador serial realice la forma de un cuadrado, se estire completamente en horizontal y vertical. Finalmente, se coloca en la posición de reposo, donde abre y cierra el efector final tres veces en diferentes tamaños. Este comportamiento específico se muestra en el vídeo **03_TerceraEtapa**.

9. Observaciones

En este apartado, se pueden añadir observaciones sobre el proceso de ensamblaje o cambios realizados al agente en futuras iteraciones. Se pueden describir los cambios realizados y los archivos modificados para llevar un mejor control en futuras iteraciones.

CAPÍTULO 16

Glosario

Handshake : En el contexto de protocolos o buses de comunicación, se refiere a un intercambio de información definida entre dos dispositivos antes de comenzar la comunicación. Este intercambio puede consistir en un mensaje o una señal física que establece las condiciones iniciales y los parámetros necesarios para que la comunicación entre los dispositivos sea exitosa.. 32

Socket : Es una abstracción de software en la capa de comunicación WiFi utilizada. Esta abstracción representa el punto final de la comunicación y se crea dentro del elemento que se va a conectar. Utiliza el protocolo seleccionado como medio de transmisión.. 8