

# Proyecto final de estudios

Exploración aumentada mediante robot holonómico diferencial con visión robótica para búsqueda y rescate

Alumno

García Girón Maximiliano José Andrés

Legajo: 9885

## Contenido

Introducción .....	5
Objetivos .....	9
Metodología .....	9
Hipótesis de trabajo .....	9
Alcance y limitaciones del proyecto.....	9
Actividades realizadas .....	10
Desarrollo esquemático del robot .....	11
Cinemática del robot.....	12
Cinemática inversa .....	12
Cinemática directa .....	14
Dinámica del robot.....	14
Diseño mecánico del robot .....	17
Diseño de piezas mecánicas .....	17
Chasis de motor PAP y eje.....	17
Soporte de batería .....	19
Base inferior .....	19
Base intermedia .....	20
Base superior.....	21
Soporte de servomotor y cámara .....	22
Parámetros cinemáticos y dinámicos finales .....	24
Selección de actuadores.....	24
Componentes electrónicos y conexiones eléctricas .....	24
Componentes electrónicos .....	24
Diagramas de conexiones .....	25
Consideraciones eléctricas .....	26
Diagrama de conexiones generales .....	27
Subsistema Raspberry-Cámara .....	28
Subsistema de energía .....	29
Subsistema Pololu A4988-Stepper .....	30
Subsistema MPU6050 .....	31
Subsistema Servomotor .....	32

Sistema de navegación asistida.....	32
Consideraciones previas.....	32
Descripción de los sensores .....	33
Descripción de la planta en espacio de estados .....	33
Descripción del filtro de Kalman .....	34
Ajuste según la gravedad terrestre .....	35
Resultados en Matlab.....	37
Implementación .....	41
Control de actuadores.....	41
Control en Arduino con librería Accelstepper.....	41
Control de velocidades angulares en Unity.....	43
Inteligencia artificial .....	45
Generación del dataset .....	45
Procesamiento y aumentación del dataset.....	47
Entrenamiento .....	50
Implementación .....	52
Comunicaciones .....	52
Comunicación Wi-Fi .....	53
Inicio y encendido .....	55
Interfaz de usuario .....	55
Sección Cámara .....	57
Sección Estado.....	58
Sección Controles.....	58
Sección Posición y Orientación .....	59
Sección FPS y Temperatura .....	59
Sección Minimap y Mapa .....	59
Materiales y costos .....	61
Possibles mejoras.....	62
Conclusiones .....	63
Palabras Clave .....	63
Bibliografía .....	63
Anexo: Códigos Arduino (C++) .....	66

ProyectoFinalArduino.ino .....	66
CoordinadorEjes.h .....	67
CoordinadorEjes.cpp .....	68
Comunicacion.h .....	71
Comunicacion.cpp .....	72
Navegacion.h .....	75
Navegacion.cpp .....	76
Estructuras.h .....	79
Anexo: Código Raspberry (Python) .....	81
Proyecto.py .....	81
Comunicacion.py .....	81
Tensorflow.py .....	85
Anexo: Códigos de inicio Raspberry .....	89
ProyectoFinal.sh .....	89
ProyectoFinal.desktop .....	89
Anexo: Código Terminal (Unity C#) .....	90
Minimap.cs .....	90
LineDraw.cs .....	93
Interfaz.cs .....	94
Lector_USB.cs .....	98
Raspberry_Comunicacion .....	102
Jerarquía de escena .....	108
Anexo: Código Dataset Generator (Unity C#) .....	109
MyActivationTag.cs .....	109
MyRotationRandomizerTag.cs .....	110
MyLightRandomizerTag.cs .....	111
Jerarquía de escena .....	112

## Introducción

De forma previa al desarrollo del presente proyecto es necesario definir algunos términos y antecedentes históricos, la palabra “robot”, según Barrientos, fue utilizada por primera vez en el año 1921 en la obra teatral “Rossum’s Universal Robot” escrita por el autor Karel Capek, siendo robot la palabra derivada del eslavo “robota”, refiriendo al trabajo realizado de manera forzada, en dicha obra los robots desarrollan los trabajos que los humanos no querían realizar, hasta que se rebelaron. Posteriormente el término se popularizó en 1945 gracias al escritor Isaac Asimov, el cual publicó en la revista “Galaxy Science Fiction” una historia donde se enunciaron y popularizaron por primera vez su versión de las tres leyes de la robótica, siendo las siguientes:

1. Un robot no puede perjudicar a un ser humano, ni con su inacción permitir que un ser humano sufra daño.
2. Un robot ha de obedecer las órdenes recibidas de un ser humano, excepto si tales órdenes entraran en conflicto con la primera ley.
3. Un robot debe proteger su propia existencia mientras tal protección no entre en conflicto con la primera o la segunda ley.

Posteriormente Asimov añadiría una cuarta ley, llamada ley cero, la cual dicta lo siguiente:

4. Un robot no puede dañar a la humanidad, ni con su inacción permitir que la humanidad sufra daños.

Dichas leyes, si bien no tienen una aplicación directa, han tenido un impacto en el imaginario colectivo, así como en la ética aplicada a la inteligencia artificial, según Wikipedia.

Sin embargo, desarrollos en la materia ya existían mucho antes de ser acuñado el término, solo a modo de dar un ejemplo, el primer autómata (la RAE en su segunda acepción lo define como máquina que imita la figura y los movimientos de un ser animado) sería desarrollado por Herón de Alejandría entre los años 10 y 70 d.C. y consistiría en aves que se movían y estatuas que servían vino. Los primeros robots móviles (entiéndase robot móvil como el robot que es capaz de realizar locomoción y tomar decisiones sobre la navegación) fueron inventados entre los años 1939 y 1945, durante la segunda guerra mundial, sirviendo como piloto automático y sistema de detonación de los cohetes V1 y V2 alemanes. Posteriormente en 1948-1949 se desarrollaron los Machina Speculatrix, dos robots sencillos con sensores de luz y una lógica analógica equivalente a una neurona (llamada perceptrón), siendo capaces de encontrar y moverse hacia una fuente de luz, evitando obstáculos. En 1970 la URSS envió su primer rover lunar radiocontrolado, el Lunokhod 1. En 1971 la URSS envió el primer rover capaz de navegación autónoma a la superficie marciana, en las misiones Mars 2 y Mars 3, el PrOP-M, sin embargo, la misión Mars 2 se estrelló sobre la superficie del planeta, mientras que la Mars 3 perdió la comunicación poco después de aterrizar debido a una intensa tormenta de polvo, por lo que dicho rover no pudo desplegarse del aterrizador. En 1976 el programa Viking envió dos sondas a Marte, las cuales consistían de un orbitador y un Lander en cada misión, ambas capaces de realizar el viaje, maniobras orbitales y aterrizaje de manera autónoma.

En cuanto a la visión robótica se pueden extraer como principales hitos históricos el invento del Kinescopio en 1929 siendo el primer sistema práctico para transmisión y recepción de imágenes. En 1950 el desarrollo

del tubo Vidicon, consistiendo en una cámara en forma de tubo con un fotoconductor, utilizada en las primeras sondas de espacio profundo de la Nasa. En 1965 Larry Roberts presenta su tesis "Machine Perception of Three-Dimensional Solids" o percepción de máquina de sólidos tridimensionales, en la cual demostró como una computadora podía producir un modelo 3D a partir de una imagen 2D. En 1966 "Shakey the Robot" fue el primer robot multipropósito, se controlaba por radio a distancia y poseía una cámara de TV, además de un sensor óptico de distancias. En 1976 el "ARPA Image Understanding Program" o programa de comprensión de imágenes de ARPA logró crear una técnica capaz de interpretar una escena a partir de una fotografía. A mediado de la década de los 80 se introdujo la primera cámara inteligente capaz de identificar objetos en un entorno de fábrica. En 1986 el sistema de plano epipolar de análisis de imagen o "Epipolar-Plane Image Analysis" fue capaz de reconstruir una escena completa en 3D a partir de una serie de imágenes. Avanzando a tiempos más actuales, en 2009 Google testeó su primer vehículo autónomo con capacidad de visión, en 2010 Google lanza su primera aplicación de detección de rostros para móviles, y en 2012 libera su primer algoritmo de reconocimiento de objetos en imágenes mediante Deep Learning, en 2013 los robots de Vision aprenden a adaptarse al trabajo con humanos, aprendiendo mediante Deep Learning las preferencias de los mismos.

En la actualidad, durante las operaciones de búsqueda y rescate, se presentan muchos riesgos que corren los rescatistas al acceder a una zona de desastre sin exploración previa, dicha exploración es llevada a cabo con frecuencia mediante robots, los cuales permiten a un operador identificar posibles riesgos tales como químicos peligrosos, focos de incendio, e incluso peligros de radiación, permitiendo a su vez identificar personas accidentadas y objetos de interés, muchas veces en lugares a priori inaccesibles por el personal humano. En algunos casos los robots van incluso más allá del simple reconocimiento, pudiendo desarmar artefactos peligrosos, extraer personas de la zona de catástrofe, incluso siendo capaz de extraer combustible de plantas nucleares en caso de accidente, como sucedió en la planta de Fukushima. Normalmente dichos robots de búsqueda son operados a distancia por alguien del personal de búsqueda y rescate, pudiendo realizar observaciones de la zona de desastre mediante cámaras y otros sensores, dependiendo en gran medida del criterio del operario para discernir e identificar los elementos del entorno, sin embargo, en este proyecto se propone la posibilidad de añadir una mejora a la interfaz entre humano y máquina, dicha mejora consistirá en un sistema de visión robótica, el cual permitirá mediante realidad aumentada asistir al operario del robot, ayudando a distinguir distintos objetos del entorno, el cual se define como un entorno no estructurado y cambiante con el tiempo.

En el proyecto actual se propone el realizar una plataforma móvil sencilla donde poder realizar pruebas y verificar la viabilidad de implementar visión robótica en las operaciones de búsqueda y rescate, para ello se ha decidido diseñar y construir un robot de tipo holonómico diferencial, ya que permitirá una navegación y control sencillos, priorizando el desarrollo de la visión robótica y la interfaz de usuario como objetivos principales. Un esquema simplificado de la plataforma móvil se muestra a continuación.

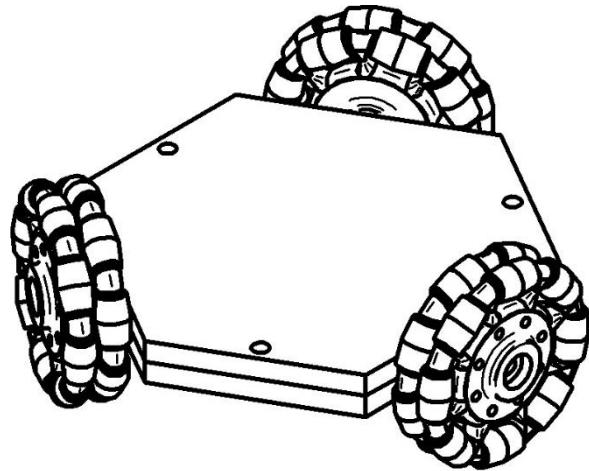


Figura 1: Esquema simplificado de robot holonómico.

Se tomará como base para la interfaz un proyecto anterior del alumno que presenta el actual proyecto, el cual fue realizado para la materia de “Realidad virtual” de carrera “Ingeniería en Mecatrónica” de la Universidad Nacional de Cuyo, dicha terminal contendría los elementos básicos que se desea presentar, sin tener implementada la visión robótica. Se presentan a continuación capturas de lo anteriormente mencionado:

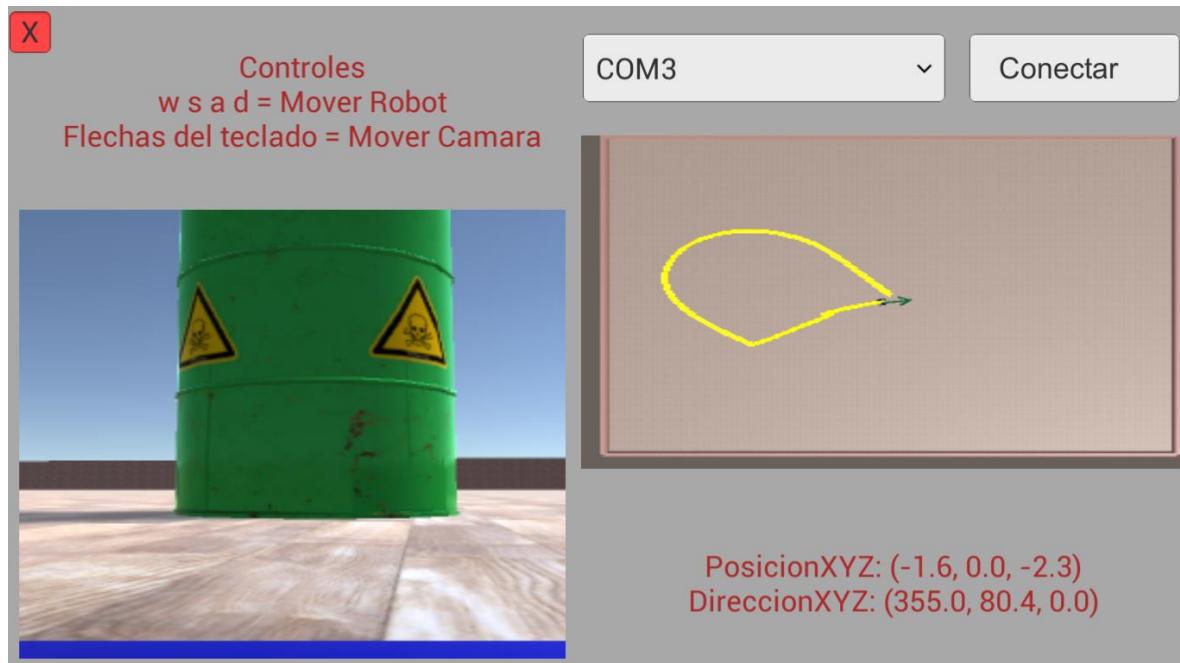


Figura 2: Esquema de interfaz, proyecto de Realidad Virtual.



Figura 3: Esquema de interfaz, proyecto de Realidad Virtual.



Figura 4: Esquema de interfaz, proyecto de Realidad Virtual con detección agregada.

En la primer y segunda imagen se presenta la interfaz sin editar con sus elementos básicos, la misma contiene las opciones de conexión para detectar y conectar con un robot, un mapa con las coordenadas de posición y dirección del robot, en dicho mapa se puede observar que se encuentra a modo demostrativo una flecha que indica visualmente la orientación del robot en verde y en amarillo el

recorrido que ha realizado, también se muestran las instrucciones de movimiento y la vista de la cámara del robot, en la tercera imagen se muestra como se pretende que se visualice la identificación de objetos.

## Objetivos

Los objetivos de desarrollo planteados para el presente proyecto son:

- Desarrollar y construir un prototipo de robot del tipo holonómico diferencial a modo de plataforma móvil.
- Desarrollar un sistema de navegación para que el operario pueda ubicar al robot en el espacio en cualquier momento.
- Desarrollar un sistema de visión robótica que brinde asistencia al operario para poder identificar objetos clave en su entorno.
- Desarrollar una interfaz que permita al operario visualizar los datos enviados por el robot, así como los que se procesen localmente.

Se considera el siguiente objetivo opcional:

- Presentar el proyecto a miembros de los cuerpos de bomberos emplazados en la ciudad de Mendoza para que ofrezcan su opinión y sugieran posibles mejoras.

## Metodología

En esta sección se definen los alcances y limitaciones del proyecto, así como un breve resumen de las actividades realizadas considerando los tiempos calculados para cada una de ellas.

## Hipótesis de trabajo

El presente proyecto es realizado bajo la hipótesis de que un sistema de reconocimiento de objetos, unido a una interfaz que aporte información extra al operario puede ser de gran utilidad en las operaciones de búsqueda y rescate, reduciendo el riesgo de los rescatistas y aumentando su efectividad.

## Alcance y limitaciones del proyecto

El presente proyecto entra dentro de los requisitos de aprobación de la asignatura Proyecto Final de Estudios de la carrera Ingeniería en Mecatrónica, por lo tanto, se considera oportuno recordar la naturaleza académica del mismo, por lo que varios elementos se presentarán y desarrollarán de forma didáctica.

Se planteará la cinemática y dinámica de un robot holonómico diferencial de forma aproximada, sin considerar vibraciones, fricciones internas u otros factores externos como la resistencia del aire, considerando en todo momento el contacto total de la rueda con el suelo.

Se planteará la navegación del mismo mediante un Filtro de Kalman que obtendrá datos de un conjunto acelerómetro-giróscopo MPU6050 y de un motor Paso a Paso que se utilizará como encoder, además de actuador, bajo la idealización de que este no pierde pasos, dicha navegación será simplificada con el caso

particular de navegación en dos dimensiones, de forma tal de asistir al operario, pero sin agregar más complejidad de la necesaria en dicho sistema.

El sistema de reconocimiento de objetos es el que tendrá mayor peso en el presente proyecto, sin embargo, dicho sistema deberá reconocer símbolos y señales bajo el estándar IRAM 10005 (dicho estándar a su vez extrae símbolos del estándar ISO 7010, por lo que existe cierta compatibilidad), no pudiendo reconocer señales y símbolos fuera de dicho estándar, además no se considera el reconocimiento de personas, quedando esto a cargo del operario.

## Actividades realizadas

Para el presente proyecto se proponen las siguientes actividades con su respectiva complejidad estimada por disciplina en una escala del 1 al 5:

Actividad	Disciplina/Complejidad
Desarrollo esquemático del robot	Mecánica/1
Formulación de cinemática y dinámica del robot	Mecánica/2
Desarrollo de planos finales	Mecánica/4
Desarrollo de la navegación asistida	Control/4
Programación del controlador	Informática/3
Conexión y comprobación de los componentes electrónicos	Electrónica/4
Armado del robot	Mecánica/5
Desarrollo e implementación del sistema de visión robótica	Informática/5
Desarrollo de la interfaz	Informática/3
Desarrollo del informe final	General/2

*Tabla 1: Actividades propuestas y complejidad estimada.*

A continuación, se presenta el cronograma planeado para las actividades previamente mostradas, dicho cronograma se usó de forma tentativa en la etapa de anteproyecto y finalmente no fue cumplido debido a una gran variedad de dificultades relacionadas al proyecto y personales. Tanto en el cronograma propuesto como en el desarrollo real el informe se realizó en conjunto con el resto del proyecto. Dicho cronograma es el siguiente:

Actividad	Tiempo en semanas (estimado)														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Desarrollo esquemático del robot															
Formulación de cinemática y dinámica del robot															
Desarrollo de planos finales															
Desarrollo de la navegación asistida															
Programación del controlador															
Conexión y comprobación de los componentes electrónicos															
Armando del robot															
Desarrollo e implementación del sistema de visión robótica															
Desarrollo de la interfaz															
Desarrollo del informe final															

Tabla 2: Cronograma propuesto.

## Desarrollo esquemático del robot

En esta sección se presentan algunos esquemas del robot considerado, dichos esquemas tienen el objetivo de ilustrar tanto el sistema de ejes coordenados como la nomenclatura de las ruedas que se utilizarán durante el resto del proyecto. Dichos esquemas son los siguientes:

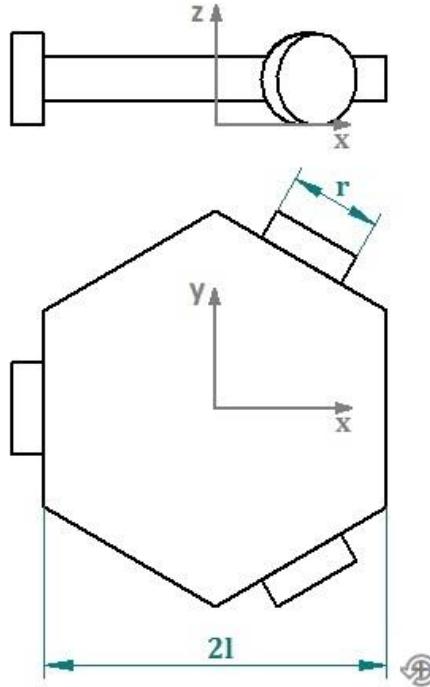


Figura 5: Definición de ejes de coordenadas y medidas.

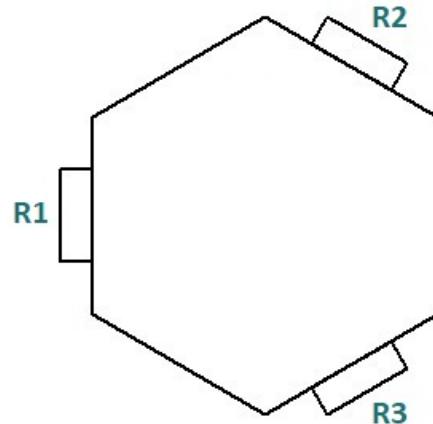


Figura 6: Numeración de las ruedas.

## Cinemática del robot

En esta sección se define la cinemática que permite el movimiento de un robot holonómico diferencial de tres ruedas, se obtiene tanto cinemática inversa como directa, además de la matriz jacobiana del sistema actuado (la matriz jacobiana completa incluiría los rodillos y las ruedas esféricas, pero no se tienen en cuenta debido a que estos incrementarían la complejidad del proyecto), se tiene en cuenta el sistema de ejes y ruedas presentado previamente en el desarrollo esquemático del robot, así como la siguiente imagen que representa la dirección en la cual se considera el sentido positivo del giro de las ruedas.

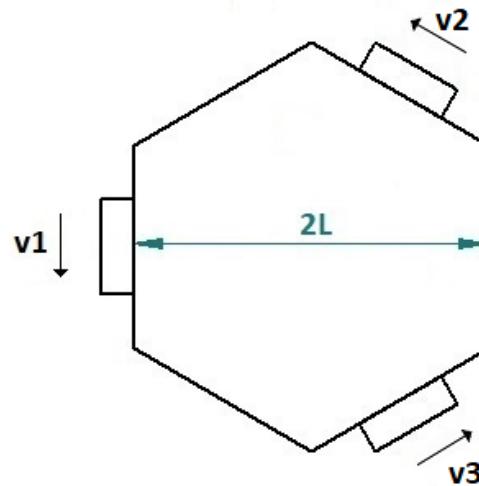


Figura 7: Definición del sentido positivo de giro de las ruedas.

## Cinemática inversa

Para definir la cinemática inversa, se define la velocidad lineal de cada rueda como:

$$r\omega_1 = L * \omega + V_1$$

$$r\omega_2 = L * \omega + V_2$$

$$r\omega_3 = L * \omega + V_3$$

Donde:

$r$  es el radio de la rueda.

$\omega_n$  es la velocidad angular de cada rueda.

$L$  es la distancia entre el centro geométrico del robot y la rueda.

$\omega$  es la velocidad angular del robot.

$V_n$  es la velocidad lineal de cada rueda.

También pueden definirse las componentes de las velocidades como:

$$v_x = V_n * \cos(\theta_n)$$

$$v_y = V_n * \sin(\theta_n)$$

Donde  $\theta_n$  es el ángulo entre la velocidad de la rueda y el eje x. Multiplicando la primer ecuación por  $\cos(\theta_n)$ , la segunda por  $\sin(\theta_n)$ , sumando ambas ecuaciones e intercambiando sus dos lados se obtiene:

$$V_n = v_x * \cos(\theta_n) + v_y * \sin(\theta_n)$$

Se considera:

$$\theta_1 = 270^\circ$$

$$\theta_2 = 150^\circ$$

$$\theta_3 = 30^\circ$$

Definiendo la velocidad tangencial de la rueda como:

$$V_n = \pi * d * \omega$$

Se puede expresar la relación entre las velocidades lineales y angular del robot, y la velocidad angular de cada rueda de la siguiente manera:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{\pi d} & \frac{L}{\pi d} \\ -\frac{\sqrt{3}}{2\pi d} & \frac{1}{2\pi d} & \frac{L}{\pi d} \\ \frac{\sqrt{3}}{2\pi d} & \frac{1}{2\pi d} & \frac{L}{\pi d} \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Siendo la matriz jacobiana actuada del sistema:

$$J = \begin{bmatrix} 0 & -\frac{1}{\pi d} & \frac{L}{\pi d} \\ -\frac{\sqrt{3}}{2\pi d} & \frac{1}{2\pi d} & \frac{L}{\pi d} \\ \frac{\sqrt{3}}{2\pi d} & \frac{1}{2\pi d} & \frac{L}{\pi d} \end{bmatrix}$$

## Cinemática directa

La cinemática directa del robot puede obtenerse a partir de la cinemática inversa invirtiendo la matriz jacobiana de la misma.

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} 0 & -\frac{\sqrt{3}\pi d}{3} & \frac{\sqrt{3}\pi d}{3} \\ -\frac{2\pi d}{3} & \frac{\pi d}{3} & \frac{\pi d}{3} \\ \frac{\pi d}{3L} & \frac{\pi d}{3L} & \frac{\pi d}{3L} \end{bmatrix} * \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

## Dinámica del robot

En esta sección se obtiene la dinámica inversa del robot, debido a su complejidad se opta por tomar como referencia el paper de “Modelado cinemático y dinámico de un robot móvil omni-direccional” listado en la bibliografía, sin embargo, por cuestiones de simplicidad se opta por hacer algunas simplificaciones, además de las planteadas en dicho paper, las cuales se enumeran a continuación.

- Se considera que el robot se mueve en forma horizontal, por lo que no se considera cambios en la energía gravitatoria del sistema.
- Se considera el modelo libre de fricción interna.
- Se considera que las ruedas apoyan perfectamente en el terreno y no existe deslizamiento.
- Por simplicidad, se considera que los rodillos no tienen inercia.

Para obtener la dinámica del sistema se recurre al método de Lagrange, el cual define que los torques de los actuadores son iguales a:

$$\tau_n = \frac{d}{dt} * \frac{\delta \mathcal{L}}{\delta \dot{q}_n} - \frac{\delta \mathcal{L}}{\delta q_n}$$

Donde  $\mathcal{L}$  es el lagrangiano del sistema, el cual se expresa de la siguiente forma:

$$\mathcal{L} = K - U$$

Donde  $K$  es la energía cinética del sistema, mientras que  $U$  es la energía potencial del sistema, la cual consideramos cero en este proyecto, por lo que el lagrangiano será igual a la energía cinética del sistema. Se define la energía cinética total como la suma de las energías parciales.

$$K = K_T + K_R + K_G$$

Donde  $K_T$  es la componente translacional de la energía cinética del sistema,  $K_R$  es la componente rotacional, y  $K_G$  es la aportación del centro de masa del robot en caso de estar descentrado al sistema de referencia, en nuestro caso  $K_G = 0$ .

Se define la energía cinética translacional como:

$$K_T = \frac{1}{2}M * (v_x^2 + v_y^2)$$

Donde  $M$ , la masa total del robot es igual a la suma de la masa del chasis  $M_c$  y la masa de las tres ruedas  $M_r$ .

$$M = M_c + 3 * M_r$$

Se define la energía cinética rotacional como:

$$K_R = \frac{1}{2} * I * \omega^2 + \frac{1}{2} * I_r * (\omega_1^2 + \omega_2^2 + \omega_3^2)$$

Donde  $I_r$ , es la inercia de las ruedas respecto a su eje de giro, y  $I$  es la inercia total del robot respecto al eje z. Se calcula  $I$  a continuación.

$$I = I_c + 3 * (I_m + M_r * L^2)$$

Donde  $I_c$  es la inercia del chasis del robot respecto al eje z,  $I_m$  es la inercia de las ruedas respecto al eje z.

Se reescribe la energía cinética total del sistema como:

$$K = \frac{1}{2}M * (v_x^2 + v_y^2) + \frac{1}{2} * I * \omega^2 + \frac{1}{2} * I_r * (\omega_1^2 + \omega_2^2 + \omega_3^2)$$

Se expresan  $v_x$ ,  $v_y$  y  $\omega$  en función de las velocidades angulares de las ruedas, las cuales fueron obtenidas en la cinemática directa.

$$K = \frac{1}{2}M * \left( \left[ \frac{\sqrt{3}\pi d}{3} \{ \omega_3 - \omega_2 \} \right]^2 + \left[ \frac{\pi d}{3} \{ \omega_2 + \omega_3 - 2\omega_1 \} \right]^2 \right) + \frac{1}{2} * I * \left[ \frac{\pi d}{3L} \{ \omega_1 + \omega_2 + \omega_3 \} \right]^2 + \frac{1}{2} * I_r * (\omega_1^2 + \omega_2^2 + \omega_3^2)$$

Derivando para según la ecuación del torque se tiene que  $\frac{\delta \mathcal{L}}{\delta q_n}$  es igual a cero, ya que ningún término depende de las posiciones angulares de la rueda. Se derivan los demás términos y se simplifica.

$$\frac{\delta \mathcal{L}}{\delta q_n} = 0$$

$$\frac{\delta \mathcal{L}}{\delta \omega_1} = \frac{1}{2}M * \frac{\pi d}{3} * 2(\omega_2 + \omega_3 - 2\omega_1) * (-2) + \frac{1}{2}I * \frac{\pi d}{3L} * 2(\omega_1 + \omega_2 + \omega_3) + \frac{1}{2}I_r * 2 * \omega_1$$

$$\frac{\delta \mathcal{L}}{\delta \omega_1} = -2 * M * \frac{\pi d}{3}(\omega_2 + \omega_3 - 2\omega_1) + I * \frac{\pi d}{3L}(\omega_1 + \omega_2 + \omega_3) + I_r * \omega_1$$

$$\frac{d}{dt} \frac{\delta \mathcal{L}}{\delta \omega_1} = -2 * M * \frac{\pi d}{3} (\dot{\omega}_2 + \dot{\omega}_3 - 2\dot{\omega}_1) + I * \frac{\pi d}{3L} (\dot{\omega}_1 + \dot{\omega}_2 + \dot{\omega}_3) + I_r * \dot{\omega}_1$$

$$\frac{d}{dt} \frac{\delta \mathcal{L}}{\delta \omega_1} = \left( 4 * M * \frac{\pi d}{3} + I * \frac{\pi d}{3L} + I_r \right) * \dot{\omega}_1 + \left( I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3} \right) * \dot{\omega}_2 + \left( I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3} \right) * \dot{\omega}_3$$

$$\frac{\delta \mathcal{L}}{\delta \omega_2} = \frac{1}{2} M * \left( -2 * \frac{\sqrt{3}\pi d}{3} \{\omega_3 - \omega_2\} + 2 * \frac{\pi d}{3} \{\omega_2 + \omega_3 - 2\omega_1\} \right) + \frac{1}{2} I * \frac{\pi d}{3L} * 2(\omega_1 + \omega_2 + \omega_3) + \frac{1}{2} I_r * 2 * \omega_2$$

$$\frac{\delta \mathcal{L}}{\delta \omega_2} = M * \left( -\frac{\sqrt{3}\pi d}{3} \{\omega_3 - \omega_2\} + \frac{\pi d}{3} \{\omega_2 + \omega_3 - 2\omega_1\} \right) + I * \frac{\pi d}{3L} (\omega_1 + \omega_2 + \omega_3) + I_r * \omega_2$$

$$\frac{d}{dt} \frac{\delta \mathcal{L}}{\delta \omega_2} = M * \left( -\frac{\sqrt{3}\pi d}{3} \{\dot{\omega}_3 - \dot{\omega}_2\} + \frac{\pi d}{3} \{\dot{\omega}_2 + \dot{\omega}_3 - 2\dot{\omega}_1\} \right) + I * \frac{\pi d}{3L} (\dot{\omega}_1 + \dot{\omega}_2 + \dot{\omega}_3) + I_r * \dot{\omega}_2$$

$$\frac{d}{dt} \frac{\delta \mathcal{L}}{\delta \omega_2} = \left( I * \frac{\pi d}{3L} - 2M * \frac{\pi d}{3} \right) * \dot{\omega}_1 + \left( M * \frac{\pi d}{3} + M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r \right) * \dot{\omega}_2 + \left( M * \frac{\pi d}{3} - M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} \right) * \dot{\omega}_3$$

$$\frac{\delta \mathcal{L}}{\delta \omega_3} = \frac{1}{2} M * \left( 2 * \frac{\sqrt{3}\pi d}{3} \{\omega_3 - \omega_2\} + 2 * \frac{\pi d}{3} \{\omega_2 + \omega_3 - 2\omega_1\} \right) + \frac{1}{2} I * \frac{\pi d}{3L} * 2(\omega_1 + \omega_2 + \omega_3) + \frac{1}{2} I_r * 2 * \omega_3$$

$$\frac{\delta \mathcal{L}}{\delta \omega_3} = M * \left( \frac{\sqrt{3}\pi d}{3} \{\omega_3 - \omega_2\} + \frac{\pi d}{3} \{\omega_2 + \omega_3 - 2\omega_1\} \right) + I * \frac{\pi d}{3L} (\omega_1 + \omega_2 + \omega_3) + I_r * \omega_3$$

$$\frac{d}{dt} \frac{\delta \mathcal{L}}{\delta \omega_3} = M * \left( \frac{\sqrt{3}\pi d}{3} \{\dot{\omega}_3 - \dot{\omega}_2\} + \frac{\pi d}{3} \{\dot{\omega}_2 + \dot{\omega}_3 - 2\dot{\omega}_1\} \right) + I * \frac{\pi d}{3L} (\dot{\omega}_1 + \dot{\omega}_2 + \dot{\omega}_3) + I_r * \dot{\omega}_3$$

$$\frac{d}{dt} \frac{\delta \mathcal{L}}{\delta \omega_3} = \left( I * \frac{\pi d}{3L} - 2M * \frac{\pi d}{3} \right) * \dot{\omega}_1 + \left( M * \frac{\pi d}{3} - M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} \right) * \dot{\omega}_2 + \left( M * \frac{\pi d}{3} + M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r \right) * \dot{\omega}_3$$

Entonces cada torque queda de la siguiente manera:

$$\tau_1 = \left( 4 * M * \frac{\pi d}{3} + I * \frac{\pi d}{3L} + I_r \right) * \dot{\omega}_1 + \left( I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3} \right) * \dot{\omega}_2 + \left( I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3} \right) * \dot{\omega}_3$$

$$\tau_2 = \left( I * \frac{\pi d}{3L} - 2M * \frac{\pi d}{3} \right) * \dot{\omega}_1 + \left( M * \frac{\pi d}{3} + M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r \right) * \dot{\omega}_2 + \left( M * \frac{\pi d}{3} - M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} \right) * \dot{\omega}_3$$

$$\tau_3 = \left( I * \frac{\pi d}{3L} - 2M * \frac{\pi d}{3} \right) * \dot{\omega}_1 + \left( M * \frac{\pi d}{3} - M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} \right) * \dot{\omega}_2 + \left( M * \frac{\pi d}{3} + M * \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r \right) * \dot{\omega}_3$$

Las expresiones anteriores pueden expresarse de forma matricial como se muestra a continuación:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} \left(4 * M * \frac{\pi d}{3} + I * \frac{\pi d}{3L} + I_r\right) & \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) & \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) \\ \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) & \left(M \frac{\pi d}{3} + M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r\right) & \left(M \frac{\pi d}{3} - M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L}\right) \\ \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) & \left(M \frac{\pi d}{3} - M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L}\right) & \left(M \frac{\pi d}{3} + M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r\right) \end{bmatrix} * \begin{bmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix}$$

O de forma condensada:

$$\tau = M_{3x3} * \dot{\omega}$$

Siendo  $M_{3x3}$  la matriz de masas iniciales, la cual debería invertirse en el caso que se desee obtener la dinámica directa.

## Diseño mecánico del robot

Se presentan en esta sección el diseño mecánico final del robot, así como la selección de piezas y actuadores, junto con las particularidades de su armado, posteriormente se expresan parámetros cinemáticos y dinámicos necesarios para el desarrollo del presente proyecto.

### Diseño de piezas mecánicas

Las piezas del presente robot han sido diseñadas en Solid Edge 2023 Student Edition, dichas piezas posteriormente han sido impresas en su mayoría en PETG, el cual es un material flexible pero resistente, algunas excepciones fueron impresas en PLA que fue el material con el que se realizó el primer prototipo y que cumplían satisfactoriamente lo que se necesitaba de dichas piezas. A continuación, se muestran capturas de las piezas diseñadas en Solid Edge, junto con el ensamblaje final de las mismas.

#### Chasis de motor PAP y eje

El chasis de los motores PAP presenta el siguiente diseño:

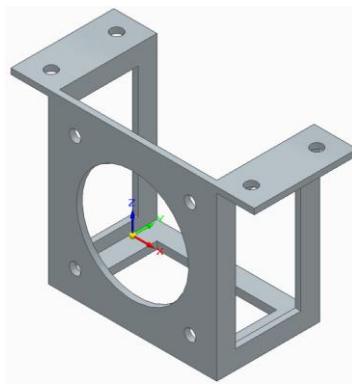
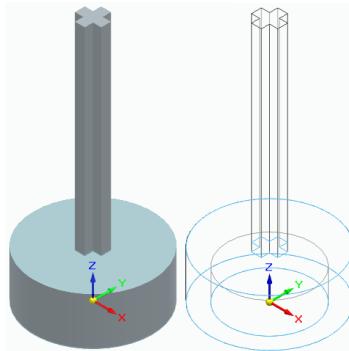


Figura 8: Diseño de chasis de motor en Solid Edge 2023.

Presenta orificios frontales para poder atornillar tornillos de métrica 3 al motor y sujetarlo a la pieza, además de un gran orificio central y tres orificios hacia los costados para mejorar la ventilación,

particularmente estos últimos están diseñados para coincidir con el disipador de calor de los motores, finalmente posee agujeros en la parte superior para poder atornillar dicho soporte a la base inferior del robot.

El eje utilizado tiene el siguiente diseño:



*Figura 9: Diseño de eje en Solid Edge 2023.*

Dicho eje fue diseñado con un agujero en la parte inferior con el objetivo de poder apoyarlo sobre el engranaje que viene montado de fábrica con los motores PAP, la forma en cruz del eje fue diseñada para acoplarse al centro en cruz que tienen las ruedas, se probó previamente un diseño en forma de línea, pero resultó ser considerablemente más frágil y tuvo que descartarse. El ensamble se realiza a presión sobre el engranaje del PAP y reforzado con silicona para aportar una mejor tracción. Luego el eje encaja en la parte interna de la rueda y se le hace un pequeño tapón de silicona en ambos lados para fijarlo en su lugar.

En la siguiente imagen puede observarse el armado de la rueda, nótese que el motor ya se encuentra ensamblado a su chasis:



*Figura 10: Ensamblaje del sistema Rueda-Eje-Motor.*

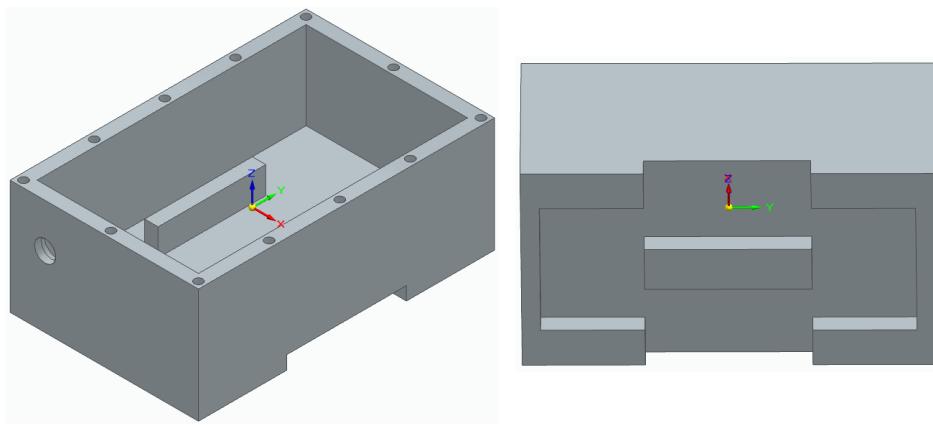
Se muestra el armado final, puede observarse los tornillos del ensamble con la base inferior:



*Figura 11: Detalle de siliconado de los ejes.*

### Soporte de batería

El soporte de batería fue diseñado para poder transportar la batería, además de mejorar la distribución de cargas del robot, consiste en un espacio rectangular con un pequeño soporte que limita el movimiento de la batería, además contiene un orificio donde se apoya una ficha tipo Jack hembra de chasis, el cual se usa para conectar el cargador de batería. En la parte inferior se han colocado ranuras para ruedas esféricas para muebles ligeros, dichas ruedas esféricas soportan hasta 4kg en conjunto. Además, el soporte incluye orificios de tornillos para atornillarse por la parte superior a la base inferior del robot. Se muestra a continuación tanto el diseño de la pieza en perspectiva isométrica y visto desde abajo:

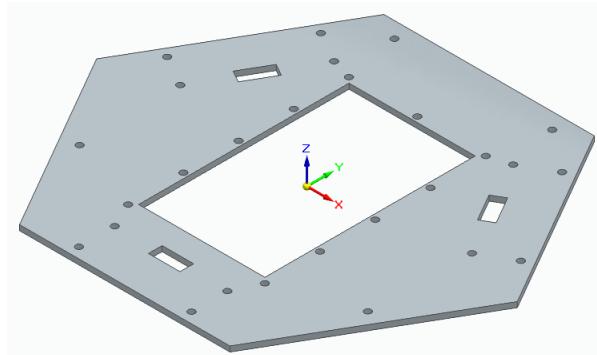


*Figura 12: Diseño de soporte de baterías en Solid Edge 2023.*

Se muestran las fotografías en la siguiente sección, ya que no se tomaron fotografías con esta parte desacoplada. Cabe aclarar que la batería sigue presentando un movimiento relativo a su soporte, por lo cual se utilizó una pequeña pieza de cartón para fijarla en su sitio.

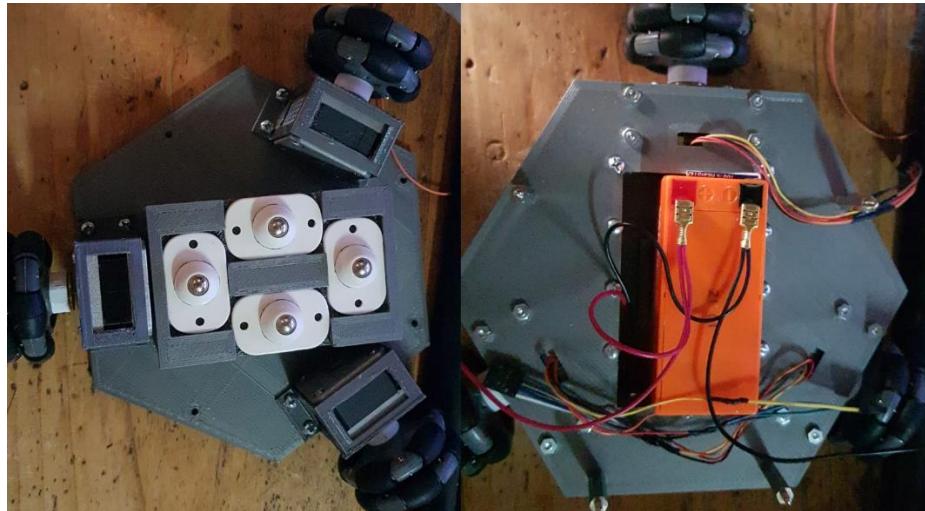
### Base inferior

La base inferior es una pieza con forma hexagonal, pensada para tener en tres de sus caras los motores y en las otras tres unos separadores para conectarla mecánicamente con la base intermedia. Posee además unas ranuras pasacable que coinciden con el lugar en donde se encuentran los cables de los PAP y una ranura grande para poder poner la batería en el soporte. El diseño puede observarse a continuación:



*Figura 13: Diseño de base inferior en Solid Edge 2023.*

Pueden observarse a continuación fotos del montaje de la pieza, siendo la primera desde abajo para dejar en evidencia las ruedas esféricas, y la segunda en vista cenital para observar el montaje de la batería, dicho montaje incluye el cableado.



*Figura 14: Ensamblaje de base inferior.*

### Base intermedia

Esta base fue creada con la misma geometría base que la anterior, solo que en lugar de los motores se distribuye en sus caras correspondientes los PCB con los drivers para PAP, en la cara frontal se colocó un espacio para PCB extra para las borneras y terminales comunes, mientras que en el centro se ubicó el módulo de relé, dicho módulo debido a su forma se encuentra sujetado por dos tornillos en disposición descendente atornillados a la placa, y dos tornillos de forma ascendente con un tapón de silicona para mantener el módulo en su lugar, los PCB fueron pegados con silicona a la pieza. Además, la pieza posee pasacables tanto para los cables de los motores como para los de la batería. Puede observarse a continuación el diseño de la pieza:

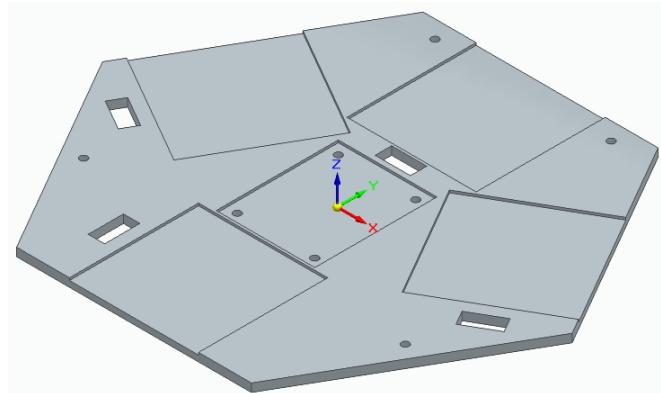


Figura 15: Diseño de base intermedia en Solid Edge 2023.

Puede observarse a continuación el montaje con las conexiones eléctricas de la pieza:

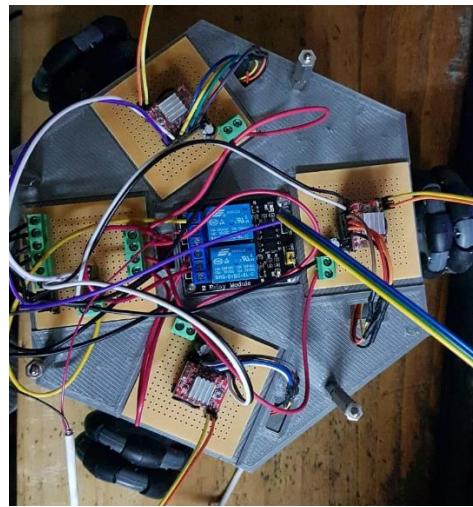


Figura 16: Ensamblaje de base intermedia.

### Base superior

La base superior fue diseñada con geometría base similar a las piezas anteriores, posee espacios de tornillo para el montaje de la Raspberry Pi, el soporte del servo y el Arduino que debido a consideraciones similares al módulo de relé debió fijarse con los tornillos desde abajo con tapones de silicona. Finalmente, la pieza presenta dos grandes pasacables para permitir el paso del cable USB desde el piso intermedio, se diseño de un tamaño grande por comodidad, debido a que el espacio era menos ajustado que en las otras piezas. Se muestra a continuación el diseño de la pieza:

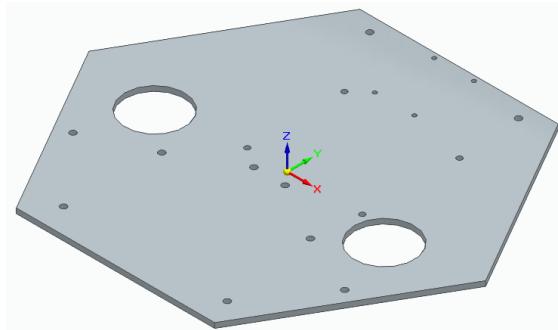


Figura 17: Diseño de base superior en Solid Edge 2023.

Se muestra a continuación el ensamblaje de la pieza, el cual consiste en el armado final del robot:



Figura 18: Ensamblaje de base superior.

### Sopporte de servomotor y cámara

Estas piezas se presentan juntas debido a que pertenecen al mismo montaje, el soporte de servomotor fue diseñado de la siguiente manera:

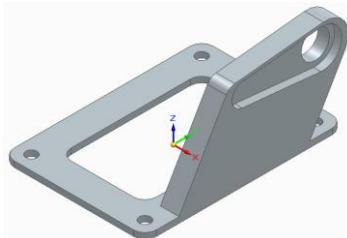
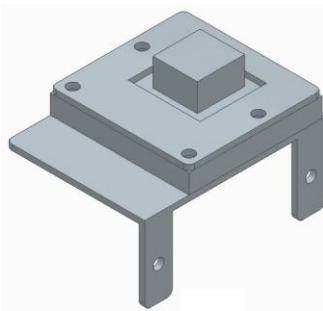


Figura 19: Diseño de soporte de servomotor en Solid Edge 2023.

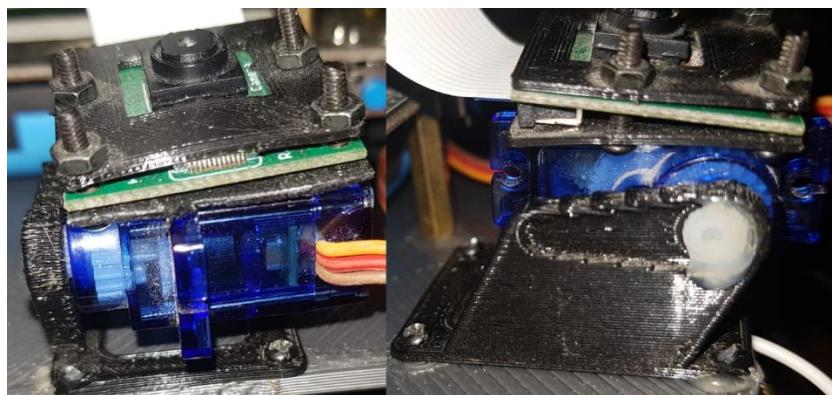
Posee un orificio para pasar el eje del servomotor y una ranura para colocar una de sus paletas, dicha ranura no fue utilizada debido a que el servomotor se fijaba correctamente con el agujero circular, además que la terminación se veía frágil, por lo que se optó fijar el servomotor con silicona, la pieza posee además cuatro orificios para tornillos de métrica 2, solo tres de ellos pudieron ser utilizados, siendo el último de ellos pegado con silicona.

Las piezas de montaje de la cámara se diseñaron de la siguiente manera:



*Figura 20: Diseño de ensamblaje de cámara en Solid Edge 2023.*

Ambas piezas poseen orificios para tornillo de métrica 2, además de una placa vertical para montar sobre el servomotor, dichas placas se rompieron, mientras que la parte plana que sobresale por debajo de la cámara fue removida, finalmente debió ser pegada sobre el servomotor con silicona, la cámara fue montada al revés de lo planeado debido a que no se tuvo en consideración el cable Flex, se optó por montarla de dicha forma e invertir la imagen por software. El resultado final fue el siguiente:



*Figura 21: Detalle de ensamblaje de servomotor y cámara.*

## Parámetros cinemáticos y dinámicos finales

Se presenta a continuación la tabla con los parámetros cinemáticos y dinámicos finales:

Parámetro	Símbolo	Valor
Radio de rueda	$d$	$58mm = 0,058m$
Distancia entre centro geométrico del robot y rueda	$L$	$112mm = 0,112m$
Masa del robot	$M$	$2,1kg$
Inercia del robot respecto al eje Z	$I$	$0,016kg * m^2$
Inercia rotacional de las ruedas con sus motores	$I_r$	$0,0001kg * m^2$

Tabla 3: Parámetros cinemáticos y dinámicos finales.

## Selección de actuadores

Se detalla a continuación los actuadores seleccionados:

**Servomotor Tower Pro Sg90:** es un servomotor de 90 gramos de peso, controlado mediante PWM con un ángulo de trabajo de 180 grados, presenta un torque de 1,8kgf.cm, utilizado como actuador para la orientación vertical de la cámara.

**Motor Paso a Paso 42HS028MF800X-01:** Motor paso a paso de la serie 42HSM, posee un ángulo de giro de 0,9 grados por paso, y con un torque de 10 N.cm, teniendo buena precisión puede considerarse como un buen sensor mientras que no pierda pasos, es utilizado para el movimiento del robot.

## Componentes electrónicos y conexiones eléctricas

En esta sección se detallan todos los componentes electrónicos utilizados, además de las conexiones entre ellos, divididas entre conexiones eléctricas, de potencia y datos, finalmente incluyendo planos del sistema electrónico dividido en subsistemas, así como el plano del circuito completo.

## Componentes electrónicos

Los componentes electrónicos utilizados en el presente proyecto son los siguientes:

**Raspberry Pi 4B de 4 GB de RAM:** Los ordenadores de la familia Raspberry Pi se caracterizan por su pequeño tamaño, similar a un Arduino Mega, su bajo consumo y su potencia informática, en este caso particular contiene una tarjeta SSD donde corre el sistema operativo Raspbian distribución Debian 12 "Bookworm", se incluye con la función de correr el servidor desde donde se envían y reciben datos desde y hacia la interfaz de usuario, y la inteligencia artificial tomando datos de la cámara, enviando además las consignas mediante puerto serie al Arduino Mega 2560 y recibir los datos de la navegación para realizar la integración con el servidor, el cual mediante puerto Wi-Fi configurado como punto de acceso puede enviar información a una terminal de computadora.

**Cámara Raspberry Pi V2.1:** Esta cámara diseñada para conectarse con una Raspberry Pi tiene una resolución de 1080p y hasta 60 frames por segundo en una resolución de 720p. Se conecta al puerto de cámara de la Raspberry Pi mediante un cable de tipo flex.

**Arduino Mega 2560 R3:** Es una placa microcontroladora conocida generalmente por su facilidad de uso y programación, utilizado como coordinador de ejes para convertir las consignas de velocidad de los actuadores en movimiento de los mismos, así como tomar los datos medidos del sensor MPU6050 y combinarlos con el movimiento de los actuadores para obtener la posición del robot. Se conecta a la Raspberry a través de un cable de impresora corto.

**MPU6050:** Placa integrada que contiene 3 acelerómetros, 3 giróscopos y termómetro, se utiliza tan cual se describe en el apartado de “Descripción de sensores”.

**Servomotor Tower Pro Sg90:** Un servomotor pequeño con un torque de 1,3 kgf/cm en reposo. Se utiliza como se especifica en la sección de “Selección de actuadores”.

**Módulo Relé HL-52S:** Módulo de relé de dos canales, dichos relés conmutan a 5V y 140mA, son utilizados para conmutar en el riel normalmente abierto para conectar los drivers Pololu A4988, los motores PAP y el servomotor solo cuando el robot esté encendido, para evitar pérdida de energía en las baterías.

**Driver Pololu A4988:** Driver de motor paso a paso del tipo pulso-dirección con posibilidad de micropaso, realiza un PWM entre las fases del motor para aumentar la fuerza de conmutación sin aumentar el calentamiento del sensor, requiriendo para su correcto funcionamiento una entrada de potencia de 8V a 35V y una resistencia de 100 microfaradios entre las terminales de potencia y tierra, además cuenta con un tornillo regulador de corriente, sobre el cual se mide voltaje respecto a tierra mediante la formula  $V_{ref} = I_{max} * 8 * R$  siendo esta última la resistencia del regulador, que es en este caso 0,1 Ohms, se utiliza con una tensión de regulador de 0,34V en modo paso completo.

**Motor Paso a Paso 42HS028MF800X-01:** Motor paso a paso con una corriente máxima de 0,425A y una resolución de 0,9 grados por paso con un torque de 10N/cm, se utiliza tanto como actuador y como sensor ya que se considera que no pierde pasos gracias al driver Pololu y la alimentación de 12V, las particularidades de su uso pueden observarse tanto en las secciones de “Selección de actuadores” como en la sección de “Descripción de los sensores”.

**Batería de 12V Vapex VT1213:** Batería de plomo y ácido de 12V y 1,3 Amperes hora, utilizada como alimentación de los motores PAP a través del driver Pololu.

**UPS MakerHawk EP-0136:** UPS con portabaterías y Coulómetro I2C para montar con una Raspberry Pi, utiliza dos baterías de litio 18650 de 3,7V, además tiene dos salidas USB 3.0 que pueden utilizarse para dar energía a otros módulos. Se utiliza montado para dar energía a la Raspberry y se utiliza una de las salidas USB para dar energía al servomotor y a los pololus.

## Diagramas de conexiones

Se detallan en esta sección las conexiones pertinentes al presente proyecto divididas en subsistemas, así como el diagrama final de conexiones realizado con el software de uso libre Cirkit Designer.

### Consideraciones eléctricas

En esta sección se detallan las corrientes utilizadas por cada componente de 5V para poder separar los circuitos y evitar sobrecarga. Se considera que se utilizan puertos USB3.0 con una capacidad de corriente de 900mA, se considera además una capacidad de corriente total de 150mA en el Arduino

Componente	Cantidad	Consumo por componente/Total
MPU6050	1	3,9mA
Relé de dos canales Arduino	1	140mA
Driver Pololu A4988	3	10/30mA
Servo Tower Pro SG90	1	250mA

*Tabla 4: Consumo de energía por componente de 5V.*

Se considera conectar el MPU6050 y el relé al Arduino, mientras que los drivers Pololu y el Servomotor se conectan a la salida USB3.0 del UPS, de esta forma no se sobrecarga el Arduino y además es mucho más sencillo realizar el corte automatizado mediante el relé del sistema de 5V. Se elige la UPS sobre la Raspberry en este caso para proteger los puertos USB de la misma frente a cualquier problema que pudiera ocurrir.

### Diagrama de conexiones generales

Todo el sistema eléctrico se muestra en la siguiente imagen:

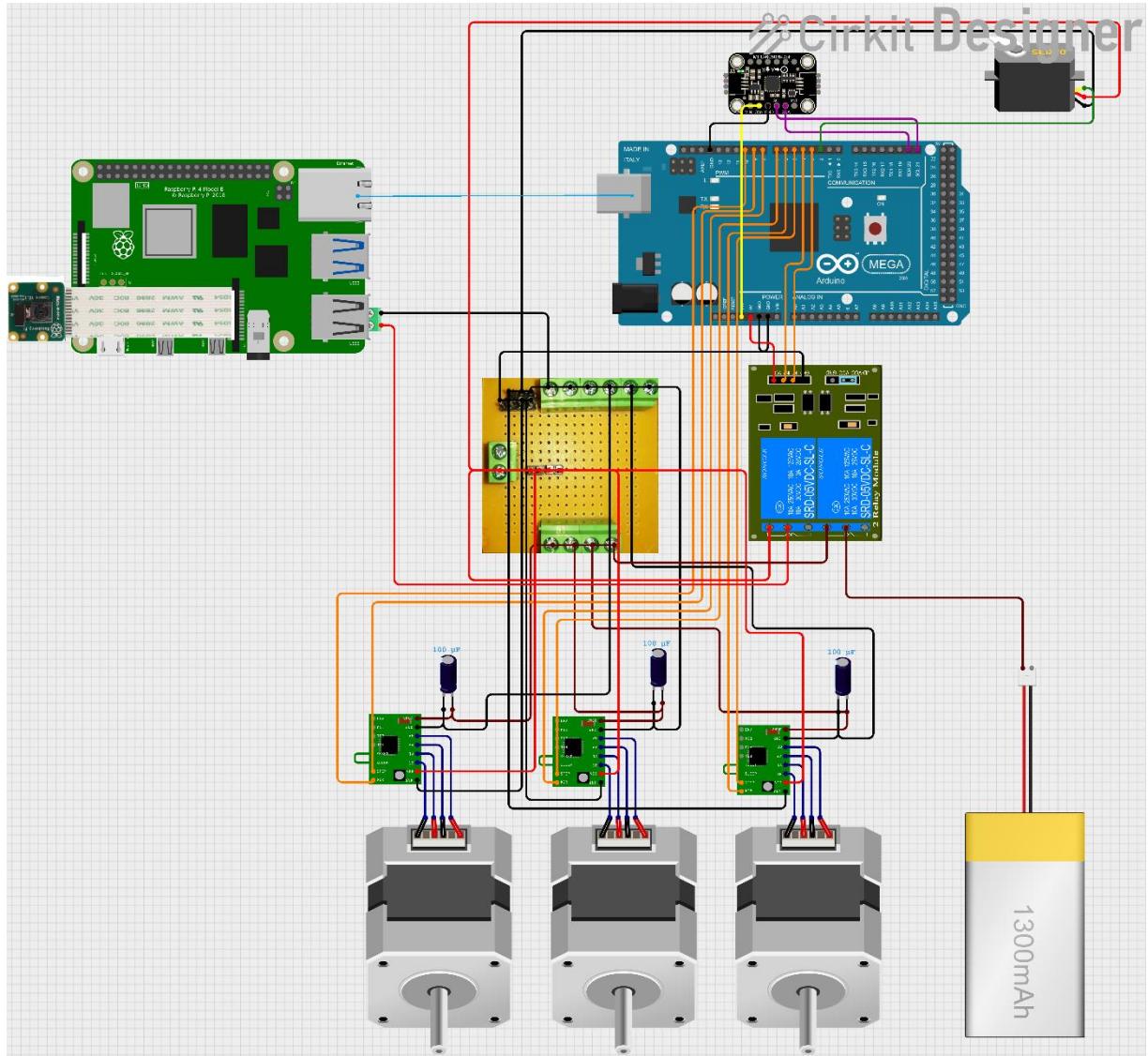


Figura 22: Diagrama de conexiones generales en Cirkit Designer.

En la anterior imagen nótese el agregado de un PCB con borneras y pinos, dicho PCB se ha utilizado para crear buses de energía comunes a todos los componentes que utilicen dicha energía, se diferencia en cable rojo brillante los cables de 5V y en rojo oscuro los de 12V, además las conexiones de tierra son en color negro, las conexiones azules son de potencia de los motores paso a paso, las naranjas son de control digital a través de GPIO, el servomotor tiene una conexión verde de PWM y el MPU tiene conexiones I2C Púrpuras y una conexión de alimentación de 3,3V amarilla. En celeste puede observarse también la representación del cable de impresora conectada al puerto USB de la Raspberry.

No se muestra la UPS que se considera conectada debajo de la Raspberry, puede observarse una pequeña bornera la cual se utilizó para poder graficar las conexiones que salen del puerto USB3.0 de la misma.

En lo referente a los drivers pololu se tiene en cuenta que el regulador de corriente requiere un ajuste previo, según la datasheet, y teniendo en cuenta el uso de paso completo, se tiene que la regulación es:

$$V_{ref} = I_M * 8 * \frac{R_s}{0.7} = 0.048V$$

Donde  $I_M = 0.425$  es la corriente máxima del motor, y  $R_s = 0.01\Omega$  es la resistencia de sensado.

Se opta por usar  $V_{ref} = 0.046V$ .

### Subsistema Raspberry-Cámara

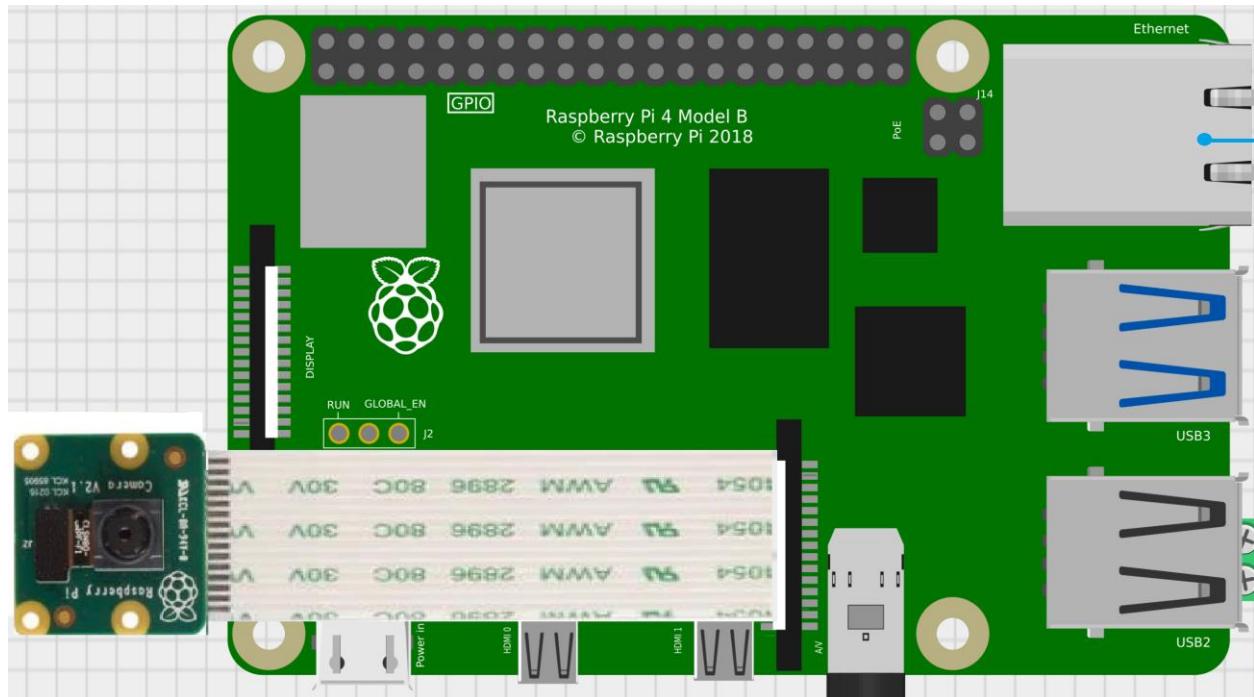


Figura 23: Diagrama de conexión de cámara en Cirkit Designer.

Nótese que la cámara se encuentra conectada a la Raspberry mediante un cable flex a la terminal de cámara.

### Subsistema de energía

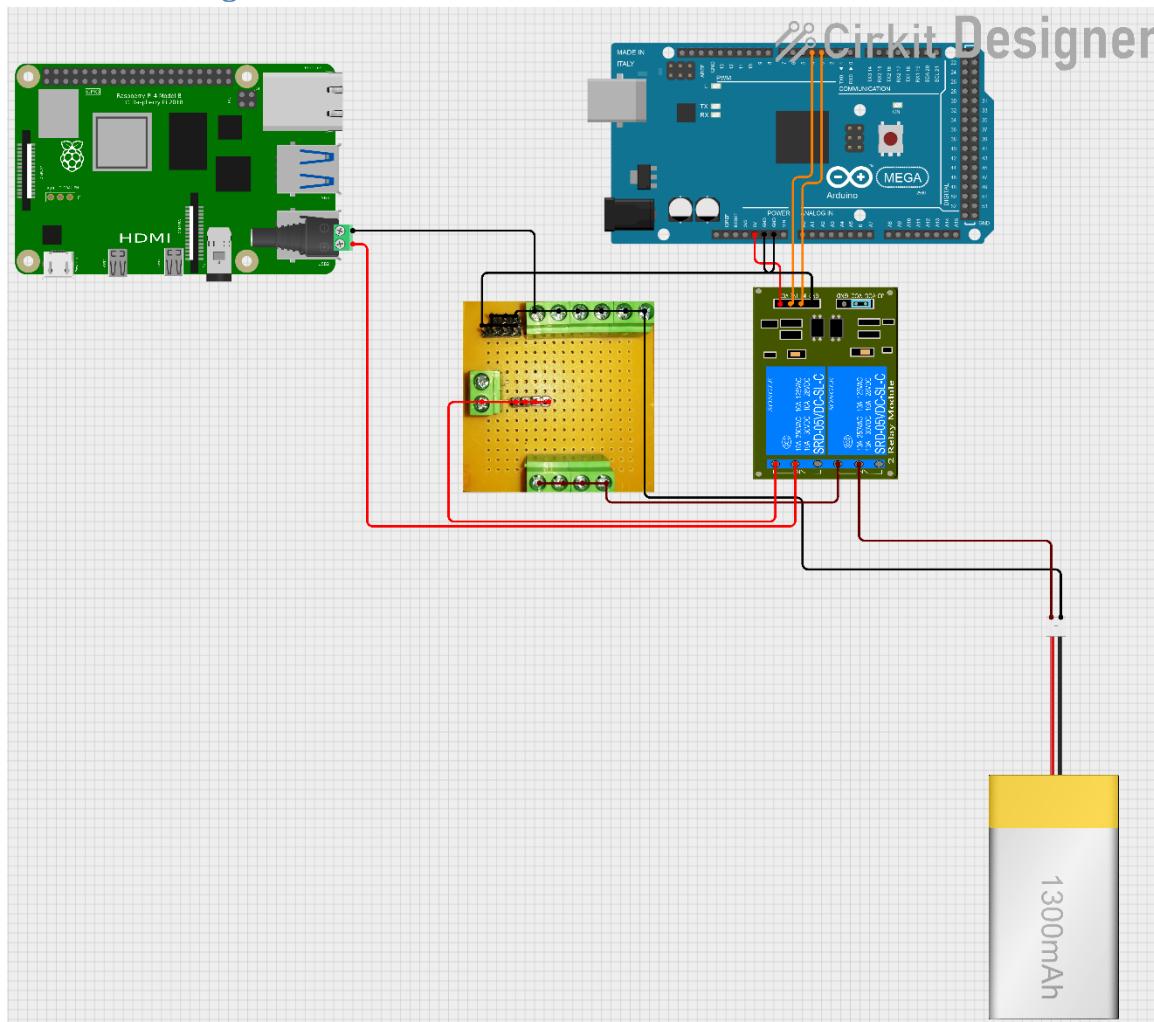
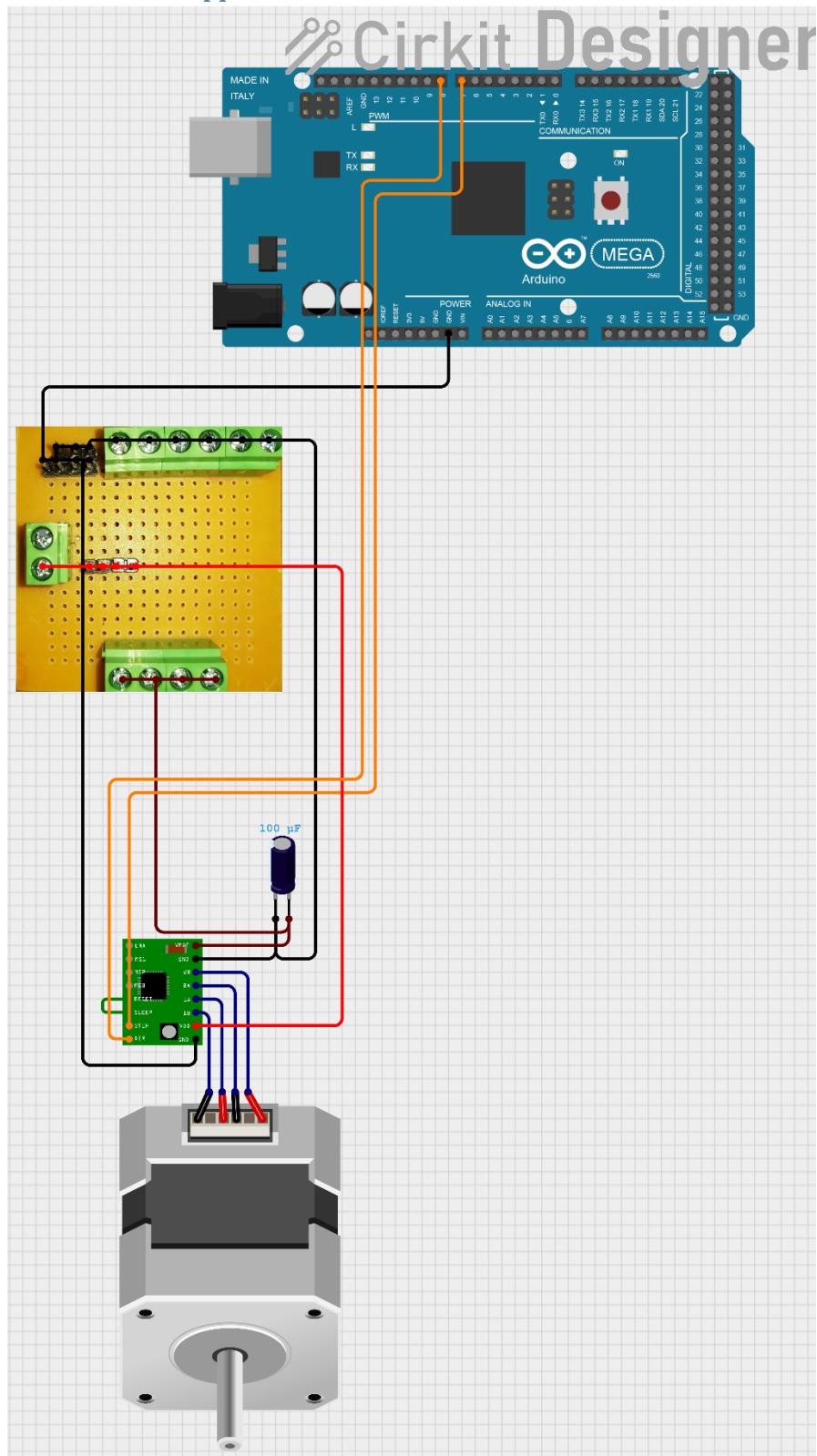


Figura 24: Diagrama de conexiones de energía en Cirkit Designer.

**Subsistema Pololu A4988-Stepper***Figura 25: Diagrama de conexiones de Pololu y Stepper en Cirkit Designer.*

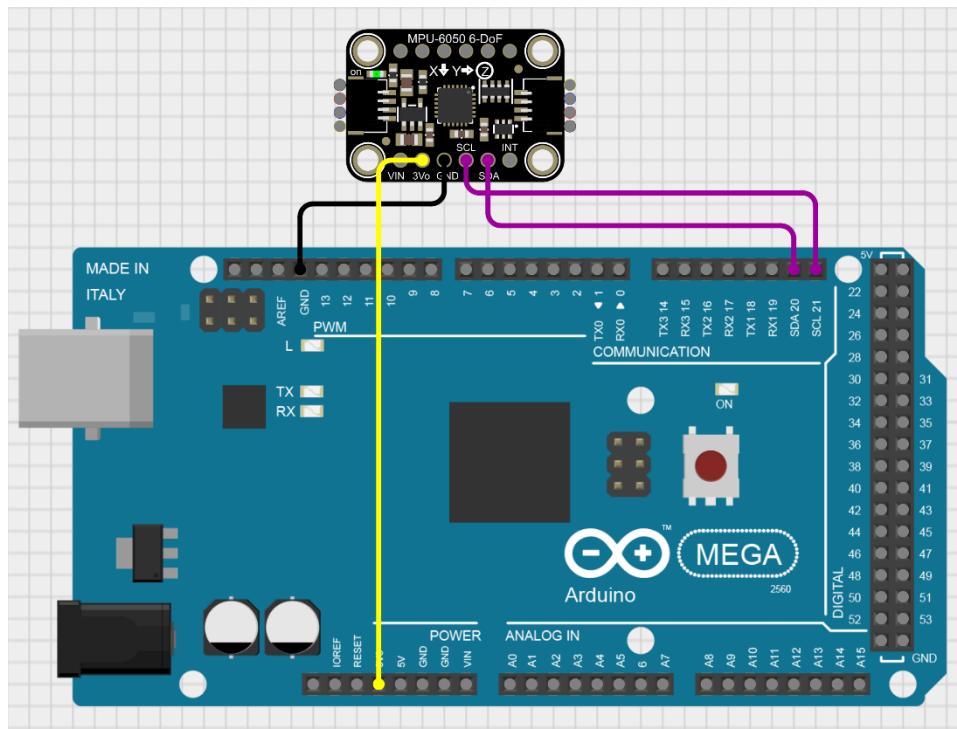
**Subsistema MPU6050**

Figura 26: Diagrama de conexiones de MPU6050 en Circuito Designer.

### Subsistema Servomotor

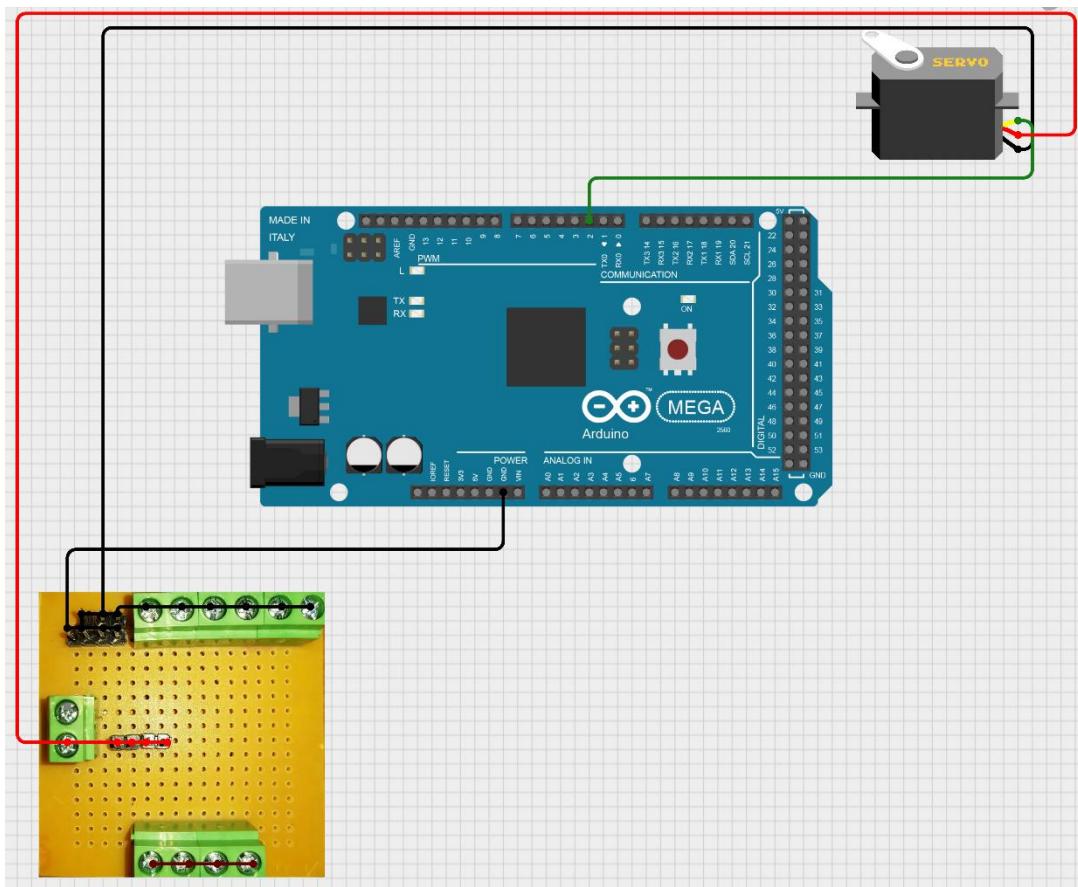


Figura 27: Diagrama de conexiones de servomotor en Circit Designer.

### Sistema de navegación asistida

En esta sección se presenta el sistema de navegación del robot, que consiste en un filtro de Kalman como observador de estados, el cual utilizará como entrada el sensor MPU6050 y tres motores paso a paso, los cuales se utilizarán como actuador y tendrán además la función de encoders. Se aprovechará la ventaja del filtro de Kalman para eliminar el error por integración de los sensores y poder indicar la ubicación del robot con mayor precisión.

### Consideraciones previas

Antes de comenzar es prudente tomar en cuenta ciertas consideraciones, las cuales son:

- Se considera que los motores paso a paso utilizados como encoders no pierden paso al moverse, por lo que marcan la posición de forma confiable, pero con cierto ruido en la medición.
- Se considera que el movimiento del robot se realiza en el plano horizontal X-Y para los cálculos e implementación.
- Se corrige la medición de los acelerómetros eliminando la fuerza de gravedad como se explica más adelante.

## Descripción de los sensores

Para la realimentación de al menos dos variables en el filtro de Kalman es necesario describir los sensores utilizados. En este caso se presenta un sensor y un actuador que cumplen dichas funciones:

**MPU6050:** placa integrada que contiene 3 acelerómetros y 3 giróscopos con comunicación I2C y capacidad de agregar un magnetómetro de 3 ejes para utilizar referencia respecto a los polos magnéticos terrestres que no se incluye en el presente proyecto. Dichos sensores se modelarán con ganancias igual a 1 (tanto para acelerómetros y giróscopo) y ruido gaussiano de media 0 para ambos tipos de sensor, varianza  $2.5456 * 10^{-3} \text{ m/s}^2$  para los acelerómetros y  $3.2527 * 10^{-4} \text{ }^\circ/\text{s}$  para el giróscopo.

**Motor paso a paso:** Motor de corriente continua sin escobillas con paso de  $0,9^\circ$ , para el filtro de Kalman se lo considera como un encoder de 400ppr sin considerar las posibles pérdidas de pasos, permitiendo conocer posición angular relativa y sentido de giro. Se lo modela con ruido gaussiano de media 0 y varianza  $7.7170 * 10^{-6} \text{ rad}$ . Permite conocer posición angular relativa y sentido de giro.

## Descripción de la planta en espacio de estados

Para poder implementar un filtro de Kalman es necesario presentar la planta en espacio de estados, dicha planta será el robot, representado de forma cinemática por simplicidad y para evitar errores que pueden aportar la dinámica, como no tener pesos o inercias exactas. Para poder utilizar la información de los motores en la planta se utiliza la cinemática directa. A continuación, se presenta las ecuaciones de estado utilizadas, considerando el sistema como un sistema discreto y continuo en el tiempo:

$$\begin{bmatrix} X \\ Y \\ \psi \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ \psi \end{bmatrix}_{t-1} + \begin{bmatrix} dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \end{bmatrix} * \begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix}_t$$

$$X_t = A * X_{t-1} + B * u$$

Donde los vectores  $X_t = \begin{bmatrix} X \\ Y \\ \psi \end{bmatrix}_t$  y  $X_{t-1} = \begin{bmatrix} X \\ Y \\ \psi \end{bmatrix}_{t-1}$  representan el vector de estados en el tiempo actual y en el paso anterior.

La matriz  $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$  representa la matriz de estados.

La matriz  $B = \begin{bmatrix} dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \end{bmatrix}$  representa la matriz de entradas. Donde  $dt$  es el paso de tiempo utilizado, el cual por conveniencia se elige a  $1\text{kHz}$ .

El vector  $u = \begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix}_t$  representa el vector de entradas.

La salida puede expresarse de la siguiente manera:

$$y = C * X_t + D * u$$

Donde la matriz de salida corresponde a  $C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$  y la matriz de transferencia directa es de  $D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ .

Se define el estado inicial del sistema como  $X_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ .

## Descripción del filtro de Kalman

El filtro de Kalman es un estimador de estados, el cual considera la existencia de ruido, tanto en el proceso como en el sensado, permitiendo estimar variables de estado ocultas, corregir las variables conocidas en base al ruido de múltiples sensores y permitiendo métodos de control por realimentación de estados. Para su funcionamiento debe describirse la planta en el espacio de estados, así como el ruido del proceso y de los sensores en dicho espacio.

El algoritmo del filtro de Kalman consiste en realizar una predicción, considerando el ruido del proceso producido por tener entradas no ideales en la planta, como en este caso las entradas de velocidades. Para posteriormente comparar dicha predicción con los datos de los sensores y realizar la corrección basada en la entrada de los sensores, teniendo en cuenta el ruido de los mismos.

Se define la matriz de ruido de proceso  $Q$  como:

$$Q = \begin{bmatrix} \frac{1}{2} * (\sigma_{Acel}^2)^2 & 0 & 0 \\ 0 & \frac{1}{2} * (\sigma_{Acel}^2)^2 & 0 \\ 0 & 0 & \sigma_{Giro}^2 \end{bmatrix}$$

Donde  $\sigma_{Acel}^2$  y  $\sigma_{Giro}^2$  son las varianzas del ruido de los acelerómetros y el giróscopo respectivamente. Nótese que la varianza de los acelerómetros se ha integrado, esto es debido a que alimentamos velocidad en la entrada del sistema, por lo que tanto la señal como el ruido deben integrarse.

Se obtiene la matriz de ruido de la medición  $R$  a partir de la cinemática directa como:

$$R = diag \left( \begin{bmatrix} 0 & -\frac{\sqrt{3}\pi d}{3} & \frac{\sqrt{3}\pi d}{3} \\ -\frac{2\pi d}{3} & \frac{\pi d}{3} & \frac{\pi d}{3} \\ \frac{\pi d}{3L} & \frac{\pi d}{3L} & \frac{\pi d}{3L} \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \sigma_{Enc}^2 \right)$$

$$R = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{\pi d}{L} * \sigma_{Enc}^2 \end{bmatrix}$$

Donde  $\sigma_{Enc}^2$  es la varianza del ruido del motor que hemos utilizado como encoder.

Se define la matriz de covarianza del proceso en su estado inicial, en este caso se ha optado por utilizar

$$P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \text{ Dicha matriz representa la indeterminación debido al ruido de las variables de estado.}$$

Se explica a continuación el algoritmo utilizado:

- Se estima el estado del sistema en el tiempo actual a partir del estado del sistema en el tiempo anterior.

$$X_p = A * X_{t-1} + B * u$$

- Para el caso discreto se multiplica lo anterior por  $1 + dt$ .

$$X_t = X_{t-1} + dt * X_p$$

- Se calcula la matriz de covarianza del proceso en el tiempo actual.

$$P_t = A * P_{t-1} * A^T + Q$$

- Se calcula la ganancia de Kalman en el tiempo actual.

$$K_t = P_t * C^T * (C * P * C^T + R)^{-1}$$

- Se corrige el estado del sistema en el tiempo actual. Considerando la salida del sistema  $y_t$  obtenida por los sensores.

$$X_{tc} = X_t + K_t * (y_t - C * X_t)$$

- Se corrige la matriz de covarianza del proceso en el tiempo actual.

$$P_{tc} = (I - K_t * C) * P_t$$

Siendo  $I$  la matriz identidad del tamaño del vector de estados.

## Ajuste según la gravedad terrestre

Se considera que el sensor MPU6050 detecta la aceleración de la gravedad además de la aceleración producida por el movimiento, por lo tanto, esta debe retirarse antes de poder trabajar con los datos del sensor. Dicha aceleración se calcula de la siguiente forma:

$$a_g = R * G$$

$$[a_g] = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

Los ángulos  $\psi$ ,  $\phi$  y  $\theta$  son respectivamente los ángulos de guiñada, cabeceo y alabeo respectivamente. Se tiene en cuenta la aceleración de la gravedad  $g = 9,81m/s^2$ .

Resolviendo se obtiene:

$$\begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} -g * (\sin(\psi) * \sin(\theta) + \cos(\psi) * \cos(\theta) * \sin(\phi)) \\ -g * (\cos(\theta) * \sin(\phi) * \sin(\psi) - \cos(\psi) * \sin(\theta)) \\ -g * \cos(\phi) * \cos(\theta) \end{bmatrix}$$

Se decide realizar un cálculo de la rotación inicial cuando el robot no se encuentra en movimiento, es decir al momento de encenderse, y posteriormente se utilizará integración de la señal de los giróscopos en cuanto al cabeceo y el alabeo, con respecto a la guiñada se tomará el valor dado por el filtro de Kalman en el paso anterior y se integrará el último valor como aproximación. Como se definió previamente:

$$\psi_0 = 0$$

Por lo que para  $t = 0$

$$\begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix}_{t=0} = \begin{bmatrix} -g * (\cos(\theta) * \sin(\phi)) \\ -g * (-\sin(\theta)) \\ -g * \cos(\phi) * \cos(\theta) \end{bmatrix}$$

Por tanto se puede calcular  $\phi_0$  y  $\theta_0$  despejando de las ecuaciones anteriores:

$$\theta_0 = \arcsen\left(\frac{g_{20}}{g}\right)$$

$$\phi_0 = \arsen\left(-\frac{g_{10}}{g \cos(\theta)}\right)$$

Por cuestiones prácticas y para evitar la mayor parte del ruido se toma el promedio de las primeras 20 mediciones para  $\phi_0$  y  $\theta_0$ . Además, cuando el robot se encuentra en reposo puede decirse que:

$$g = \sqrt{g_1^2 + g_2^2 + g_3^2}$$

Tomando las distintas componentes de  $g$  de los datos del sensor.

Durante las pruebas de implementación cabe destacar que los giróscopos poseen de forma estable un valor de deriva (o drift), el cual es constante, por lo que debe eliminarse antes de poder realizar la corrección.

## Resultados en Matlab

Antes de la implementación en Arduino se hicieron pruebas en Matlab y Simulink para probar la respuesta en el tiempo del filtro de Kalman y del filtro de gravedad. Dichos filtros se implementaron de la manera expresada anteriormente, se agregó un control PID controlando la posición, junto con una consigna trapezoidal para probar el sistema en movimiento. El código es similar al que se utilizó después en Arduino y por tanto no se incluye. Los resultados obtenidos son los siguientes:



Figura 28: Prueba de Filtro de Kalman en Simulink, eje X.

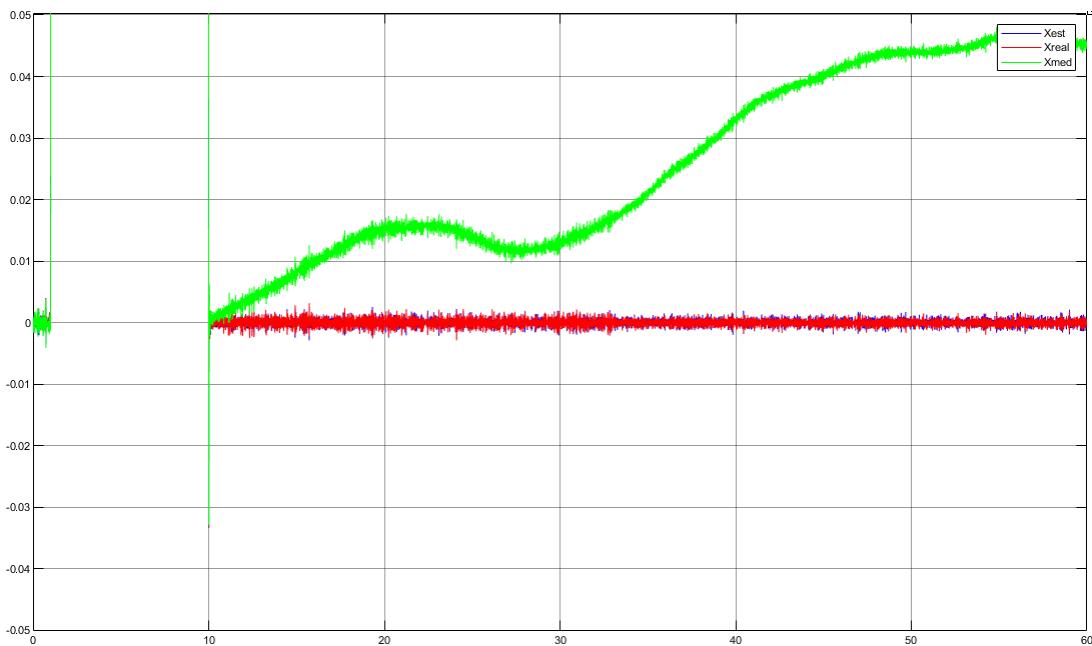
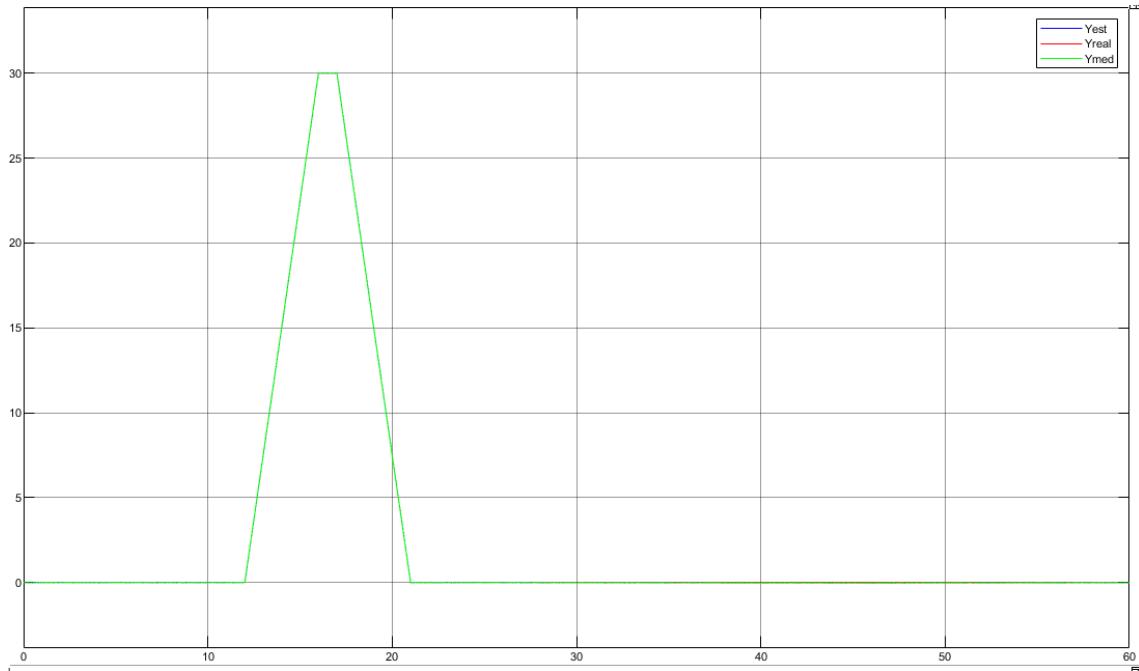
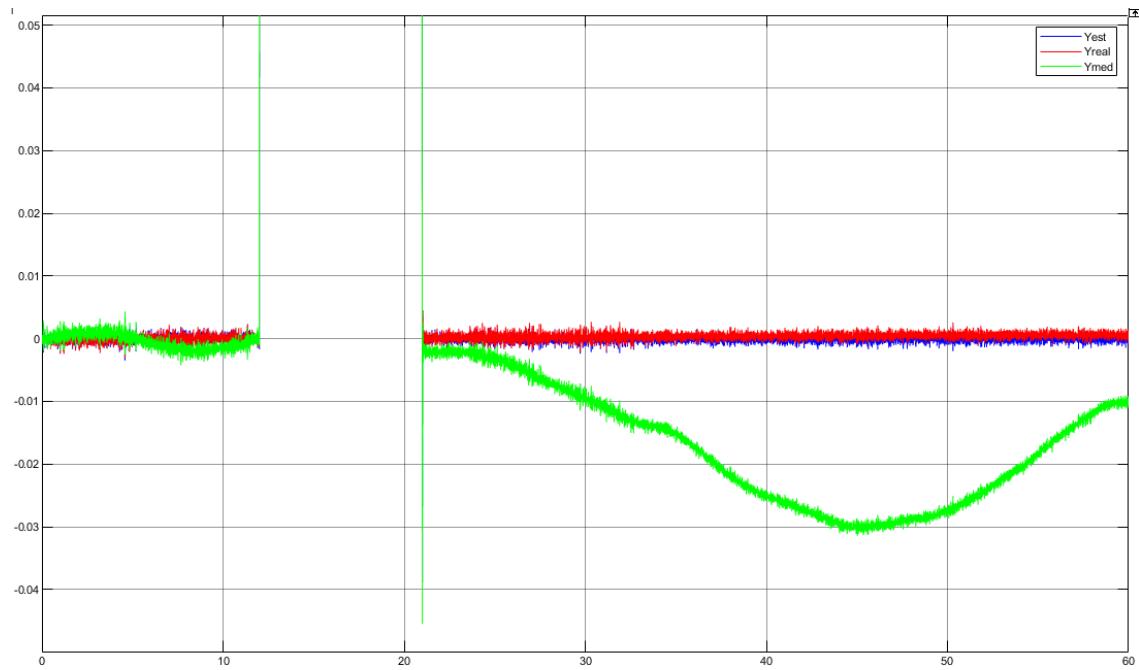


Figura 29: Prueba de Filtro de Kalman en Simulink, eje X, detalle.

En la primera parte de la imagen puede verse la respuesta completa en el tiempo, en la segunda parte se observa un acercamiento de la imagen, puede notarse como la posición medida e integrada a partir de los sensores tienden a divergir de la posición real, mientras que la posición medida por el filtro de Kalman se mantiene más próxima a la posición real. Dicha diferencia puede observarse también en el eje y, además de en la rotación como puede verse a continuación:



*Figura 30: Prueba de Filtro de Kalman en Simulink, eje Y.*



*Figura 31: Prueba de Filtro de Kalman en Simulink, eje Y, detalle.*

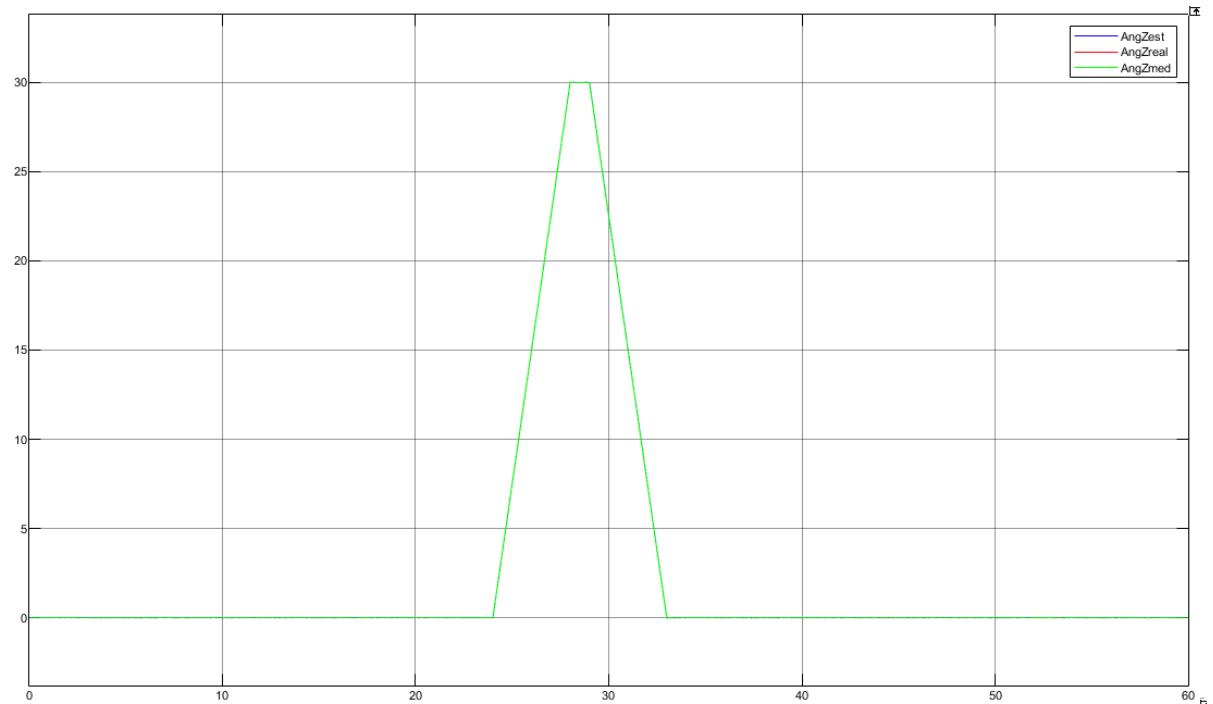


Figura 32: Prueba de Filtro de Kalman en Simulink, rotación.

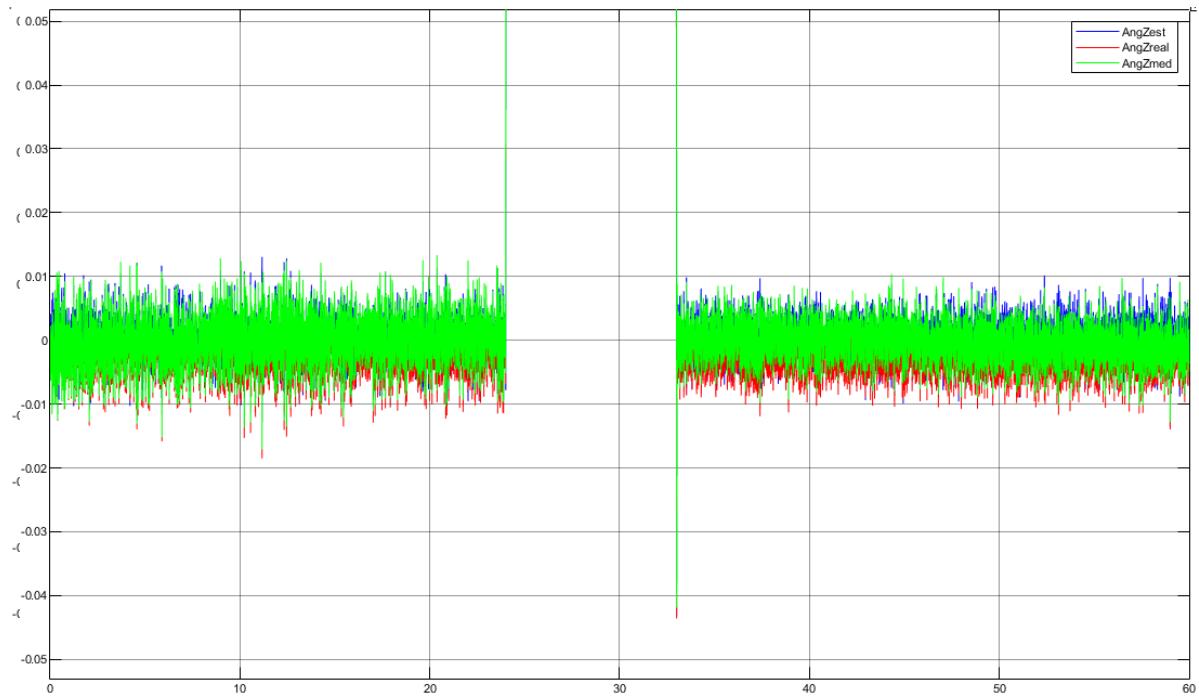


Figura 33: Prueba de Filtro de Kalman en Simulink, rotación, detalle.

En cuanto al filtro de gravedad puede observarse en la siguiente imagen la diferencia entre la gravedad real y estimada en los tres ejes:

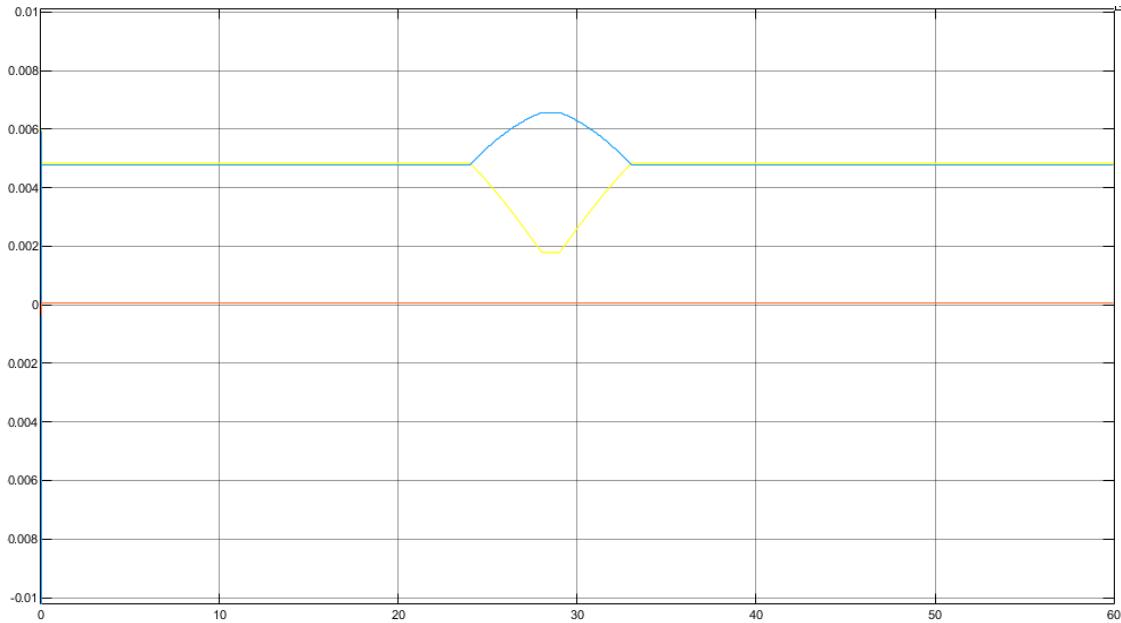


Figura 34: Prueba de Filtro de gravedad en Simulink, ejes X,Y,Z.

Puede observarse una pequeña desviación al momento de girar el robot en la simulación, esto es debido a que se le dio un ángulo inicial para probar el cálculo de ángulo, este a su vez tuvo un pequeño error en la estimación, el cual se observa al girar respecto al eje z. Puede observarse a continuación la rotación respecto al eje x, donde la línea amarilla es la calculada y la azul es la real:

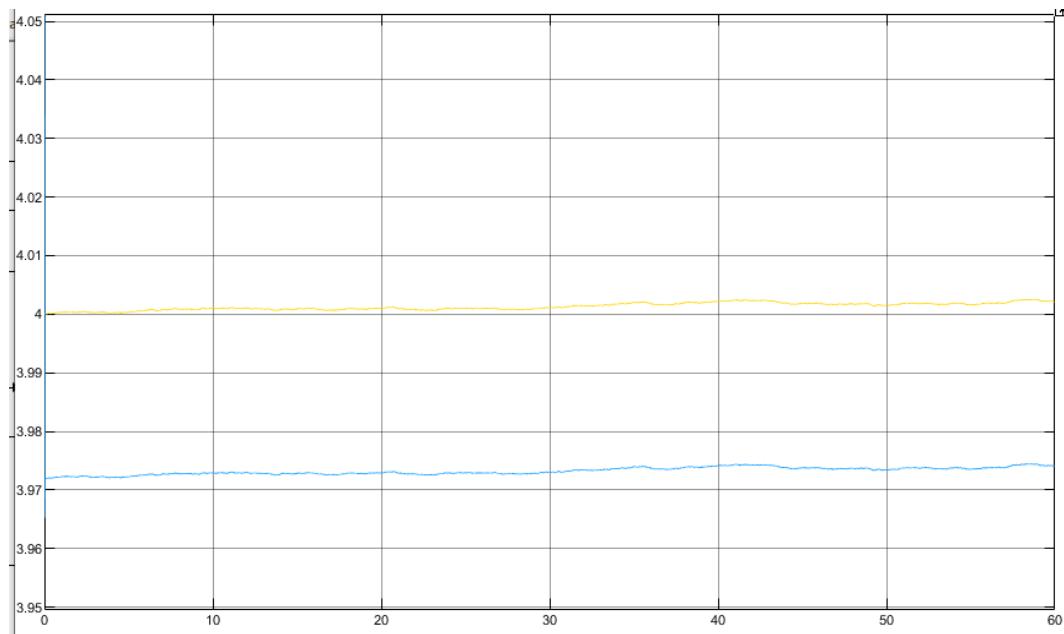


Figura 35: Prueba de Filtro de gravedad en Simulink, ángulo estimado vs medido.

## Implementación

Para la correcta implementación del filtro de Kalman se realizan los siguientes pasos:

- Al inicializar el robot se calculan los ángulos de inclinación iniciales y la deriva de los giróscopos.
- Durante el movimiento, cada vez que se realice la conmutación de los PAP, se mide el movimiento según la cantidad de pulsos y la dirección de cada motor.
- Se leen los sensores del MPU6050, se elimina la deriva de la medición y se registra el tiempo de la medición.
- Se realiza la corrección para eliminar la aceleración de la gravedad.
- Se aplica el algoritmo de Kalman.
- Se convierte el resultado de coordenadas locales a absolutas, y los radianes a grados mediante:

$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & \frac{180}{\pi} \end{bmatrix}$$

## Control de actuadores

En esta sección se detallan los fundamentos matemáticos y la implementación del control tanto de los motores PAP como del servo.

Por cuestiones de carga informática se decide hacer un control distribuido donde desde el programa de la interfaz se realizan los cálculos pertinentes a los PAP, para enviar posteriormente la consigna en formato de ancho de pulso con signo. Con dicha consigna en el Arduino se realizará el movimiento con aceleración constante para evitar sobreesfuerzos, además de reducir la pérdida de pasos, a su vez enviando a los drivers A4988 la consigna en forma de pulsos y dirección. Finalmente, el servo se controla mediante incrementos angulares recibidos desde la terminal donde se corre la interfaz, y posteriormente convertidos en PWM de forma local en el Arduino.

Ambas consignas son generadas por el programa de la terminal por un joystick de dos sticks, en este caso se utilizó un joystick modelo Redragon Harrow Pro G808, se asignan las consignas  $v_x$  y  $v_y$  a los ejes correspondientes del stick izquierdo, mientras que la consigna  $\omega$  y  $S$  se asocian a los ejes X e Y del stick derecho.

La implementación de cada una de las partes puede encontrarse en los anexos, concretamente en el código de “CoordinadorEjes.cpp” en la sección de código del Arduino, y en “Lector\_USB.cs” en el código de Unity.

## Control en Arduino con librería Accelstepper

En esta sección se describen los cálculos necesarios para utilizar la librería Accelstepper de Arduino en modo control de velocidad, nótese que en dicho modo la librería no controla aceleraciones, por tanto, dicho control debe programarse manualmente.

Para calcular la aceleración del motor se parte de la dinámica inversa obtenida anteriormente

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} \left(4 * M * \frac{\pi d}{3} + I * \frac{\pi d}{3L} + I_r\right) & \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) & \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) \\ \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) & \left(M \frac{\pi d}{3} + M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r\right) & \left(M \frac{\pi d}{3} - M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L}\right) \\ \left(I * \frac{\pi d}{3L} - 2 * M * \frac{\pi d}{3}\right) & \left(M \frac{\pi d}{3} - M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L}\right) & \left(M \frac{\pi d}{3} + M \frac{\sqrt{3}\pi d}{3} + I * \frac{\pi d}{3L} + I_r\right) \end{bmatrix} * \begin{bmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix}$$

Para obtener la dinámica directa:

$$\tau = M_{3x3} * \dot{\omega}$$

$$M_{3x3}^{-1} \tau = \dot{\omega}$$

Debido a que los drivers Pololu usan PWM con mayor corriente de la nominal, utilizando para ello la batería de 12V, lo que aporta mayor fuerza durante la conmutación, se utiliza el torque medio que es igual al torque nominal del PAP:

$$\tau_{med} = 10Ncm = 0,1Nm$$

Calculando numéricamente en Matlab se obtiene:

$$\dot{\omega} = \begin{bmatrix} 3,83rad/s^2 \\ 3,83rad/s^2 \\ 3,83rad/s^2 \end{bmatrix}$$

$$\dot{\omega} = 219,27^{\circ}/s^2$$

Expresado en pulsos:

$$\dot{\omega} = 197,34P/s^2$$

$$\dot{\omega} = 197,34PPS/s$$

Expresado como periodo y en milisegundos:

$$T = 5,07ms/PPS$$

La aceleración se realiza con un periodo de:

$$T = 1ms/PPS$$

Se usa como velocidad máxima 500PPS en lugar de los 1000PPS sugeridos por la librería Accelstepper de Arduino, esto con el reducir el objetivo de reducir la pérdida de pasos a mayores velocidades. Se define entonces:

$$\omega_{max} = 500PPS$$

La conmutación de  $\omega_{max}$  a  $-\omega_{max}$  se realiza en un periodo de conmutación  $T_c = 1s$ , por lo que intentará seguir la entrada enviada desde la terminal, se considera que puede llegar a perderse pasos, especialmente al acelerar a las velocidades más altas con baja batería, sin embargo se considera que el filtro de Kalman puede solventar dicha pérdida, mientras se mantiene una buena maniobrabilidad.

## Control de velocidades angulares en Unity

En esta sección se presenta el cálculo de velocidades en Unity, dicho cálculo se realiza de forma previa al control en Arduino, pero se decidió colocarla después debido a que utiliza la velocidad en ancho de pulsos máxima obtenida previamente.

Recordando la cinemática inversa:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{\pi d} & \frac{L}{\pi d} \\ -\frac{\sqrt{3}}{2\pi d} & \frac{1}{2\pi d} & \frac{L}{\pi d} \\ \frac{\sqrt{3}}{2\pi d} & \frac{1}{2\pi d} & \frac{L}{\pi d} \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Se expresa como un sistema de ecuaciones:

$$\begin{cases} v_x * \frac{\sqrt{3}}{\pi d} = (-\omega_2 + \omega_3) \\ v_y * \frac{3}{\pi d} = (-2\omega_1 + \omega_2 + \omega_3) \\ \omega * \frac{3L}{\pi d} = (\omega_1 + \omega_2 + \omega_3) \end{cases}$$

Se define la acción de los sticks como:

$$\begin{cases} LY = (-\omega_2 + \omega_3) \\ LX = (-2\omega_1 + \omega_2 + \omega_3) \\ RX = (\omega_1 + \omega_2 + \omega_3) \end{cases}$$

Se define así debido a que los ejes del robot se encuentran rotados  $90^\circ$  respecto a los sticks. Reemplazando y reordenando se obtiene:

$$\begin{cases} v_x = LY * \frac{\pi d}{\sqrt{3}} \\ v_y = LX * \frac{\pi d}{3} \\ \omega = RX * \frac{\pi d}{3L} \end{cases}$$

Recordando la cinemática directa:

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} 0 & -\frac{\sqrt{3}\pi d}{3} & \frac{\sqrt{3}\pi d}{3} \\ -\frac{2\pi d}{3} & \frac{\pi d}{3} & \frac{\pi d}{3} \\ \frac{\pi d}{3L} & \frac{\pi d}{3L} & \frac{\pi d}{3L} \end{bmatrix} * \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

Se obtiene el sistema de ecuaciones:

$$\begin{cases} \omega_1 = -\frac{1}{\pi d} * v_y + \frac{L}{\pi d} * \omega \\ \omega_2 = -\frac{\sqrt{3}}{2\pi d} * v_x + \frac{1}{2\pi d} v_y + \frac{L}{\pi d} * \omega \\ \omega_3 = \frac{\sqrt{3}}{2\pi d} * v_x + \frac{1}{2\pi d} v_y + \frac{L}{\pi d} * \omega \end{cases}$$

Reemplazando con lo obtenido previamente se obtiene:

$$\begin{cases} \omega_1 = -\frac{1}{\pi d} * LX * \frac{\pi d}{3} + \frac{L}{\pi d} * RX * \frac{\pi d}{3L} \\ \omega_2 = -\frac{\sqrt{3}}{2\pi d} * LY * \frac{\pi d}{\sqrt{3}} + \frac{1}{2\pi d} LX * \frac{\pi d}{3} + \frac{L}{\pi d} * RX * \frac{\pi d}{3L} \\ \omega_3 = \frac{\sqrt{3}}{2\pi d} * LY * \frac{\pi d}{\sqrt{3}} + \frac{1}{2\pi d} LX * \frac{\pi d}{3} + \frac{L}{\pi d} * RX * \frac{\pi d}{3L} \end{cases}$$

$$\begin{cases} \omega_1 = -\frac{1}{3} * LX + \frac{1}{3} * RX \\ \omega_2 = -\frac{1}{2} * LY + \frac{1}{6} * LX + \frac{1}{3} * RX \\ \omega_3 = \frac{1}{2} * LY + \frac{1}{6} * LX + \frac{1}{3} * RX \end{cases}$$

Se corrige para que cada entrada pueda hacer uso completo de al menos un motor manteniendo la proporción, se multiplica LX por 2, LY por 3 y RX por 3, queda:

$$\begin{cases} \omega_1 = -LX + RX \\ \omega_2 = -LY + \frac{1}{2} * LX + RX \\ \omega_3 = LY + \frac{1}{2} * LX + RX \end{cases}$$

Finalmente, como el sistema de referencia del joystick es distinto al propuesto invertimos la rotación y el movimiento en el eje Y:

$$\begin{cases} \omega_1 = LX - RX \\ \omega_2 = -LY - \frac{1}{2} * LX - RX \\ \omega_3 = LY - \frac{1}{2} * LX - RX \end{cases}$$

Teniendo en cuenta el rango de valores de los sticks que es de -1 a 1, se realiza la normalización en caso de que el valor absoluto de uno o varios valores obtenidos fuera mayor a 1. Dicha normalización se realiza dividiendo el valor obtenido para cada motor por el valor más alto.

Finalmente se multiplica el resultado por  $\omega_{max} = 500$  para obtener la velocidad en PPS.

## Inteligencia artificial

En esta sección se detallan cada uno de los pasos utilizados para la generación de la inteligencia artificial de reconocimiento de señales, esto incluye desde la búsqueda de imágenes y generación del dataset hasta la implementación y resultados finales. También en esta sección se detallan las herramientas utilizadas para cada parte del proceso.

### Generación del dataset

Para la generación del dataset se utilizó en primer lugar el paquete Unity Perception, el cual permite crear un dataset sintético mediante aleatorizaciones y posterior detección y etiquetado de los objetos en cámara, permitiendo además simular Lidar, Radar y Sonar entre otros sensores, refiriéndose en este proyecto a la cámara que es la herramienta utilizada, dicha herramienta genera una imagen por foto y a dicha imagen adjunta a elección del usuario se adjunta un documento con un mapa de segmentación semántica donde se identifican las clases y los rectángulos de detección, y una máscara de segmentación para crear mapas de segmentación. El dataset resultante se genera en formato SOLO (Synthetic Optimized Labeled Objects).

Para el presente proyecto se utilizó la imagen generada y el documento con los rectángulos de detección junto con la detección semántica. Se realizaron los siguientes pasos:

- Se crea un nuevo proyecto con 3D HDRP activado en la versión de Unity 2021.3.11.f1.
- En el administrador de paquetes se descarga la librería de Unity Perception.
- Se crea una nueva escena, se coloca la cámara en posición 0,0,-27 con un campo de visión vertical igual a 27. Se le agrega el script Perception Camera con los etiquetadores BoundingBox2DLbeler, RenderedObjectInfoLabeler y SemanticSegmentationLabeler.
- Se configura la luz a luz direccional con luz direccional sin sombra, intensidad de 10 Lux y rotación de 0,60,0. Se crea otra luz direccional con los mismos parámetros y rotación 0,-60,0. Esto para simular una fuente de luz compleja como en el caso de dos ventanas e iluminación natural.
- Se agregan los objetos a detectar, en este caso objetos triangulares con la señalización deseada a los que se etiqueta como se muestra en la siguiente tabla:

Clase	Biológico	Corrosivo	Inflamable	Radioactivo	Tóxico
Simbolo					

Tabla 5: Señales y sus clases correspondientes.

- Se configuran los archivos SemanticSegmentationLabelConfig y IdLabelConfig con los objetos a detectar. Dicha configuración se aplica a los detectores de la cámara.
- Se crean los objetos de fondo, los cuales vienen en dos variedades, unos en formas cúbicas, esféricas, cilíndricas y de cápsula y otro en forma de una pared que se coloca entre estos y los objetos a detectar.
- Se crea un escenario de simulación y se programa el conteo de iteraciones a 900.
- Se asocian los objetos de fondo a la aleatorización BackgroundObjectPlacementRandomizer, el cual colocará los objetos de fondo de manera aleatoria en la zona de fondo. Esto no incluye la pared.
- Se asocian los objetos de fondo con la aleatorización TextureRandomizer, la cual asigna al azar una textura entre las que se incluyen texturas de madera, metal, piedras y ladrillos, e imágenes de stock de personas, laboratorios y depósitos en formato png de 256X256 píxeles. Dichas texturas e imágenes no deben contener la señalización a detectar.
- Se asocian dichos objetos con la aleatorización HueOffsetRandomizer que varía ligeramente el color de las texturas de fondo.
- Se asocian los objetos de fondo con la aleatorización RotationRandomizer, la cual los hace girar entre 0 y 360 grados en los tres ángulos de Euler.
- Se asocian los objetos a detectar con la aleatorización ForegroundObjectPlacementRandomizer, la cual coloca las señales al frente de forma visible en un área designada de 4.6X3.3 y con una distancia de separación de 1.5 según los ejes coordinados de Unity.
- Se asocian los objetos a detectar con la aleatorización MyRotationRandomizer, dicha aleatorización produce una rotación entre 30 y -30 grados a los objetos, además de escalarlos entre 0.0025 y 0.006 para simular distintos ángulos de cámara y distancias.
- Se asocian las luces a con la aleatorización MyLightRandomizer, la cual multiplica la intensidad de las mismas por un valor entre 0.7 y 5 y cambia el color de las luces.
- La aleatorización MyActivationTag no tiene ningún objeto asignado, en lugar de ello activa o desactiva el dibujado de la pared de fondo y le asigna una textura con el objetivo de dar un entorno más ordenado que los otros objetos de fondo. Dichas texturas se eligieron teniendo en cuenta el tipo de ambiente de trabajo incluyendo sillas, mesas, diferentes edificios incluyendo la Facultad de Ingeniería, personas, cajas, elementos de oficina y mascotas. Además, posee la capacidad de desactivar la ubicación de objetos a detectar, con el objetivo de crear ejemplos nulos a partir de cierta cuenta de objetos. En primera instancia de generación no se usa.
- Se ajusta la resolución de salida a 640X480 para que coincida con la cámara de la Raspberry.
- Se corre el código con el botón Play de Unity y se espera la generación de las 900 imágenes.

- Posteriormente en MyActivationTag se cambia el número de imágenes con detecciones deseadas a cero y se generan 300 imágenes nulas con el objetivo que el detector tenga menos falsos positivos. Se aclara más adelante el porqué de esta separación respecto al paso anterior.

El código de las aleatorizaciones MyRotationRandomizer, MyLightRandomizer y MyActivationTag se encuentran en la sección del anexo “Código Dataset Generator”.

## Procesamiento y aumentación del dataset

Para poder utilizar el dataset es necesario convertirlo de formato SOLO a COCO (Common Objects in Context) con la utilización de la herramienta Pysolotools. Es importante aclarar que dicha herramienta no puede procesar los ejemplos nulos, por lo que solo se convierten las 900 imágenes con detecciones, mientras que a las imágenes nulas se les quitan los archivos de detecciones.

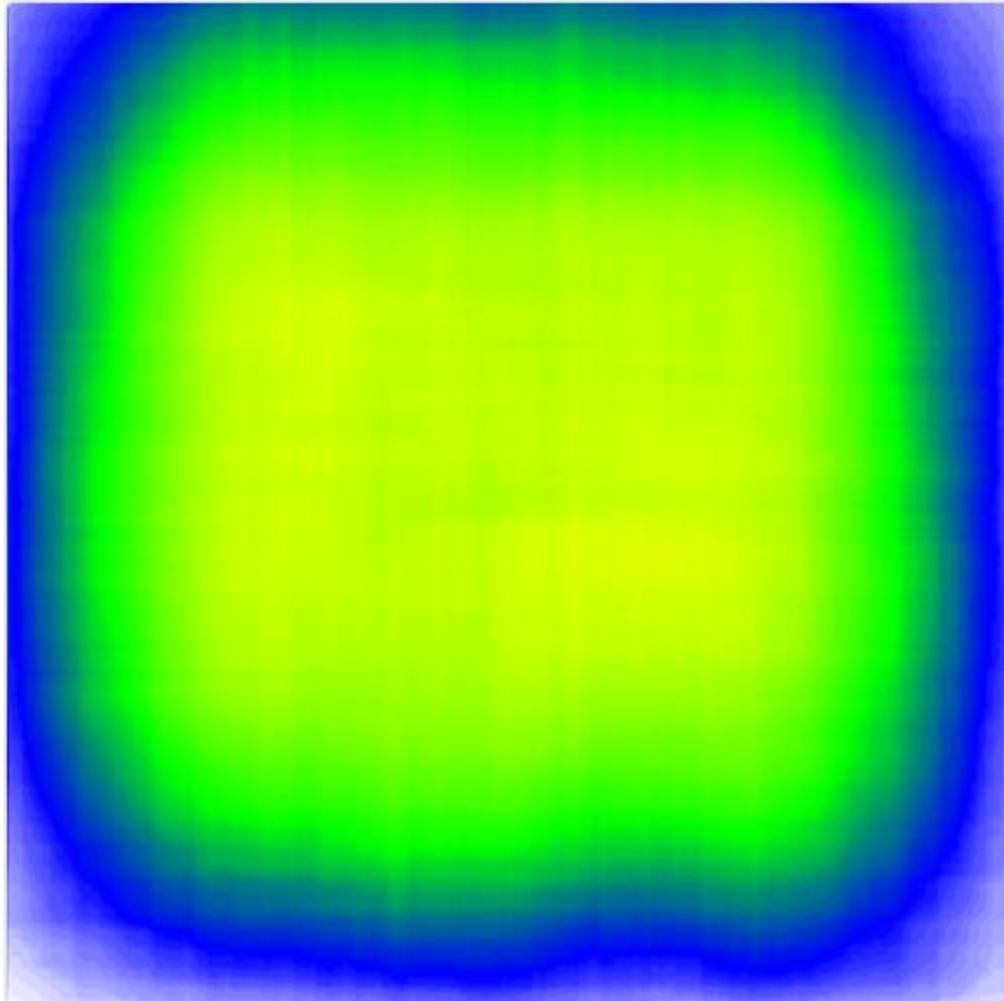
Posteriormente se cargan las 900 imágenes con detecciones a Roboflow, este proceso tarda varios minutos ya que al cargarse se leen las detecciones permitiendo procesos de aumentado, se agregan las imágenes al dataset y se repite el proceso con las otras 300 imágenes, con la excepción de que al no tener archivo de detecciones se pedirá un etiquetado, en dicho etiquetado se debe seleccionar la opción de “Mark (300) images as Null”. En ambos casos se debe asignar todas las imágenes a entrenamiento para poder utilizar la aumentación gratuita de Roboflow.

Posteriormente puede realizarse una revisión de salud del dataset, esto incluye verificar el histograma de anotaciones por imagen, balance de clases, mapas de calor para cada una de las clases. En cada caso se busca que las imágenes estén bien distribuidas entre clases (una diferencia en el balance de clases mayor al 10% puede ocasionar Overfitting en dicha clase). Se muestra a continuación el gráfico de balance de clases:



*Figura 36: Balance de clases, Roboflow.*

Puede observarse que entre la clase más representada y la menos hay un desequilibrio del 9%, lo que entra dentro de lo deseado. Se muestra también el mapa de calor para todas las clases:



*Figura 37: Mapa de calor, Roboflow.*

Puede observarse una correcta distribución al centro de la imagen, pero quedando un margen hacia abajo con muy pocas detecciones, lo cual puede llegar a ocasionar que la detección sea menos precisa en esa zona, como se busca mejorar la detección hacia el centro se considera un margen aceptable.

El histograma de conteo de objetos por imagen se muestra a continuación:

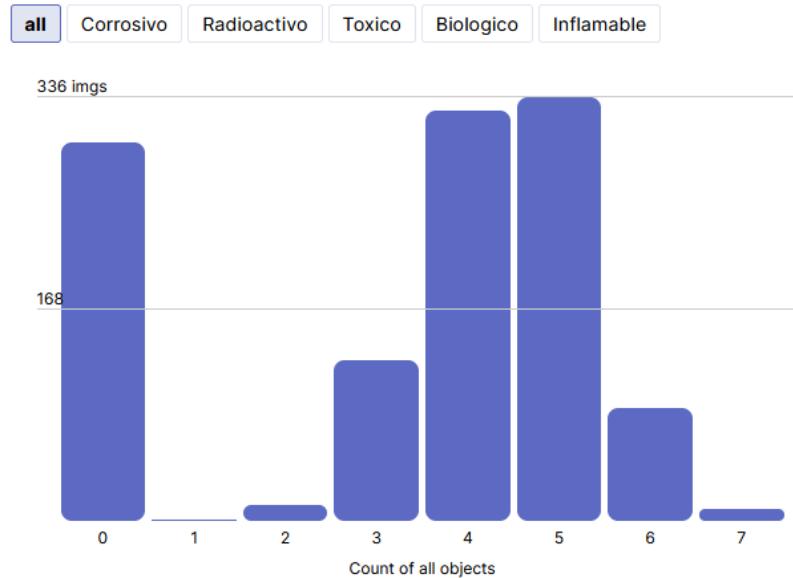


Figura 38: Histograma de recuento de clases por imagen, Roboflow.

Se puede tomar como ejemplo de clase la clase Biológico, donde puede observarse que se presenta gran mayoría de imágenes con un solo objeto de una clase, teniendo en segundo lugar dos detecciones representando prácticamente el 90% del dataset.

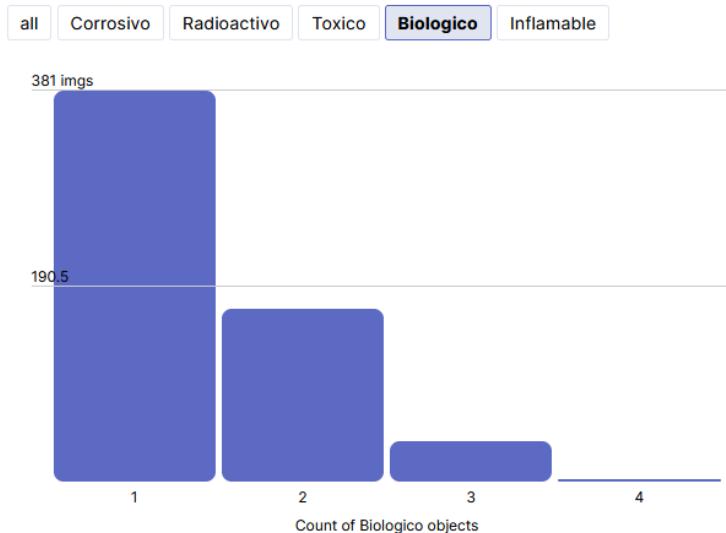


Figura 39: Histograma de recuento de clase “biológico” por imagen, Roboflow.

Posterior a dicha revisión se realiza la aumentación del dataset, lo que aumenta la cantidad de imágenes agregando una capa de procesamiento visual a la imagen. Roboflow permite aumentar hasta el triple el tamaño del dataset de entrenamiento. Se eligen los siguientes tipos de aumentación:

- Inversión horizontal y vertical.
- Rotación de 90° hacia la derecha, izquierda y de 180°.

- Zoom del 30%.
- Cizalladura del 10% horizontal y vertical.
- Matiz (Hue) de  $\pm 15^\circ$
- Saturación de  $\pm 25\%$
- Exposición de  $\pm 10\%$
- Difuminado (Blur) de 2,5 píxeles
- Ruido hasta 0,1% de los píxeles

Una vez procesado el dataset se elige la opción de exportar en formato Pascal VOC (Visual Object Classes), el cual posteriormente se dividirá en proporción 80/10/10 entre entrenamiento, validación y prueba al momento de convertirse a TFRecord para ser utilizado por Tensorflow.

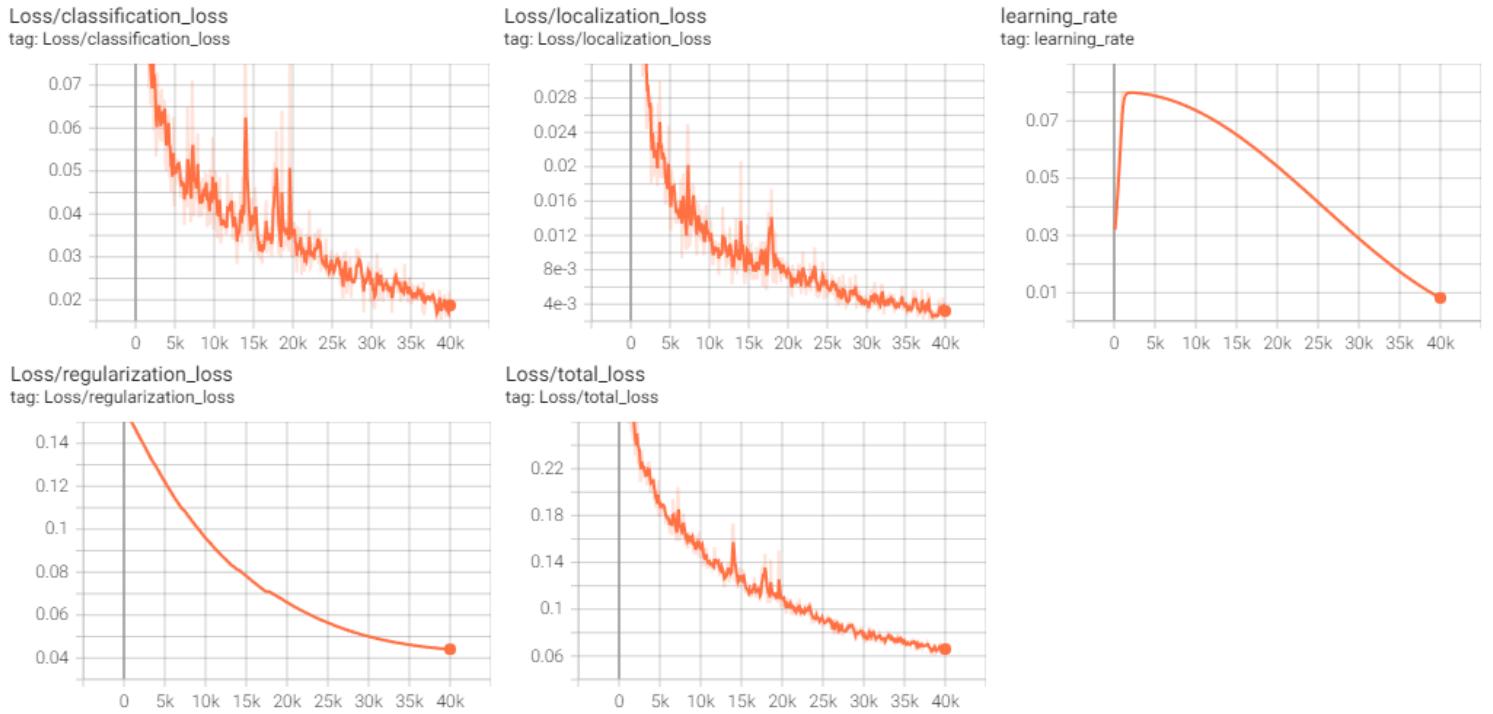
## Entrenamiento

Para el entrenamiento se utiliza una versión modificada del cuaderno de Google Colab de Evan Juras (puede encontrarse el enlace en la bibliografía), el cual además de código para realizar el entrenamiento incluye código para separar imágenes, convertir el dataset a TFRecord y crear un mapa de etiquetas. Se detallan en forma general los pasos realizados para el entrenamiento de Tensorflow Lite:

- Se instala en una computadora Docker y se crea un entorno virtual según las instrucciones de Google Colab para correr un cuaderno en entorno local. En este caso se utilizó como procesador gráfico una GPU RTX 3070.
- Con el entorno local creado y conectado a Google Colab se importan las librerías y dependencias de Tensorflow.
- Se cargan las imágenes en un zip llamado images sin carpetas, esto para evitar problemas de compatibilidad, dichas imágenes se dividen en tres carpetas correspondientes a entrenamiento, validación y prueba.
- Se crea el labelmap o mapa de etiquetas donde se definen los nombres de las clases. Además, se convierte el dataset en formato TFRecords.
- Se configura el entrenamiento, en este caso se utilizó el modelo ssd-mobilenet-v2-fpnlite-320 el cual se entrena con 40.000 pasos y un tamaño de lote de 16 imágenes, es capaz de leer imágenes con resolución de 320X320.
- Se configura el modo de Fine Tuning a Detección y se usa la configuración estándar para dicho modo.
- Se realiza el entrenamiento, en este caso duró aproximadamente 2 horas y media. Además, de forma opcional puede crearse un Tensorboard el cual permite ver el progreso del entrenamiento en tiempo real, sin embargo, en el entorno donde se probó la función de tiempo real no funcionó, por tanto, se realizó la revisión después del entrenamiento.
- Se convierte el modelo a Tensorflow Lite.
- Se prueba y se realiza el cálculo de la precisión promedio media o mean average precisión (mAP) y se descarga el modelo.
- Como pasos opcionales el modelo puede cuantizarse, lo cual aumenta la velocidad a costa de la precisión de la detección, además puede crearse una versión para correr en Google Coral, el cual

mejora considerablemente la velocidad de inferencia, pero como dicho periférico es considerablemente caro se descarta dicha versión.

A continuación, se muestra el Tensorboard obtenido con el entrenamiento:



*Figura 40: Pérdidas y tasa de entrenamiento, Tensorboard.*

Es necesario destacar que, si bien siempre se desea que se reduzca la pérdida total, también es necesario que la tasa de aprendizaje nunca llegue a cero, ya que esto puede ser síntoma de un Overfitting, lo cual no es deseable debido a que genera malas detecciones, especialmente en este caso ya que el dataset es completamente sintético pero la inteligencia artificial debe ser capaz de funcionar correctamente en un entorno real.

El mAP calculado para cada clase se muestra a continuación:

Clase	mAP
Biológico	87,87%
Corrosivo	87,19%
Inflamable	88,62%
Radioactivo	89,11%
Tóxico	87,62%
Total	88,08%

*Tabla 6: Precisión promedio media o mAP.*

## Implementación

Esta sección se agregan algunos detalles sobre la implementación de la inteligencia artificial generada en el paso anterior en la Raspberry Pi, puede verse la implementación completa en la sección de Códigos de Raspberry, en el objeto Tensorflow.

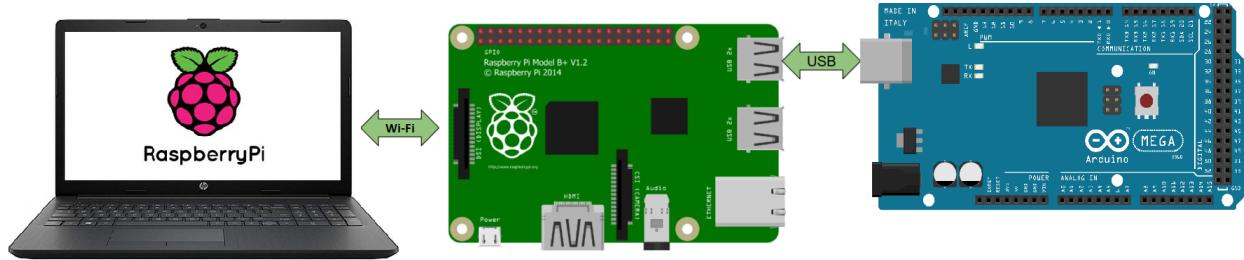
Debido al alto costo computacional de Tensorflow al momento de ejecutarse la inferencia se opta por ejecutarlo en su propio hilo, esto permite obtener algunos cuadros por segundo más, pero sin detección, por lo que se cuenta que el movimiento no sea lo suficientemente rápido para que la imagen se desfase respecto a la detección anterior, la cual se realiza a razón de entre 3,5 y 4 imágenes por segundo. Dicha detección se realiza con un umbral de confianza de 20%, esto es debido a que la clase Corrosivo experimentalmente da un umbral bajo de confianza, prefiriéndose una buena detección de las clases, pero teniendo a cambio algunas detecciones falsas.

Las detecciones se muestran en una imagen procesada cada vez que se envían a la terminal, dicho procesamiento incluye hacer un recorte de los rectángulos para evitar detecciones fuera del área de la foto para su posterior dibujado, generación de un rectángulo con la etiqueta y el valor de confianza de la detección.

## Comunicaciones

En esta sección se detalla el esquema de comunicaciones utilizado, así como algunas particularidades del mismo, mientras que en la sección de Comunicación Wi-Fi se explica el proceso de conexión para utilizar la Raspberry Pi como puerta de enlace.

El diagrama de comunicaciones utilizado se muestra a continuación:



*Figura 41: Diagrama de comunicaciones.*

La comunicación entre la notebook y la Raspberry se realiza mediante sockets TCP/IP, se utilizan tres sockets donde se envían distintos mensajes en codificación UTF-8, en dichos sockets la Raspberry funciona como servidor debido a que tiene una IP fija gracias al punto de acceso, en caso de necesitarlo esto también permite utilizar el servidor VNC incorporado en la Raspberry para utilizar el monitor remoto, sin embargo, el programa de la terminal funciona independientemente del mismo.

La comunicación entre Raspberry y Arduino se realiza mediante el puerto USB de la Raspberry, el cual se configura a 115200 Baudios, siendo la mayor velocidad recomendada para comunicación con una computadora.

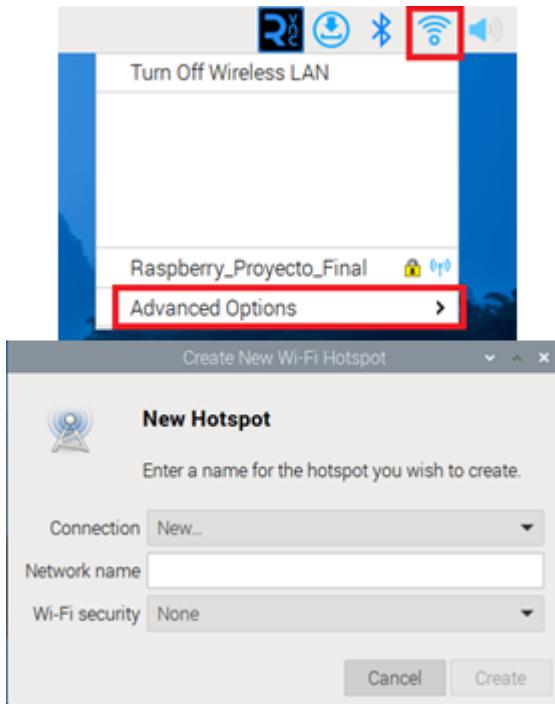
Se presenta a continuación una tabla que indica envío, receptor, canal y tramas posibles para las distintas comunicaciones, cabe destacar que en todo caso se utiliza el carácter ';' para marcar el final de un mensaje, en caso de envío Wi-Fi se agrega el puerto de dicha comunicación, la tabla mencionada es la siguiente:

Remitente	Destinatario	Canal	Mensaje y aclaración
Arduino	Raspberry	USB	D; Relé apagado
Raspberry	Notebook	Wi-Fi P40002	DF____; Relé apagado, Tasa de detecciones de Tensorflow
Raspberry	Notebook	Wi-Fi P40002	DFN; Relé apagado, Tensorflow apagado
Arduino	Raspberry	USB	X_Y_P_S_C__; Posición y temperatura del MPU6050 y el servomotor
Raspberry	Notebook	Wi-Fi P40002	X_Y_P_S_C_F__; Posición y temperatura del MPU6050 y el servomotor, Tasa de detecciones de Tensorflow
Raspberry	Notebook	Wi-Fi P40002	X_Y_P_S_C_FN; Posición y temperatura del MPU6050 y el servomotor, Tensorflow apagado
Raspberry	Notebook	Wi-Fi P40001	[Imagen]; Imagen capturada por la cámara y tratada con Tensorflow si este se encuentra activado
Notebook	Raspberry	Wi-Fi P40000	T; Comutar activación/desactivación de Tensorflow
Notebook	Raspberry	Wi-Fi P40000	R; Comutar activación/desactivación del relé
Raspberry	Arduino	USB	R; Comutar activación/desactivación del relé
Notebook	Raspberry	Wi-Fi P40000	W______; Consigna de velocidad para los tres PAP y el servo
Raspberry	Arduino	USB	W______; Consigna de velocidad para los tres PAP y el servo
Raspberry	Arduino	USB	D; Desconectar el relé, enviado en caso de desconexión de la terminal

Tabla 7: Comunicación y mensajes posibles.

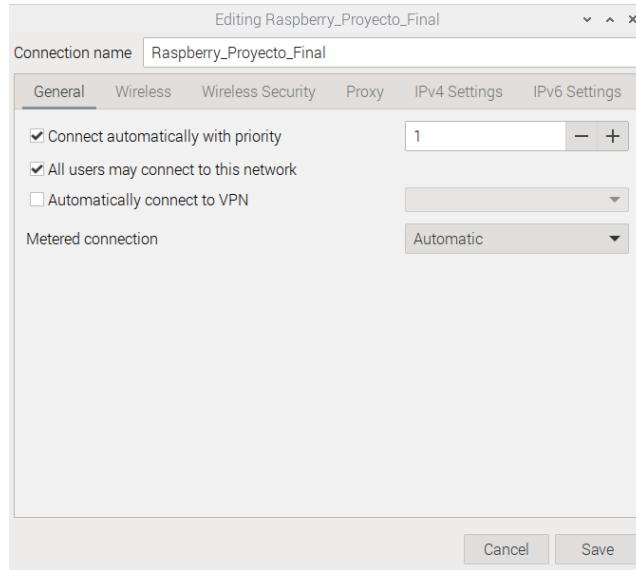
## Comunicación Wi-Fi

En este apartado se muestran los pasos para utilizar la Raspberry como puerta de acceso Wi-Fi, es importante aclarar que estos pasos son para la distribución Bookworm del sistema operativo Raspbian OS, por tanto, es probable que no funcione en otras versiones. Para comenzar se necesita entrar al administrador de redes y seleccionar "Opciones Avanzadas", luego se elige "Crear punto de acceso" y se da nombre y contraseña a la red como se muestra en las siguientes imágenes:



*Figura 42: creación de puerta de enlace, Raspbian Bookworm.*

Posteriormente se selecciona en “Opciones Avanzadas” la opción editar conexiones, se elige la pestaña pertinente y se hace doble clic, en la pestaña “General” se seleccionan las opciones “Conectar automáticamente con prioridad” y se le da una prioridad de 1 (a mayor número mayor prioridad, siendo el estándar 0), se guarda y se reinicia.



*Figura 43: configuración de puerta de enlace, Raspbian Bookworm.*

Luego de realizada esta configuración la Raspberry encenderá con la puerta de enlace Wi-Fi como red predeterminada y se asignará automáticamente la IP 10.42.0.1 la cual se utilizará para los sockets TCP/IP.

## Inicio y encendido

En esta sección se detallan los pasos a seguir para que la aplicación en Python se ejecute al iniciar la Raspberry, esto es necesario para evitar que un operario sin experiencia deba entrar al sistema operativo por medio del servidor VNC de la Raspberry, además de permitir la activación del robot solo pulsando el botón de encender y esperando a que la red esté disponible.

Para ello en primer lugar se debe crear un shell script, ya que Tensorflow requiere un entorno virtual para correr, dicho script se encuentra en el Anexo: Códigos de inicio Raspberry, posteriormente se debe generar un archivo desktop para que pueda iniciarse como aplicación. Dicho archivo debe tener una copia almacenada en la ruta /home/Usuario/.config/autostart permitiendo la inicialización del programa con una consola que permite saber si el programa ha tenido algún problema en caso de revisarse con un cliente VNC.

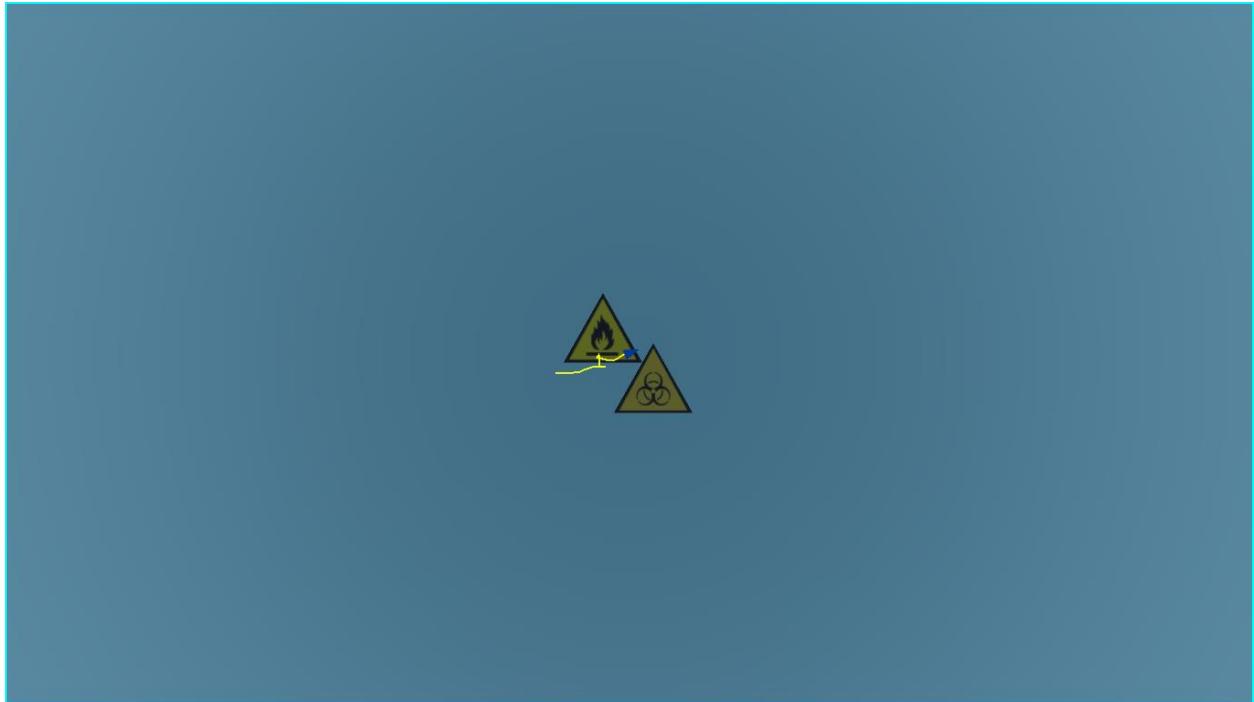
## Interfaz de usuario

En esta sección se presentan las funcionalidades de la interfaz desarrollada, así como el esquema de control utilizado.

La interfaz ha sido realizada en el motor Unity en la versión 2019.2.9f1, la cual el alumno a cargo del presente proyecto ya había utilizado previamente para realizar la interfaz de otro proyecto, siendo por tanto más sencillo programar en dicha versión.

La interfaz como tal presenta tres sub interfaces, dichas sub interfaces incluyen la interfaz principal, donde se puede acceder a todas las funciones del programa de Unity, la interfaz de cámara donde se puede ver a pantalla completa la imagen enviada por la Raspberry escalada a la resolución de la notebook utilizada (1366X768), finalmente la interfaz del mapa que muestra a pantalla completa el mapa de movimiento de la Raspberry con mayor resolución y menor zoom que en la interfaz principal, permitiendo ver una mayor área. A continuación, se pueden ver capturas de las tres interfaces:

*Figura 44: interfaz principal.**Figura 45: Interfaz de cámara a pantalla completa.*



*Figura 46: Interfaz de mapa a pantalla completa.*

Nótese en la interfaz de cámara a pantalla completa que se ha deformado la imagen, esto es debido a que no se ha mantenido la relación de aspecto respecto a la cámara para evitar dicho efecto.

Se detalla a continuación la función de cada elemento de interfaz y sus formas de operación.

### **Sección Cámara**

Esta sección es válida tanto para la cámara en la interfaz principal como en la interfaz a pantalla completa, dicha sección recibe la imagen desde el objeto Raspberry\_Comunicacion que la recibe a su vez desde la Raspberry (esto en todos los casos, por lo que posteriormente se omitirá en la explicación de las demás partes) y la imprime en pantalla como una imagen simple, el formato de recepción es PNG. En dicha sección, y si Tensorflow está activado, se pueden observar también las detecciones, además, se ha colocado un ligero contorno para separarla del resto de la interfaz. En la imagen además puede observarse una detección de la clase Inflamable con su confiabilidad.



Figura 47: Interfaz principal, sección Cámara.

Cabe aclarar que la tasa de imágenes es estable cuando la Raspberry se utiliza conectada a una red preexistente, mientras que cuando esta actúa como punto de acceso se pierde dicha estabilidad, pero no se ha podido determinar si es debido a que mayor cantidad de imágenes llegan con fallos (cosa que esta parte de la interfaz detecta y en lugar de actualizar la imagen mantiene la anterior) o si el envío de imágenes es mucho más lento.

### Sección Estado

En esta zona de la interfaz se muestra el estado general del robot, incluyendo el estado de la conexión con la terminal, del relé y del Tensorflow, para ello se lee el flag de conexión y la última trama recibida por el socket de navegación y se compara con la trama de las distintas partes correspondientes a su desactivación, colocando ON u OFF según sea el caso.

Conexión: ON  
Relé: ON  
Tensorflow: ON

Figura 48: Interfaz principal, sección Estado.

### Sección Controles

Esta sección está pensada para permitir al usuario identificar fácilmente que controles se emplean para permitir conocer el esquema de control respecto a un controlador de Xbox o compatible (en este caso el Redragon Harrow Pro), no siendo más que un recuadro de texto plano.

```

Start: Conmutar Relé
Stick Izquierdo: Movimiento
Stick Derecho: Orientación
Y: Conmutar Tensorflow
A: Conmutar Interfaz
RT: Mapa
LT: Ubicar ícono
LB: Cancelar ícono
Pad: Cambiar y mover ícono
Select: Salir

```

*Figura 49: Interfaz principal, sección Controles.*

## Sección Posición y Orientación

En esta sección se muestran la posición y orientación obtenidos de los datos de navegación sin procesar y a modo informativo, se muestra a continuación el caso con y sin conexión.

```

X: 5.30
Y: 1.43
R: 20.66
S: 80
X: _____.
Y: _____.
R: _____.
S: _____.

```

*Figura 50: Interfaz principal, sección Posición y Orientación.*

## Sección FPS y Temperatura

En esta sección se muestran tanto los frames o imágenes por segundo (FPS) que procesa Tensorflow como la temperatura del MPU6050 a modo informativo. En caso de estar desactivados los relés la temperatura no se mostrará, en caso de estar desactivado Tensorflow los FPS no se mostrarán. Un detalle a tener en cuenta es que si se miden los FPS con la Raspberry funcionando como punto de acceso se obtiene hasta 2.90 FPS, mientras que la Raspberry conectada a una red común suele dar 3.90 FPS, lo cual deja en evidencia el alto consumo que genera el punto de acceso.

```

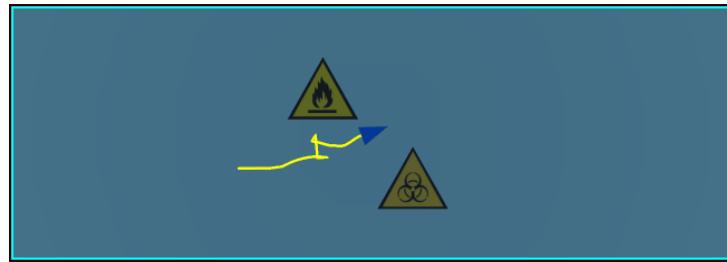
FPS: 2.76
Temp: 31.12

```

*Figura 51: Interfaz principal, sección FPS y Temperatura.*

## Sección Minimap y Mapa

Esta sección cubre el funcionamiento tanto del mapa a pantalla completa como del minimapa en la interfaz principal ya que tienen el mismo funcionamiento.



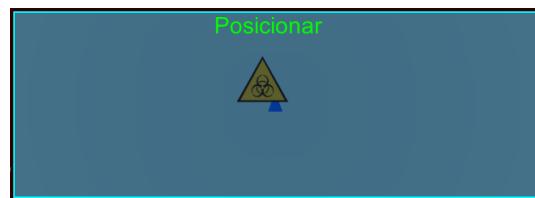
*Figura 52: Interfaz principal, sección Minimap.*

Internamente se crea una cámara en el motor Unity que graba un plano y un objeto triangular azul cuyo movimiento representa la posición y orientación del robot, dicha cámara se mueve junto con el robot permaneciendo este en el centro, además la fuente de luz de la escena se mueve con las cámaras. Ambas cámaras funcionan con proyección ortográfica para mantener una vista plana del área, la cámara del mapa tiene un tamaño de 25 mientras que la del minimapa tiene un tamaño de 5, lo cual representa el área mostrada. Ambas cámaras muestran los objetos de todas las capas excepto de la capa identificada por la otra cámara.

La colocación de objetos funciona moviendo una de las cinco plantillas disponibles sobre el robot, dichas plantillas contienen el mismo objeto utilizado para crear las señales en la sección de generación del dataset, dicho objeto representa la señal que se desea colocar y posee dos tamaños, uno para el mapa y otro para el minimapa representados por dos objetos separados, pero con distinto identificador de capa según en qué cámara debe mostrarse para evitar perder la visibilidad de la señal (aunque en el mapa se ve más grande en relación con el robot y la trayectoria). En un primer paso de la colocación se elige entre las plantillas y posteriormente puede moverse dicho objeto, al volver a tocar el botón de colocación se coloca por debajo del robot una copia del objeto de plantilla (el cual vuelve a su posición oculta debajo del plano), así puede seguirse viendo la trayectoria y el ícono del robot sin problema. Se muestra a continuación como se muestran al usuario los estados de la colocación donde se encuentra y el resultado final:



*Figura 53: Interfaz principal, sección Minimap, modo selección de señales.*



*Figura 54: Interfaz principal, sección Minimap, modo posicionamiento de señales.*



*Figura 55: Interfaz principal, sección Minimap, señal posicionada.*

Finalmente, en esta sección de la interfaz se dibuja en amarillo la trayectoria seguida del robot y el ícono del robot con la orientación obtenida de la navegación, cabe aclarar que si no se detecta movimiento de 1cm no se dibuja un nuevo segmento de línea.

## Materiales y costos

Se detallan en esta sección los materiales utilizados y sus costos. Se tiene en cuenta debido a la situación económica del país argentino los precios a fecha del 19 de febrero de 2024, así como por simplicidad se tienen en cuenta que los materiales sean accesibles mediante el sitio web de mercado libre, aunque en caso de no ser posible se aclara un reemplazo, se busca ser representativo de los costos de los materiales, por tanto se agrega al costo total de los mismos el costo de envío, sin embargo, debe tenerse en cuenta que una parte de los materiales listados ya se encuentran a disposición del alumno antes de la realización del mismo. Dicha lista de materiales se muestra a continuación:

Material	Cantidad	Costo total
Raspberry Pi 4B, 4GB con Wi-Fi	1	104.000
Tarjeta de memoria de 64GB	1	7.024
Cámara V2 para Raspberry Pi	1	129.000
Arduino Mega 2560	1	32.500
UPS MakerHawk EP-0136	1	51.669
Batería 18650 2200mAh	2	16.989
Acelerómetro y Giróscopo MPU6050	1	9.609
Modulo Doble Relé Arduino	1	10.299
Batería Gel Vapex VT1213	1	19.900
Driver Pololu A4988	3	8.630
Motor Paso a paso 1kgf/cm y paso de 0,9 grados 42HS028 (precio de Nema 17)	3	300.393
Servomotor Tower Pro Sg90	1	9.094
Rueda omnidireccional 58mm (precio de Alibaba)	3	75.949
Set de 4 Ruedas esféricas para muebles	1	10.000
Partes impresas 3D (por encargo)	1	13.800
Partes varias (tornillos, cables, etc.)	1	30.000

Total	-	828.856
-------	---	---------

Tabla 8: Materiales y costos

Además, se da un listado de herramientas que utilizará el alumno, el costo de dichas herramientas no se agrega al costo total del proyecto, dicho listado se presenta a continuación:

Herramienta	Tipo
Microsoft Office	Software
Solid Edge ST6	Software
Raspbian Pi OS (Bookworm)	Software
OpenCV	Software
Google Colab	Software
MATLAB	Software
Derive 6	Software
Cirkit Designer	Software
Arduino IDE	Software
Unity y Unity Perception Package	Software
Anaconda	Software
Computadora	Electrónica
Lector de tarjetas SD	Electrónica
Cables para impresora	Electrónica
Soldador	Electrónica
Cargador de baterías de 15V	Electrónica
Cargador de baterías de 5V	Electrónica
Joystick Redragon Harrow Pro G808	Electrónica
Destornillador, llave de tuercas, calibre de Vernier y herramientas varias	Mecánica

Tabla 9: Herramientas utilizadas

## Posibles mejoras

En esta sección se tratan las posibles mejoras a aplicar sobre el presente proyecto, estas son:

- Acelerar la inferencia de las imágenes con el periférico Google Coral o cambiando la Raspberry por un miniordenador Nvidia Jetson Nano para mejorar la tasa de imágenes procesadas.
- Reemplazar el modelo de Tensorflow Lite por un modelo más grande y preciso, agregando imágenes reales al dataset sintético para mejorar el rendimiento.
- Reemplazar los MPU6050 con sensores con magnetómetros y que posean mayor precisión (debido a su baja precisión solo puede corregir la pérdida del 1% de pasos).
- Realizar cambios para que el programa de la interfaz tenga soporte para más tipos de mandos y para mouse y teclado.
- Cambiar el escalado de las imágenes para que no se note estirada en la interfaz de cámara a pantalla completa.
- Realizar una versión de la interfaz para móviles.

- Agregarle un disipador y coolers a la Raspberry, ya que suele calentarse de forma considerable y puede llegar a impactar en el rendimiento de la misma.
- Sustituir la UPS por un regulador para tener solo un punto de carga de batería.
- Agregar medidores de batería tanto para la carga como para la interfaz.
- Mejorar el diseño del robot para facilitar el mantenimiento.

## Conclusiones

Se considera que el proyecto ha cumplido todos los objetivos en cuanto a las actividades a desarrollar, sin embargo, cabe destacar que la duración fue de aproximadamente un año y medio en contraposición a los cuatro meses que se planteó inicialmente, esto debido a que el proyecto ha sido muy ambicioso desde el principio, sin tener en cuenta las partes desarrolladas previamente del mismo en otras materias ni las dificultades referentes al mismo y a la vida personal del alumno. Queda pendiente además el presentar el presente proyecto a la fuerza de bomberos como actividad de extensión para evaluar la viabilidad del uso de una interfaz aumentada para las tareas de búsqueda y rescate.

En cuanto al rendimiento de las partes todas funcionan adecuadamente, sin embargo, la Raspberry se encuentra sobreexigida, además que el Tensorflow tiene todavía problemas en ciertas perspectivas y distancias, no teniendo el mejor rendimiento. Esto posiblemente sea debido a la naturaleza completamente sintética del dataset de entrenamiento y a la falta de procesamiento previo a la detección, debido a esto se puede notar una discrepancia entre el mAP de más del 87% en todas las clases con la tasa real de detecciones que a veces no superan el 40%, y otras veces presenta detecciones erróneas y falsos positivos con un margen alto de certeza.

En cuanto al contenido se cubrieron en su gran mayoría todas las áreas que integran la mecatrónica, siendo un proyecto integrador de la carrera en su conjunto.

## Palabras Clave

Robótica, visión, realidad aumentada, HMI

## Bibliografía

Libro: "Fundamentos de Robótica", autor: Antonio Barrientos, editorial Mc Graw Hill.

Artículo: "Cómo la IA puede ayudar a salvar vidas en situaciones de emergencia", Microsoft news, enlace: <https://news.microsoft.com/es-xl/features/como-la-ia-puede-ayudar-a-salvar-vidas-en-situaciones-de-emergencia/>

Artículo: "Robots al rescate: Uso de la tecnología para mitigar los efectos de los desastres naturales", boletín de la Organización de las Naciones Unidas, enlace: <https://www.un.org/es/robots-al-rescate-uso-de-la-tecnolog%C3%ADA-para-mitigar-los-efectos-de-los-desastres-naturales>

Artículo: "Mobile Robot", Wikipedia en inglés, enlace: [https://en.wikipedia.org/wiki/Mobile\\_robot](https://en.wikipedia.org/wiki/Mobile_robot)

Artículo: “Three Laws of Robotics”, Wikipedia en inglés, enlace: [https://en.wikipedia.org/wiki/Three\\_Laws\\_of\\_Robotics](https://en.wikipedia.org/wiki/Three_Laws_of_Robotics)

Artículo: “The History of Machine Vision – Timeline”, Robotics Biz, enlace: <https://roboticsbiz.com/the-history-of-machine-vision-timeline/>

Artículo: “ISO 7010”, Wikipedia en inglés, enlace: [https://en.wikipedia.org/wiki/ISO\\_7010](https://en.wikipedia.org/wiki/ISO_7010)

Documento: “IRAM 10005”, sitio: slideshare, enlace: <https://es.slideshare.net/FaniiNavarro/iram-10005pdf>

Tienda: “Mercado Libre”, enlace: <https://www.mercadolibre.com.ar/>

Paper: “MODELADO CINEMATICO Y DINAMICO DE UN ROBOT MÓVIL OMNI-DIRECCIONAL.”, autores: V. F. Muñoz Martínez, G. Gil-Gómez y A. García Cerezo. Instituto Andaluz de Automática Avanzada y Robótica. Dpto. Ingeniería de Sistemas y Automática. Universidad de Málaga. Parque Tecnológico de Andalucía,  
enlace: <https://intranet.ceautomatica.es/old/actividades/jornadas/XXIV/documentos/ro/201.pdf>

Trabajo de fin de grado en Ingeniería aeroespacial: “Estudio y aplicación del filtro de Kalman en fusión de sensores en UAVs.”, autora: María Esther Aranda Rosamanta, tutora: Juana María Martínez Heredia. Escuela Técnica superior. Dpto. ingeniería electrónica. Universidad de Sevilla, enlace: [https://idus.us.es/bitstream/handle/11441/66071/TFG\\_M%C2%AA%20Esther%20Aranda%20Romasanta.pdf](https://idus.us.es/bitstream/handle/11441/66071/TFG_M%C2%AA%20Esther%20Aranda%20Romasanta.pdf)

Paper: “Observability of Sensorless Electric Drives”, autores: Mohamad Koteich, Gilles Duc, Abdelmalek Maloum, Guillaume Sandou, revista: Research Gate Enlace: [https://www.researchgate.net/publication/301846354\\_Observability\\_of\\_Sensorless\\_Electric\\_Drives](https://www.researchgate.net/publication/301846354_Observability_of_Sensorless_Electric_Drives)

Discusión: “How much current can I draw from the Arduino's pins?”, foro: Stack Exchange, enlace: <https://electronics.stackexchange.com/questions/67092/how-much-current-can-i-draw-from-the-arduinoss-pins>

Datasheet: MPU-6000/MPU-6050 Product Specification, autor: InvenSense, enlace: <https://pdf1.alldatasheet.com/datasheet-pdf/view/517744/ETC1/MPU-6050.html>

Datasheet: 5V Dual-Channel Relay Module, sitio: Components 101, enlace: <https://components101.com/switches/5v-dual-channel-relay-module-pinout-features-applications-working-datasheet>

Artículo: USB 101: All You Need to Know About USB 3.2, sitio: TeleTec, enlace: <https://www.teletecsi.com/blogs/usb-101-all-you-need-to-know-about-usb-32>

Datasheet: A4988 DMOS Microstepping Driver with Translator and Overcurrent Protection, autor: Allegro, enlace: <https://pdf1.alldatasheet.com/datasheet-pdf/view/338780/ALLEGRO/A4988.html>

Datasheet: Servo Motor Micro SG90, sitio: Proto Suplies, enlace:  
<https://protosupplies.com/product/servo-motor-micro-sg90/>

Manual: Perception Package, autor: Unity Technologies, enlace:  
<https://docs.unity3d.com/Packages/com.unity.perception@1.0/manual/>

Cuaderno: TensorFlow Lite Object Detection API in Colab, Autor: Evan Juras, EJ Technology Consultants, enlace: [https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train\\_TFLite2\\_Object\\_Detection\\_Model.ipynb](https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detection_Model.ipynb)

## Anexo: Códigos Arduino (C++)

### ProyectoFinalArduino.ino

```
#include <Servo.h>
#include "Comunicacion.h"
#include "Navegacion.h"
#include "CoordinadorEjes.h"

Comunicacion ControlSerie;
Navegacion Kalman;
CoordinadorEjes Steppers;

void setup(void) {
    //Se inicia cada uno de los objetos
    ControlSerie.Iniciar();
    Kalman.Iniciar();
    Steppers.Start();
}

void loop() {
    //Se revisa si se ha recibido un mensaje completo, se lee y se dan las
    instrucciones correspondientes al coordinador
    if(ControlSerie.MensajeListo()){
        ControlSerie.LeerMensaje();
        Steppers.SetEstadoRele(ControlSerie.GetEstadoRele());
        Steppers.SetVelocidadObjetivo(ControlSerie.GetVelocidades());
    }
    Steppers.Actualizar();
    //Se actualiza la navegación si el timer eta listo
    if(Kalman.TimerListo()){
        Kalman.LeerSensores();
        Kalman.SetDatoEncoder(Steppers.GetDato());
        Kalman.CorreccionGravedad();
        Kalman.Kalman();
    }
    Steppers.Actualizar();
    //Se envían los datos de la navegación si el timer está listo
    if(ControlSerie.TimerListo()){
        ControlSerie.EnviarNavegacion(Kalman.GetPosicion(),Steppers.GetServo());
    }
    Steppers.Actualizar();
}

void serialEvent(){
    //Se cargan los caracteres recibidos en el buffer
```

```
    ControlSerie.LlenarBuffer();
}

CoordinadorEjes.h
#ifndef CoordinadorEjes_h
#define CoordinadorEjes_h

#include "Estructuras.h"
#include <Arduino.h>
#include <Servo.h>
#include <AccelStepper.h>

namespace NamespaceCoordinadorEjes{
    #define DriverOption 1
    #define ServoPin 2
    #define Enable12VRail 3
    #define Enable5VRail 4
    #define Step1 5
    #define Dir1 6
    #define Step2 7
    #define Dir2 8
    #define Step3 9
    #define Dir3 10

}
using namespace NamespaceCoordinadorEjes;

class CoordinadorEjes{
public:
    CoordinadorEjes();
    void Start();
    void ApagarRele();
    void EncenderRele();
    void SetVelocidadObjetivo(Velocidad_Motor Velocidad);
    Encoders GetDato();
    int GetServo();
    void SetEstadoRele(bool Estado);
    void Actualizar();

private:
    AccelStepper Stepper1;
    AccelStepper Stepper2;
    AccelStepper Stepper3;
    bool Estado_Rele=false;
    Encoders DatoEncoder;
```

```
Velocidad_Motor VelocidadObjetivo;
Velocidad_Motor VelocidadActual;
Servo Camara_Posicion;
int AngServo=90;
int DirServo=1;
long int AccelTime=1000;
long int Timer=0;
long int ServoTimer=0;
long int PeriodoServo=9223372000000;
};

#endif
```

## CoordinadorEjes.cpp

```
#include "CoordinadorEjes.h"
```

```
CoordinadorEjes::CoordinadorEjes():
    //Al crearse el coordinador de ejes se crean los objetos Accelstepper con los
    puertos correspondientes
```

```
    Stepper1(DriverOption,Step1,Dir1),
    Stepper2(DriverOption,Step2,Dir2),
    Stepper3(DriverOption,Step3,Dir3)
{}
```

```
void CoordinadorEjes::Start(){
    //Se configuran los puertos del rele y se inician apagados, además del servo
    pinMode(Enable12VRail1,OUTPUT);
    pinMode(Enable5VRail1,OUTPUT);
    digitalWrite(Enable12VRail1,HIGH);
    digitalWrite(Enable5VRail1,HIGH);
    this->Camara_Posicion.attach(ServoPin);
    //Se configura posicion y velocidad inicial, así como la velocidad máxima
    this->Stepper1.setCurrentPosition(0);
    this->Stepper2.setCurrentPosition(0);
    this->Stepper3.setCurrentPosition(0);
    this->Stepper1.setMaxSpeed(500);
    this->Stepper2.setMaxSpeed(500);
    this->Stepper3.setMaxSpeed(500);
    this->Stepper1.setSpeed(0);
    this->Stepper2.setSpeed(0);
    this->Stepper3.setSpeed(0);
}
```

```
void CoordinadorEjes::ApagarRele(){
    //Se apaga el rele y se colocan en 0 todas las velocidades
```

```
digitalWrite(Enable12VRail,HIGH);
digitalWrite(Enable5VRail,HIGH);
this->Stepper1.setSpeed(0);
this->Stepper2.setSpeed(0);
this->Stepper3.setSpeed(0);
this->VelocidadObjetivo.DS=0;
this->VelocidadObjetivo.W1=0;
this->VelocidadObjetivo.W2=0;
this->VelocidadObjetivo.W3=0;
this->VelocidadActual.DS=0;
this->VelocidadActual.W1=0;
this->VelocidadActual.W2=0;
this->VelocidadActual.W3=0;
this->Estado_Rele=false;
}

void CoordinadorEjes::EncenderRele(){
//Se enciende el rele y se da inicio a los timers
digitalWrite(Enable12VRail,LOW);
digitalWrite(Enable5VRail,LOW);
this->Estado_Rele=true;
this->ServoTimer=millis();
this->Timer=micros();
}

void CoordinadorEjes::SetVelocidadObjetivo(Velocidad_Motor Velocidad){
if(this->Estado_Rele){
//Se asignan las velocidades objetivo, solo si el rele se encuentra encendido
this->VelocidadObjetivo.W1=-Velocidad.W1;
this->VelocidadObjetivo.W2=-Velocidad.W2;
this->VelocidadObjetivo.W3=-Velocidad.W3;
this->VelocidadObjetivo.DS=Velocidad.DS;
//Se leen las velocidades del servo como grados por segundo y la direccion
this->DirServo=1;
if(this->VelocidadObjetivo.DS<0){
this->VelocidadObjetivo.DS=-this->VelocidadObjetivo.DS;
this->DirServo=-1;
}
if(this->VelocidadObjetivo.DS==0){
//Se asigna un periodo muy largo si la velocidad es 0
this->PeriodoServo=9223372000000;
}
else{
//La velocidad actual se establece como un timer en milisegundos
this->PeriodoServo=1000/this->VelocidadObjetivo.DS;
}
```

```
        }
    }
}

Encoders CoordinadorEjes::GetDatos(){
    //Retorna la posicion de los encoders en pasos
    this->DatoEncoder.Encoder1==this->Stepper1.currentPosition();
    this->DatoEncoder.Encoder2==this->Stepper2.currentPosition();
    this->DatoEncoder.Encoder3==this->Stepper3.currentPosition();
    return this->DatoEncoder;
}

int CoordinadorEjes::GetServo(){
    //Retorna el angulo del servo en grados
    return this->AngServo;
}

void CoordinadorEjes::SetEstadoRele(bool Estado_Rele){
    //Se lee la consigna del rele
    this->Estado_Rele=Estado_Rele;
    if(this->Estado_Rele){
        EncenderRele();
    }
    else{
        ApagarRele();
    }
}

void CoordinadorEjes::Actualizar(){
    if(this->Estado_Rele){
        if(micros()-this->Timer>this->AccelTime){
            //Se mira el timer de aceleracion y se aumenta o disminuye la velocidad de
            todos los motores
            if(this->VelocidadActual.W1<this->VelocidadObjetivo.W1){
                this->VelocidadActual.W1=this->VelocidadActual.W1+1;
            }
            if(this->VelocidadActual.W1>this->VelocidadObjetivo.W1){
                this->VelocidadActual.W1=this->VelocidadActual.W1-1;
            }

            if(this->VelocidadActual.W2<this->VelocidadObjetivo.W2){
                this->VelocidadActual.W2=this->VelocidadActual.W2+1;
            }
            if(this->VelocidadActual.W2>this->VelocidadObjetivo.W2){
                this->VelocidadActual.W2=this->VelocidadActual.W2-1;
            }
        }
    }
}
```

Comunicacion.h

```
#ifndef Comunicacion_h
#define Comunicacion_h

#include "Estructuras.h"
#include <Arduino.h>

class Comunicacion{
public:
```

```
Comunicacion();
void Iniciar();
bool MensajeListo();
void EnviarNavegacion(Coordenadas Navegacion, int Servo);
void LlenarBuffer();
void LeerMensaje();
bool GetEstadoRele();
Velocidad_Motor GetVelocidades();
float Decode(int Posicion);
bool EnvioActivo();
bool TimerListo();

private:
    char Buffer[64];
    int Puntero=0;
    bool EstadoMensaje=false;
    Velocidad_Motor Velocidades;
    bool Estado_Rele=false;
    unsigned long TimerComunicacion=0;
    unsigned long IntervaloComunicacion=1000;
};

#endif
```

### Comunicacion.cpp

```
#include "Comunicacion.h"

Comunicacion::Comunicacion(){
}

void Comunicacion::Iniciar(){
    //Se inicia el puerto serie
    Serial.begin(115200);
}

bool Comunicacion::MensajeListo(){
    //Se devuelve el flag que indica que el mensaje esta listo para leer
    return this->EstadoMensaje;
}

void Comunicacion::EnviarNavegacion(Coordenadas Navegacion, int Servo){
    if(this->Estado_Rele==false){
        //Se arma el mensaje para enviar si esta encendido el rele
        Serial.print("D;");
    }
}
```

```
else{
    //Se arma el mensaje para enviar si esta encendido el rele
    Serial.print("X");
    Serial.print(Navegacion.X);
    Serial.print("Y");
    Serial.print(Navegacion.Y);
    Serial.print("P");
    Serial.print(Navegacion.Psi);
    Serial.print("S");
    Serial.print(Servo);
    Serial.print("C");
    Serial.print(Navegacion.Temperatura);
    Serial.print(";");
}
}

void Comunicacion::LlenarBuffer(){
    //Se cargan los caracteres en el buffer hasta que encuentra el ';' y marca como
    //listo el mensaje
    while(Serial.available()>0){
        this->Buffer[this->Puntero]=Serial.read();
        this->Puntero=this->Puntero+1;
    }
    if(this->Buffer[this->Puntero-1]==';'){
        this->EstadoMensaje=true;
    }
}

void Comunicacion::LeerMensaje(){
    this->EstadoMensaje=false;
    this->Puntero=0;
    if(this->Buffer[Puntero]=='R' && this->Estado_Rele==true){
        //Se apaga el rele e ignorar el resto del mensaje
        this->Estado_Rele=false;
    }
    else if(this->Buffer[Puntero]=='R'){
        //Encender Rele
        this->Estado_Rele=true;
    }
    else if(this->Buffer[Puntero]=='W'){
        //Cambiar la velocidad de los steppers, expresada en ancho de pulso
        this->Puntero=Puntero+1;
        this->Velocidades.W1=Decode(Puntero);
        this->Puntero=Puntero+1;
        this->Velocidades.W2=Decode(Puntero);
```

```
    this->Puntero=Puntero+1;
    this->Velocidades.W3=Decode(Puntero);
    this->Puntero=Puntero+1;
    this->Velocidades.DS=Decode(Puntero);
}
else if(this->Buffer[Puntero]=='D'){
    //Cambiar la velocidad del servo, expresada en grados por segundo
    this->Estado_Rele=false;
}
this->Puntero=0;
}

bool Comunicacion::GetEstadoRele(){
    //retorna el estado deseado del rele
    return this->Estado_Rele;
}

Velocidad_Motor Comunicacion::GetVelocidades(){
    //retorna las consignas de velocidad
    return this->Velocidades;
}

float Comunicacion::Decode(int Posicion){
    //se decodifica el mensaje como entero con signo
    int Numero=0;
    int Signo=1;
    if(this->Buffer[Posicion]=='-'){
        Signo=-1;
        Posicion=Posicion+1;
    }
    while(Buffer[Posicion]!=',' && Buffer[Posicion]!=';'){
        Numero=Numero*10;
        Numero=Numero+(Buffer[Posicion]-'0');
        Posicion=Posicion+1;
    }
    this->Puntero=Posicion;
    return Numero*Signo;
}

bool Comunicacion::TimerListo(){
    //Se retorna el estado del temporizador
    bool Timer = false;
    if(millis() - this->TimerComunicacion >= this->IntervaloComunicacion){
        Timer = true;
        this->TimerComunicacion=millis();
```

```
    }
    return Timer;
}
```

## Navegacion.h

```
#ifndef Navegacion_h
#define Navegacion_h

#include "Estructuras.h"
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

class Navegacion{
public:
    Navegacion();
    void Iniciar();
    void CalcularDrift(int NumValores);
    Coordenadas GetPosicion();
    bool TimerListo();
    void LeerSensores();
    void SetDatoEncoder(Encoders);
    void GravedadInicial(int NumValores,int i);
    void CorreccionGravedad();
    void Kalman();

private:
    unsigned long TimerDatos;
    unsigned long IntervaloLectura;
    float dt;
    MPU6050_Data LecturaSensores;
    Coordenadas Posicion;
    Coordenadas PosicionAnterior;
    Coordenadas PosicionAbsoluta;
    Adafruit_MPU6050 mpu;
    Encoders DatoEncoder;
    Velocidad_Robot Velocidad;
    float PPR=400;
    float Phi0=0;
    float Tita0=0;
    float Tita;           //Asociado a wy
    float Phi;            //Asociado a wx
    float Psi;            //Asociado a wz
    float g=0.0;
```

```
float DriftX=0.0;
float DriftY=0.0;
float DriftZ=0.0;
float P[3]={1.0,1.0,1.0};
float Q[3]={0.00000324,0.00000324,0.00032527};
float R[3]={0.0,0.0,0.00001598};
};

#endif
```

## Navegacion.cpp

```
#include "Navegacion.h"
```

```
Navegacion::Navegacion(){
}
```

```
void Navegacion::Iniciar(){
    //Se inicia el modulo MPU6050
    mpu.begin();
    mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
    mpu.setGyroRange(MPU6050_RANGE_250_DEG);
    mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
    //Se calcula la deriva del modulo para eliminarla
    int NumValores=20;
    for(int i=0;i<NumValores;i++){
        CalcularDrift(NumValores);
    }
    //Se calcula el angulo para eliminar la gravedad del movimiento
    for(int i=0;i<NumValores;i++){
        LeerSensores();
        GravedadInicial(NumValores,i);
    }
    //Se inicia el timer
    TimerDatos=millis();
    IntervaloLectura=50;
}
```

```
void Navegacion::CalcularDrift(int NumValores){
    //Se calcula el drift para cada medicion y se divide por la cantidad de
    mediciones
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    DriftX=DriftX+g.gyro.x/NumValores;
    DriftY=DriftY+g.gyro.y/NumValores;
    DriftZ=DriftZ+g.gyro.z/NumValores;
```

```
}

Coordenadas Navegacion::GetPosition(){
    //Regresa la posicion en coordenadas globales
    return this->PosicionAbsoluta;
}

bool Navegacion::TimerListo(){
    //Retorna el estado del timer
    bool Timer = false;
    if(millis() - this->TimerDatos >= this->IntervaloLectura){
        Timer = true;
    }
    return Timer;
}

void Navegacion::LeerSensores(){
    //Se leen los sensores, incluyendo el intervalo de tiempo
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    this->LecturaSensores.ax=a.acceleration.x;
    this->LecturaSensores.ay=a.acceleration.y;
    this->LecturaSensores.az=a.acceleration.z;
    this->LecturaSensores.wx=g.gyro.x-DriftX;
    this->LecturaSensores.wy=g.gyro.y-DriftY;
    this->LecturaSensores.wz=g.gyro.z-DriftZ;
    this->LecturaSensores.C=temp.temperature;
    dt=(millis() - this->TimerDatos)/1000.0;
    TimerDatos=millis();
}

void Navegacion::SetDatoEncoder(Encoders Datos){
    //Se cargan los datos del encoder
    this->DatoEncoder=Datos;
}

void Navegacion::GravedadInicial(int NumValores, int i){
    //Se leen los angulos iniciales a partir de la gravedad
    i=i+1;
    float
    g=sqrt(LecturaSensores.ax*LecturaSensores.ax+LecturaSensores.ay*LecturaSensores.ay+
            +LecturaSensores.az*LecturaSensores.az);
    this->Tita0=Tita0+asin(this->LecturaSensores.ay/g)/NumValores;
    this->Phi0=Phi0+asin(-this->LecturaSensores.ax/(g*cos(Tita0*NumValores/i)))/NumValores;
```

```

this->g=this->g+g/NumValores;
if(i==NumValores){
    Tita=Tita0;
    Phi=Phi0;
    Psi=0;
}
}

void Navegacion::CorreccionGravedad(){
//Se mide y elimina la fuerza de la gravedad de la medición de los sensores
float grados=PI/180.0;
Tita=Tita+(dt)*LecturaSensores.wx;
Phi=Phi+(dt)*LecturaSensores.wy;
Psi=Psi+(dt)*LecturaSensores.wz;
float snPsi=sin(Psi);
float csPsi=cos(Psi);
float snPhi=sin(Phi);
float csPhi=cos(Phi);
float snTita=sin(Tita);
float csTita=cos(Tita);
LecturaSensores.ax=LecturaSensores.ax+g*(snPsi*snTita+csPsi*csTita*snPhi);
LecturaSensores/ay=LecturaSensores/ay+g*(csTita*snPhi*snPsi-csPsi*snTita);
LecturaSensores.az=LecturaSensores.az-g*(csPhi*csTita);
}

void Navegacion::Kalman(){

PosicionAnterior=Posicion;
//Se realiza la predicción en base a los encoders y en base a las velocidades
float Y1=PI*(sqrt(3.0)/3.0*(-
DatoEncoder.Encoder2+DatoEncoder.Encoder3))*0.058/PPR;
float Y2=PI*(1.0/3.0*(-
2.0*DatoEncoder.Encoder1+DatoEncoder.Encoder2+DatoEncoder.Encoder3))*0.058/PPR;
float
Y3=PI*((DatoEncoder.Encoder1+DatoEncoder.Encoder2+DatoEncoder.Encoder3)*0.058/(3*
0.112))/PPR;

Velocidad.VelocidadX=Velocidad.VelocidadX+dt*LecturaSensores.ax;
Velocidad.VelocidadY=Velocidad.VelocidadY+dt*LecturaSensores.ay;
Velocidad.VelocidadWZ=LecturaSensores.wx;

float xp1=PosicionAnterior.X+dt*Velocidad.VelocidadX;
float xp2=PosicionAnterior.Y+dt*Velocidad.VelocidadY;
float xp3=PosicionAnterior.Psi+dt*Velocidad.VelocidadWZ;

```

```

//Se carga la matriz de covarianza inicial para el punto actual
P[0]=P[0]+Q[0];
P[1]=P[1]+Q[1];
P[2]=P[2]+Q[2];

//Se estima la matriz de Kalman
float K[3];
K[0]=P[0]/(P[0]+R[0]);
K[1]=P[1]/(P[1]+R[1]);
K[2]=P[2]/(P[2]+R[2]);

//Se realiza la corrección
Posicion.X=xp1+K[0]*(Y1-xp1);
Posicion.Y=xp2+K[1]*(Y2-xp2);
Posicion.Psi=xp3+K[2]*(Y3-xp3);

this->Psi=Posicion.Psi;

//Se convierte de coordenadas locales a absolutas
float DeltaX=Posicion.X-PosicionAnterior.X;
float DeltaY=Posicion.Y-PosicionAnterior.Y;

PosicionAbsoluta.X=PosicionAbsoluta.X+DeltaX*cos(Psi)-DeltaY*sin(Psi);
PosicionAbsoluta.Y=PosicionAbsoluta.Y+DeltaX*sin(Psi)+DeltaY*cos(Psi);
PosicionAbsoluta.Psi=Posicion.Psi*180.0/PI;
PosicionAbsoluta.Temperatura=LecturaSensores.C;

//Se carga la matriz de covarianza corregida
P[0]=(1-K[0])*P[0];
P[1]=(1-K[1])*P[1];
P[2]=(1-K[2])*P[2];
}

```

## Estructuras.h

```

#ifndef Estructuras_h
#define Estructuras_h

struct Coordenadas {
    float X=0;
    float Y=0;
    float Psi=0;
    float Temperatura=0;
};

struct Velocidad_Robot{

```

```
int VelocidadX=0;
int VelocidadY=0;
int VelocidadWZ=0;
int VelocidadServo=0;
};

struct MPU6050_Data{
    float ax=0;
    float ay=0;
    float az=0;
    float wx=0;
    float wy=0;
    float wz=0;
    float C=0;
};

struct Encoders{
    long Encoder1=0;
    long Encoder2=0;
    long Encoder3=0;
};

struct Velocidad_Motor{
    int W1=0;
    int W2=0;
    int W3=0;
    int DS=0;
};

#endif
```

## Anexo: Código Raspberry (Python)

### Proyecto.py

```
from Comunicacion import Comunicacion
from Tensorflow import Tensorflow

#from Tensorflow import Tensorflow

import time

import cv2
from picamera2 import Picamera2
#Se inicializan todos los objetos
net = Comunicacion()
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": 'RGB888',
"size": (640, 480)}))
picam2.start()
TF=Tensorflow()

while True:
    #Se capture la imagen
    Imagen = cv2.rotate(picam2.capture_array(),cv2.ROTATE_180)
    #Si se recibio la orden se cambia el estado del tensorflow
    if net.GetToogleTensorflow():
        TF.ToogleTensorflow()
    #Si la medicion esta lista se carga la imagen para realizar la nueva medicion
    if TF.IsReady():
        TF.SetImage(Imagen)
    #Se procesa la imagen capturada
    fps,imagen=TF.CV2ProcessImage(Imagen)
    #Se carga la imagen para el envio
    net.SetImagen(imagen,fps);
```

### Comunicacion.py

```
import socket
from threading import Thread
from threading import Lock
import serial
import serial.tools.list_ports
import time
import cv2

class Comunicacion:
    def __init__(self):
```

```
    self.DisponibleImagen=False
    self.ToogleTensorflow=False
    self.Rеле=False
    #Se realiza la configuracion y conexion de los servidores
    for port in serial.tools.list_ports.comports():
        if str(port.manufacturer).startswith("Arduino"):
            ArduinoPort= "/dev/" + str(port.name)
            self.ArduinoSerial = serial.Serial(ArduinoPort , 115200)
    self.ip="10.42.0.1" # "10.42.0.1" para la red de raspberry, ifconfig para
cualquier otra
        self.PortControl=40000
        self.PortImagen=40001
        self.PortNavegacion=40002
        self.ServerControl=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.ServerImagen=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.ServerNavegacion=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.ServerControl.bind((self.ip, self.PortControl))
        self.ServerImagen.bind((self.ip, self.PortImagen))
        self.ServerNavegacion.bind((self.ip, self.PortNavegacion))
    #Se preparan los locks y threads
        self.LockImagen=Lock()
        self.LockSerial=Lock()

        self.ThreadControl= Thread(target=self.ComunicacionControl)
        self.ThreadImagen= Thread(target=self.ComunicacionImagen)
        self.ThreadNavegacion= Thread(target=self.ComunicacionNavegacion)
        self.ServerControl.listen(1)
        self.ServerImagen.listen(1)
        self.ServerNavegacion.listen(1)
        self.ThreadControl.start()
        self.ThreadImagen.start()
        self.ThreadNavegacion.start()
        time.sleep(3)

def ComunicacionControl(self):
    Buffer=str()
    while True:
        #Se aceptan conexiones
        ClienteControl, _ = self.ServerControl.accept()
        print("Control")
        self.Connected=True
        try:
            while self.Connected:
                Data=ClienteControl.recv(100) #retorna 0 si se cierra el
cliente, retorna un vector de bytes vacio si no hay nada
```

```

        if Data:
            if not Data == "":
                Buffer=Buffer + (Data.decode('UTF-8'))
                if Buffer[len(Buffer)-1] == ';':
                    #Se ve que tipo de mensaje se ha recibido y se lo
                    pasa al arduino o al objeto Tensorflow
                    if Buffer[0]==‘R’ or Buffer[0]==‘W’:
                        self.LockSerial.acquire()
                        self.ArduinoSerial.write(Buffer.encode('utf-
                            8'))
                        if Buffer[0]==‘R’:
                            self.Rеле=not self.Rеле
                            self.LockSerial.release()
                        elif Buffer[0]==‘T’:
                            self.ToogleTensorflow=True
                            Buffer= “”
                    else:
                        break
            except:
                pass
            #Se realiza la desconexion y se le da la orden de apagar el rele al
            arduino
            self.Connected=False
            if self.Rеле:
                Buffer=“R;”
                self.Rеле=False
            self.ArduinoSerial.write(Buffer.encode('utf-8'))
            Buffer= “”
            ClienteControl.close()
            print("Desconectado:Control")

def ComunicacionImagen(self):
    while True:
        #Se aceptan conexiones
        ClienteImagen, _ = self.ServerImagen.accept()
        print("Imagen")
        self.Connected=True
        try:
            while self.Connected:
                if self.DisponibleImagen:
                    #Se envia la imagen y ‘;’ para indicar el fin del mensaje
                    self.LockImagen.acquire()
                    _,ImagenPNG=cv2.imencode('.png',self.Imagen)
                    self.DisponibleImagen=False
                    self.LockImagen.release()

```

```
        Send=ImagenPNG.tobytes()+';'.encode('UTF-8')
        ClienteImagen.sendall(Send)
    except:
        pass
    #Se realiza la desconexion
    self.Connected=False
    ClienteImagen.close()
    self.DisponibleImagen=False
    self.Imagen=""
    print("Desconectado:Imagen")

def ComunicacionNavegacion(self):
    Buffer=str()
    while True:
        #Se aceptan conexiones
        ClienteNavegacion, _ = self.ServerNavegacion.accept()
        print("Navegacion")
        self.Connected=True
        try:
            while self.Connected:
                if (self.ArduinoSerial.in_waiting > 0):
                    #Se recibe el mensaje del arduino hasta detectar el ';'
                    self.LockSerial.acquire()
                    Buffer=Buffer+self.ArduinoSerial.read(self.ArduinoSerial.
in_waiting).decode('utf-8')
                    self.LockSerial.release()
                    if Buffer[len(Buffer)-1] == ';':
                        #Se agrega el mensaje del Tensorflow de los FPS y se
envia
                        Buffer=Buffer.rstrip(Buffer[len(Buffer)-1])
                        self.LockImagen.acquire()
                        Buffer=Buffer+self.FPS
                        self.LockImagen.release()
                        Send=Buffer.encode('UTF-8')
                        ClienteNavegacion.sendall(Send)
                        Buffer=""
                except:
                    pass
        #Se realiza la desconexion
        self.Connected=False
        Buffer=""
        self.ArduinoSerial.read_all()
        ClienteNavegacion.close()
        print("Desconectado:Navegacion")
```

```

def SetImagen(self,Imagen,fps):
    #Se carga la imagen y el dato de los FPS para enviar
    self.LockImagen.acquire()
    self.DisponibleImagen=True
    self.Imagen=Imagen
    self.FPS=fps
    self.LockImagen.release()

def GetToogleTensorflow(self):
    #Se lee si ha habido orden de conmutar el Tensorlow
    if self.ToogleTensorflow:
        self.ToogleTensorflow=False
        return True
    else:
        return False

```

## Tensorflow.py

```

import os
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util
from tflite_runtime.interpreter import Interpreter
from threading import Thread
from threading import Lock

class Tensorflow:
    def __init__(self):
        self.min_conf_threshold = 0.20
        self.imW=640
        self.imH=480

        #Se carga el mapa de etiquetas
        with open("/home/maxiwarhammer/Documents/ProyectoFinal/labelmap.txt",
'r') as labelmap:
            self.labels = [line.strip() for line in labelmap.readlines()]

        #Se carga el modelo de Tensorflow
        self.interpreter =
Interpreter(model_path="/home/maxiwarhammer/Documents/ProyectoFinal/detect.tflite
")

```

```
    self.interpreter.allocate_tensors()

    #Se leen los detalles del modelo
    self.input_details = self.interpreter.get_input_details()
    self.output_details = self.interpreter.get_output_details()
    self.height = self.input_details[0]['shape'][1]
    self.width = self.input_details[0]['shape'][2]

    self.input_mean = 127.5
    self.input_std = 127.5

    #Se configura el modelo como TF2
    self.boxes_idx, self.classes_idx, self.scores_idx = 1, 3, 0

    #Se inicia el calculo de FrameRate (o FPS)
    self.TFframe_rate_calc = 1
    self.freq = cv2.getTickFrequency()
    self.Ready=True;
    self.TensorflowActive=True

    #Se activan los threads y locks
    self.LockOutput=Lock()
    self.LockImage=Lock()
    self.ThreadDetect= Thread(target=self.TFDetect)
    self.ThreadDetect.start()

def TFDetect(self):
    while True:
        while not self.Ready and self.TensorflowActive:
            #Se toma el tiempo inicial del frame
            t1 = cv2.getTickCount()

            #Se obtiene la imagen
            self.LockImage.acquire()
            frame=self.Imagen
            self.LockImage.release()
            frame_resized = cv2.resize(frame, (self.width, self.height))
            input_data = np.expand_dims(frame_resized, axis=0)

            #Se normalizan los pixeles
            input_data = (np.float32(input_data) - self.input_mean) /
self.input_std

            #Se realiza la deteccion
```

```

        self.interpreter.set_tensor(self.input_details[0]['index'],input_
data)
        self.interpreter.invoke()
#Se guardan los resultados
        self.LockOutput.acquire()
        self.boxes =
self.interpreter.get_tensor(self.output_details[self.boxes_idx]['index'])[0] #
Bounding box coordinates of detected objects
        self.classes =
self.interpreter.get_tensor(self.output_details[self.classes_idx]['index'])[0] #
Class index of detected objects
        self.scores =
self.interpreter.get_tensor(self.output_details[self.scores_idx]['index'])[0] #
Confidence of detected objects
        self.TFframe_rate_calc= self.freq/(cv2.getTickCount()-t1)#
Calculate framerate
        self.LockOutput.release()
        self.Ready=True;

def CV2ProcessImage(self,imagen):
#Se leen los resultados
        self.LockOutput.acquire()
try:
    Boxes=self.boxes
    Classes=self.classes
    Scores=self.scores
    frame_rate=self.TFframe_rate_calc
except:
    Boxes={}
    Clases={}
    Scores={}
    frame_rate=0
        self.LockOutput.release()

#Se revisan todas las detecciones y se dibuja el recuadro si cumple con
el minimo de confianza
for i in range(len(Scores)):
    if ((Scores[i] > self.min_conf_threshold) and (Scores[i] <= 1.0)):

        #Se toman las coordenadas y se dibuja la caja
        #Se corrige si las cajas tienen limites fuera de la imagen, suele
ocurrir si una deteccion se realiza cerca del borde
        ymin = int(max(1,(Boxes[i][0] * self.imH)))
        xmin = int(max(1,(Boxes[i][1] * self.imW)))
        ymax = int(min(self.imH,(Boxes[i][2] * self.imH)))

```

```
        xmax = int(min(self.imW,(Boxes[i][3] * self.imW)))

        cv2.rectangle(imagen, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

        #Se escriben las etiquetas
        #Se busca la etiqueta usando el indice de la etiqueta
        object_name = self.labels[int(Classes[i])]
        #Se agrega el puntaje porcentual de detección
        label = '%s: %d%%' % (object_name, int(Scores[i]*100))
        #Se configura la fuente
        labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
        #Se asegura que no se esten escribiendo las etiquetas muy cerca
del borde
        label_ymin = max(ymin, labelSize[1] + 10)
        #Se dibuja el recuadro blanco donde iran las detecciones
        cv2.rectangle(imagen, (xmin, label_ymin-labelSize[1]-10),
(xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED)
        #Se escribe el texto de la etiqueta
        cv2.putText(imagen, label, (xmin, label_ymin-7),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)
        #Se agrega el mensaje de los FPS y se retorna la imagen procesada
if not self.TensorflowActive:
    fps="FN;"
else:
    fps= "F" + '{:.2f}'.format(frame_rate) + ";"
return fps,imagen

def IsReady(self):
    #Retorna si se ha terminado la detección anterior
    return self.Ready;
def SetImage(self,imagen):
    #Se carga la imagen y se prepara la siguiente detección
    self.LockImage.acquire()
    self.Imagen=imagen
    self.Ready=False
    self.LockImage.release()
def ToogleTensorflow(self):
    #Se activa o desactivan las detecciones
    self.TensorflowActive=not self.TensorflowActive
```

## Anexo: Códigos de inicio Raspberry

### ProyectoFinal.sh

```
#!/bin/bash
```

```
cd /home/maxiwarhammer/Documents/ProyectoFinal &&
source ProyectoFinalEnv/bin/activate &&
python3 Proyecto.py
```

### ProyectoFinal.desktop

```
[Desktop Entry]
Version=1.0
Name=ProyectoFinal
Comment=Mi proyecto final de estudios
Exec=lxterminal -e /home/maxiwarhammer/Documents/ProyectoFinal/ProyectoFinal.sh
Type=Application
Terminal=false
```

## Anexo: Código Terminal (Unity C#)

### Minimap.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Minimap : MonoBehaviour
{
    // Start is called before the first frame update
    private Camera MinimapCam;
    private GameObject Robot;
    bool ColocarSenal = false;
    bool SenalElegida = false;
    int ObjetoSeleccionado = 0;
    public string Texto = "";
    Vector3 SenalPosicion, DefaultPosicion;
    public GameObject Biologico, Corrosivo, Inflamable, Radioactivo, Toxic;
    GameObject[] ObjetosLista;
    Quaternion Rotacion;
    void Start()
    {
        ObjetosLista = new GameObject[5];

        Rotacion = Quaternion.identity;
        SenalPosicion = new Vector3(0, 0, 0);
        DefaultPosicion = new Vector3(0, -9000, 0);
        MinimapCam = GameObject.Find("MapCamera").GetComponent<Camera>();
        Robot = GameObject.Find("Robot");
        IniciarLista();
    }

    // Update is called once per frame
    void Update()
    {
        MinimapCam.transform.position = new Vector3(Robot.transform.position.x,
        Robot.transform.position.y + 10f, Robot.transform.position.z);
        LeerBotones();
        if (ColocarSenal && !SenalElegida)
        {
            ElegirSenal();
        }
        if (ColocarSenal && SenalElegida)
        {
            MoverSenal();
        }
    }

    void IniciarLista()
    {
        //Se prepara la lista de objetos a clonar para ubicarlos
        ObjetosLista[0] = Instantiate(Biologico, DefaultPosicion, Rotacion);
        ObjetosLista[0].name = "Biologico";
        ObjetosLista[1] = Instantiate(Corrosivo, DefaultPosicion, Rotacion);
        ObjetosLista[1].name = "Corrosivo";
        ObjetosLista[2] = Instantiate(Inflamable, DefaultPosicion, Rotacion);
    }
}

```

```

        ObjetosLista[2].name = "Inflamable";
        ObjetosLista[3] = Instantiate(Radioactivo, DefaultPosicion, Rotacion);
        ObjetosLista[3].name = "Radioactivo";
        ObjetosLista[4] = Instantiate(Toxico, DefaultPosicion, Rotacion);
        ObjetosLista[4].name = "Toxico";
    }
    void LeerBotones()
    {
        //Máquina de estados básica para definir que función se realiza
        float LT = GameObject.Find("Lector_USB").GetComponent<Lector_USB>().LT;
        float PrevLT = GameObject.Find("Lector_USB").GetComponent<Lector_USB>().PrevLT;
        float LB = GameObject.Find("Lector_USB").GetComponent<Lector_USB>().LB;
        float PrevLB = GameObject.Find("Lector_USB").GetComponent<Lector_USB>().PrevLB;
        if(LT >= 0.5 && PrevLT < 0.5)
        {

            if (ColocarSenal && !SenalElegida)
            {
                Texto = "Posicionar";
                SenalElegida = true;
            }
            else if (!ColocarSenal && !SenalElegida)
            {
                IniciarColocacion();
                Texto = "Seleccionar";
                ColocarSenal = true;
            }
            else
            {
                Texto = "";
                FijarSenal();
                ColocarSenal = false;
                SenalElegida = false;
            }
        }
        if(LB == 1 && PrevLB == 0)
        {
            Cancelar();
            Texto = "";
            ColocarSenal = false;
            SenalElegida = false;
        }
    }

    void IniciarColocacion()
    {
        //Inicia la colocación de una señal
        SenalPosicion = Robot.transform.position;
        SenalPosicion.y = SenalPosicion.y + 0.1f;
        ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
        SenalPosicion;
    }

    void ElegirSenal()
    {
        //Se elige la señal, evitando órdenes contradictorias
        Vector2 Dpad = GameObject.Find("Lector_USB").GetComponent<Lector_USB>().Dpad;
    }
}

```

```

        Vector2 PrevDpad =
GameObject.Find("Lector_USB").GetComponent<Lector_USB>().PrevDpad;
    if ((Dpad.x*Dpad.y)>=0 && PrevDpad == Vector2.zero)
    {
        if (Dpad.x > 0 || Dpad.y > 0)
        {
            ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
DefaultPosicion;
            ObjetoSeleccionado = ObjetoSeleccionado + 1;
            RevisarIndice();
            IniciarColocacion();
        }
        if (Dpad.x < 0 || Dpad.y < 0)
        {
            ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
DefaultPosicion;
            ObjetoSeleccionado = ObjetoSeleccionado - 1;
            RevisarIndice();
            IniciarColocacion();
        }
    }
}

void RevisarIndice()
{
    //Si el índice se sale de la matriz se acomoda antes de accederse al array
    if (ObjetoSeleccionado == -1)
    {
        ObjetoSeleccionado = 4;
    }
    else if (ObjetoSeleccionado == 5)
    {
        ObjetoSeleccionado = 0;
    }
}

void MoverSenal()
{
    Vector2 Dpad = GameObject.Find("Lector_USB").GetComponent<Lector_USB>().Dpad;
    if (Dpad.x > 0)
    {
        SenalPosicion.x = SenalPosicion.x + 0.1f;
        ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
SenalPosicion;
    }
    if (Dpad.x < 0)
    {
        SenalPosicion.x = SenalPosicion.x - 0.1f;
        ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
SenalPosicion;
    }
    if (Dpad.y > 0)
    {
        SenalPosicion.z = SenalPosicion.z + 0.1f;
        ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
SenalPosicion;
    }
    if (Dpad.y < 0)
    {
}
}

```

```

        {
            SenalPosicion.z = SenalPosicion.z - 0.1f;
            ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
SenalPosicion;
        }

    }

    void FijarSenal()
{
    //Se fija la señal clonada debajo del plano del robot y se esconde la señal
original
    SenalPosicion.y = SenalPosicion.y - 0.2f;
    Instantiate(ObjetosLista[ObjetoSeleccionado], SenalPosicion, Rotacion);
    ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
DefaultPosicion;
}

void Cancelar()
{
    //Devuelve las señales a su posición oculta
    ObjetosLista[ObjetoSeleccionado].GetComponent<Transform>().position =
DefaultPosicion;
}

}

```

**LineDraw.cs**

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LineDraw : MonoBehaviour
{

    LineRenderer Linea;
    string Navegacion;
    Vector3 Posicion;
    Vector3 PosicionPrev;
    float Rotacion=0f;
    float Escala = 1f;
    // Start is called before the first frame update
    void Start()
    {
        Linea = GameObject.Find("Trayectoria").GetComponent<LineRenderer>();
        Linea.startWidth = 0.1f;
        Linea.endWidth = 0.1f;
        Linea.startColor = new Color(255, 0, 0);
        Linea.endColor = new Color(255, 0, 0);
        Posicion.x = 0;
        Posicion.y = 0;
        Posicion.z = 0;
        PosicionPrev.x = 0;
        PosicionPrev.y = 0;
        PosicionPrev.z = 0;
    }
}

```

```

// Update is called once per frame
void Update()
{
    Navegacion =
GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().NavegacionDatos;
    LeerNavegacion();
    float Modulo = (Posicion - PosicionPrev).magnitude;
    if (Modulo >= 0.01)
    {
        Linea.positionCount = Linea.positionCount + 1;
        Linea.SetPosition(Linea.positionCount - 1, Posicion);
        PosicionPrev = Posicion;
    }
    else
    {
        Posicion = PosicionPrev;
    }
    GameObject.Find("Robot").GetComponent<Transform>().position = Posicion;
    GameObject.Find("Robot").GetComponent<Transform>().eulerAngles = new
Vector3(90.0f, Rotacion, 0.0f);
}

void LeerNavegacion()
{
    //Se leen los datos de navegación para rotar el objeto robot y dibujar la línea
    string X, Z, R;
    if (!(Navegacion[0] == 'D' || Navegacion[0] == 'E'))
    {
        X = Navegacion.Substring(Navegacion.IndexOf("X") + 1, Navegacion.IndexOf("Y")
- Navegacion.IndexOf("X") - 1);
        Z = Navegacion.Substring(Navegacion.IndexOf("Y") + 1, Navegacion.IndexOf("P")
- Navegacion.IndexOf("Y") - 1);
        R = Navegacion.Substring(Navegacion.IndexOf("P") + 1, Navegacion.IndexOf("S")
- Navegacion.IndexOf("P") - 1);
        Posicion.x =
Escala*float.Parse(X,System.Globalization.CultureInfo.InvariantCulture);
        Posicion.z = Escala*float.Parse(Z,
System.Globalization.CultureInfo.InvariantCulture);
        Rotacion = -float.Parse(R,
System.Globalization.CultureInfo.InvariantCulture)+90.0f;
    }
}
}

```

## Interfaz.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Interfaz : MonoBehaviour
{
    public Texture2D RaspiTexture = null;
    public Text StatusText,FPSText,PosicionText,ColocacionText;
    float FPSPrev = 0.0f;
    bool TFActive=false;

```

```
string NavegacionText;
public string UI;
// Start is called before the first frame update
void Start()
{
    UI = "Base";
    StatusText = GameObject.Find("Status").GetComponent<Text>();
    FPSText = GameObject.Find("FPS").GetComponent<Text>();
    PosicionText = GameObject.Find("Posicion").GetComponent<Text>();
    ColocacionText = GameObject.Find("Colocacion").GetComponent<Text>();
    if (RaspiTexture == null)
    {
        RaspiTexture = new Texture2D(2, 2);
    }
}

// Update is called once per frame
void Update()
{

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexNavegacion.WaitOne();
    NavegacionText =
GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().NavegacionOnDatos;

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexNavegacion.ReleaseMutex();
    SelectUI();
    if (UI == "Base")
    {
        UpdateImage();
        UpdateStatusText();
        UpdateFPSText();
        UpdatePosicionText();
        UpdateColocacion();
    }
    if (UI == "Camara")
    {
        UpdateImage();
    }
}

void UpdateImage()
{
    //Se leen los bytes de la imagen y se cargan en una textura
    if
(GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().ImageIsReady())
    {
        Texture2D Texture = new Texture2D(2, 2);

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexImagen.WaitOne();

Texture.LoadImage(GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().BytesImagen);
```

```
GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexImagegen.ReleaseMutex();
    if (Texture.height > 8)          //La textura de error es de 8X8
    {
        Texture.Apply();
        RaspiTexture = Texture;
    }
    if (UI == "Camara")
    {
        GameObject.Find("RaspiCamara2").GetComponent<RawImage>().texture =
RaspiTexture;
    }
    if (UI == "Base")
    {
        GameObject.Find("RaspiCamara1").GetComponent<RawImage>().texture =
RaspiTexture;
    }
    else
        if (UI == "Camara")
    {
        GameObject.Find("RaspiCamara2").GetComponent<RawImage>().texture =
RaspiTexture;
    }
    if (UI == "Base")
    {
        GameObject.Find("RaspiCamara1").GetComponent<RawImage>().texture =
RaspiTexture;
    }
}
void UpdateStatusText()
{
    bool Connected =
GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().IsConnected;
    if (Connected)
    {
        StatusText.text = "Conexión: ON" + "\n";
        if (NavegacionText[0] == 'D' || NavegacionText[0] == 'E')
        {
            StatusText.text = StatusText.text + "Relé: OFF" + "\n";
        }
        else
        {
            StatusText.text = StatusText.text + "Relé: ON" + "\n";
        }
        if (!(NavegacionText[0] == 'E'))
        {
            if (NavegacionText.Substring(NavegacionText.IndexOf("F") + 1, 1) == "N")
            {
                StatusText.text = StatusText.text + "Tensorflow: OFF";
            }
            else
            {
                StatusText.text = StatusText.text + "Tensorflow: ON";
            }
        }
    }
}
```

```

        }
    else
    {
        StatusText.text = StatusText.text + "Tensorflow: OFF";
    }
}
else
{
    StatusText.text= "Conexión: OFF" + "\n";
}
}

void UpdateFPSText()
{
    string FPS;
    if (!(NavegacionText[0] == 'E'))
    {
        if(!((NavegacionText.IndexOf(";") - NavegacionText.IndexOf("F") - 1)<0)){
            FPS = NavegacionText.Substring(NavegacionText.IndexOf("F") + 1,
NavegacionText.IndexOf(";") - NavegacionText.IndexOf("F") - 1);
            if (FPS == "N")
            {
                FPS = "_._";
            }
            else
            {
                FPS = "._.";
            }
        }
        else
        {
            FPS = "._.";
        }
        FPSText.text = "FPS: " + FPS + "\n";
        string Temp;
        if (NavegacionText[0] == 'D' || NavegacionText[0] == 'E')
        {
            Temp = "._.";
        }
        else
        {
            Temp = NavegacionText.Substring(NavegacionText.IndexOf("C") + 1,
NavegacionText.IndexOf("F") - NavegacionText.IndexOf("C")-1);
        }
        FPSText.text = FPSText.text + "Temp: " + Temp + "\n";
    }
}

void UpdatePosicionText()
{
    string X, Y, R, S;
    if (NavegacionText[0] == 'D' || NavegacionText[0] == 'E')
    {
        X = "_____•__";
        Y = "_____•__";
        R = "_____•__";
        S = "_____•__";
    }
}

```

```

        }
        else
        {
            X = NavegacionText.Substring(NavegacionText.IndexOf("X") + 1,
NavegacionText.IndexOf("Y") - NavegacionText.IndexOf("X") - 1);
            Y = NavegacionText.Substring(NavegacionText.IndexOf("Y") + 1,
NavegacionText.IndexOf("P") - NavegacionText.IndexOf("Y") - 1);
            R = NavegacionText.Substring(NavegacionText.IndexOf("P") + 1,
NavegacionText.IndexOf("S") - NavegacionText.IndexOf("P") - 1);
            S = NavegacionText.Substring(NavegacionText.IndexOf("S") + 1,
NavegacionText.IndexOf("C") - NavegacionText.IndexOf("S") - 1);
        }
        PosicionText.text = "X: " + X + "\n";
        PosicionText.text = PosicionText.text + "Y: " + Y + "\n";
        PosicionText.text = PosicionText.text + "R: " + R + "\n";
        PosicionText.text = PosicionText.text + "S: " + S;
    }

void SelectUI()
{
    if (UI=="Base")
    {
        GameObject.Find("InterfazBase").GetComponent<Canvas>().enabled = true;
        GameObject.Find("InterfazCamara").GetComponent<Canvas>().enabled = false;
        GameObject.Find("InterfazMapa").GetComponent<Canvas>().enabled = false;
    }
    if (UI=="Camara")
    {
        GameObject.Find("InterfazBase").GetComponent<Canvas>().enabled = false;
        GameObject.Find("InterfazCamara").GetComponent<Canvas>().enabled = true;
        GameObject.Find("InterfazMapa").GetComponent<Canvas>().enabled = false;
    }
    if (UI=="Mapa")
    {
        GameObject.Find("InterfazBase").GetComponent<Canvas>().enabled = false;
        GameObject.Find("InterfazCamara").GetComponent<Canvas>().enabled = false;
        GameObject.Find("InterfazMapa").GetComponent<Canvas>().enabled = true;
    }
}
void UpdateColocacion()
{
    ColocacionText.text = GameObject.Find("MapCamera").GetComponent<Minimap>().Texto;
}
}

```

### Lector\_USB.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO.Ports;
using UnityEngine.InputSystem;
using UnityEngine.UI;

public class Lector_USB : MonoBehaviour
{
    Controles Control;

```

```
private string Motores;
public string ComunicaciónMensaje;
int Stepper1;
int Stepper2;
int Stepper3;
int Servo;
bool ReleLeido;
bool TensorflowLeido;

public Vector2 IStick, DStick;
public float A;
public float Y;
public float LT;
public float LB;
public float RT;
public float buttonStart;
public float Select;
public Vector2 Dpad;
public float PrevA = 0;
public float PrevY=0;
public float PrevLT = 0;
public float PrevLB = 0;
public float PrevRT = 0;
public float PrevbuttonStart = 0;
public Vector2 PrevDpad;
public int Interfaz;

// Start is called before the first frame update
void Start()
{
    ReleLeido = true;
    TensorflowLeido = true;
    Control = new Controles();
    //Se prepara la lectura del control y sus variables
    Control.Deteccion.Enable();
    Control.Deteccion.LeftAnalogStick.performed += Contexto => IStick =
    Contexto.ReadValue<Vector2>();
    Control.Deteccion.LeftAnalogStick.canceled += Contexto => IStick = Vector2.zero;
    Control.Deteccion.RightAnalogStick.performed += Contexto => DStick =
    Contexto.ReadValue<Vector2>();
    Control.Deteccion.RightAnalogStick.canceled += Contexto => DStick = Vector2.zero;
    Control.Deteccion.A.performed += Contexto => A = Contexto.ReadValue<float>();
    Control.Deteccion.A.canceled += Contexto => A = 0f;
    Control.Deteccion.Y.performed += Contexto => Y = Contexto.ReadValue<float>();
    Control.Deteccion.Y.canceled += Contexto => Y = 0f;
    Control.Deteccion.LT.performed += Contexto => LT = Contexto.ReadValue<float>();
    Control.Deteccion.LT.canceled += Contexto => LT = 0f;
    Control.Deteccion.LB.performed += Contexto => LB = Contexto.ReadValue<float>();
    Control.Deteccion.LB.canceled += Contexto => LB = 0f;
    Control.Deteccion.RT.performed += Contexto => RT = Contexto.ReadValue<float>();
    Control.Deteccion.RT.canceled += Contexto => RT = 0f;
    Control.Deteccion.Select.performed += Contexto => Select =
    Contexto.ReadValue<float>();
    Control.Deteccion.Select.canceled += Contexto => Select = 0f;
    Control.Deteccion.Start.performed += Contexto => buttonStart =
    Contexto.ReadValue<float>();
    Control.Deteccion.Start.canceled += Contexto => buttonStart = 0f;
```

```

        Control.Deteccion.Dpad.performed += Contexto => Dpad =
Contexto.ReadValue<Vector2>();
        Control.Deteccion.Dpad.canceled += Contexto => Dpad = Vector2.zero;
    }

    // Update is called once per frame
    void Update()
    {
        CrearMensaje();
        if (A == 1 && PrevA==0 || RT >= 0.5 && PrevRT < 0.5)
        {
            CambiarInterfaz();
        }
        if (Select == 1)
        {
            Quit();
        }
        UpdateButtons();
    }

    void StepperCalculate()
    {
        float Motor1 = IStick.x - DStick.x;
        float Motor2 = -IStick.y - IStick.x/2 - DStick.x;
        float Motor3 = IStick.y - IStick.x/2 - DStick.x;
        Servo = (int)(-90*DStick.y);           // el - es porque el servo está instalado al
revez de lo que se quiere
        if(Motor1 * Motor1 > 1 || Motor2 * Motor2 > 1 || Motor3 * Motor3 > 1)
        {
            if(Motor1*Motor1>Motor2*Motor2 && Motor1*Motor1 > Motor3*Motor3)
            {
                Motor1 = Motor1 / Mathf.Abs(Motor1);
                Motor2 = Motor2 / Mathf.Abs(Motor1);
                Motor3 = Motor3 / Mathf.Abs(Motor1);
            }
            if(Motor2 * Motor2 > Motor1 * Motor1 && Motor2 * Motor2 > Motor3 * Motor3)
            {
                Motor1 = Motor1 / Mathf.Abs(Motor2);
                Motor2 = Motor2 / Mathf.Abs(Motor2);
                Motor3 = Motor3 / Mathf.Abs(Motor2);
            }
            if(Motor3 * Motor3 > Motor1 * Motor1 && Motor3 * Motor3 > Motor2 * Motor2)
            {
                Motor1 = Motor1 / Mathf.Abs(Motor3);
                Motor2 = Motor2 / Mathf.Abs(Motor3);
                Motor3 = Motor3 / Mathf.Abs(Motor3);
            }
        }
        Stepper1 = (int)(500 * Motor1);
        Stepper2 = (int)(500 * Motor2);
        Stepper3 = (int)(500 * Motor3);
    }

    public string LeerComando()
    {
        if (!ReleLeido)
        {
            ReleLeido = true;
    }
}

```

```

        }
    else
    {
        if (!TensorflowLeido)
        {
            TensorflowLeido = true;
        }
    }
    return ComunicaciónMensaje;
}

void CrearMensaje()
{
    //Se prepara el mensaje dando prioridad al relé y luego a desactivar el
    tensorflow
    if (buttonStart == 1 && PrevbuttonStart == 0)
    {

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexCon
trol.WaitOne();
        ComunicaciónMensaje = "R;";
        ReleLeido = false;

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexCon
trol.ReleaseMutex();
    }
    else if (ReleLeido && Y == 1 && PrevY == 0)
    {

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexCon
trol.WaitOne();
        ComunicaciónMensaje = "T;";
        TensorflowLeido = false;

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexCon
trol.ReleaseMutex();
    }
    else if (ReleLeido && TensorflowLeido)
    {
        StepperCalculate();

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexCon
trol.WaitOne();
        ComunicaciónMensaje = "W" + Stepper1.ToString() + "," + Stepper2.ToString() +
        "," + Stepper3.ToString() + "," + Servo.ToString() + ";";
    }

GameObject.Find("Raspberry_Comunicacion").GetComponent<Raspberry_Comunicacion>().MutexCon
trol.ReleaseMutex();
}
}

void CambiarInterfaz()
{
    //Se cambia entre las interfaces posibles
    string Interfaz=GameObject.Find("Interfaces").GetComponent<Interfaz>().UI;
    if (A == 1)
    {
        if (Interfaz == "Base")

```

```

        {
            GameObject.Find("Interfaces").GetComponent<Interfaz>().UI = "Camara";
        }
        else
        {
            GameObject.Find("Interfaces").GetComponent<Interfaz>().UI = "Base";
        }
    }
    else
    {
        if (Interfaz == "Base")
        {
            GameObject.Find("Interfaces").GetComponent<Interfaz>().UI = "Mapa";
        }
        else
        {
            GameObject.Find("Interfaces").GetComponent<Interfaz>().UI = "Base";
        }
    }
}

void UpdateButtons()
{
    PrevY = Y;
    PrevA = A;
    PrevLB = LB;
    PrevLT = LT;
    PrevRT = RT;
    PrevbuttonStart = buttonStart;
    PrevDpad = Dpad;
}

void Quit()
{
    Application.Quit();
    #if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
    #endif
}
}

```

## Raspberry\_Comunicacion

```

using System;
using System.Net.Sockets;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using System.IO;
using System.Text;
using System.Threading;

public class Raspberry_Comunicacion : MonoBehaviour
{
    private TcpClient ClientControl, ClientImagen, ClientNavegacion;
    private string hostIP;
    private int PortControl, PortImagen, PortNavegacion;

```

```
private NetworkStream NetStreamControl,NetStreamImagen,NetStreamNavegacion;
private float SendClock = 0;
private float SendTimer = 0.1f;
private float JoinTimer = 0;
public bool IsConnected = false;
private bool IsConnecting = false;
public string NavegacionDatos = "E;";
private Thread ControlThread;
private Thread ImagenThread;
private Thread NavegacionThread;
private Thread ReconnectThread;
private bool ImageReady = false;
public bool DataIsReady = false;
public byte[] BytesImagen= new byte[0];
public Mutex MutexImagen = new Mutex();
public Mutex MutexNavegacion = new Mutex();
public Mutex MutexControl = new Mutex();
public Mutex MutexConnect = new Mutex();
private bool ControlComReady = false;
private string message;

// Start is called before the first frame update
void Start()
{
    //Se definen las variables que definen al servidor en la raspberry
    ClientControl = new TcpClient();
    ClientImagen = new TcpClient();
    ClientNavegacion = new TcpClient();
    hostIP = "10.42.0.1";      //IP Raspberry
    PortControl = 40000;
    PortImagen = 40001;
    PortNavegacion = 40002;
    IsConnecting = true;
    //Se prepara el thread que conecta el cliente
    ThreadStart FunctionReconnect = new ThreadStart(Reconnect);
    ReconnectThread = new Thread(FunctionReconnect);
    ReconnectThread.Start();
}

void Connect()
{
    try
    {
        //Se conectan los clientes
        ClientControl.Connect(hostIP, PortControl);
        ClientImagen.Connect(hostIP, PortImagen);
        ClientNavegacion.Connect(hostIP, PortNavegacion);
        MutexConnect.WaitOne();
        if (IsConnecting)
        {
            IsConnected = true;
        }
        MutexConnect.ReleaseMutex();
        //Se preparan los network stream
        NetStreamControl = ClientControl.GetStream();
        NetStreamImagen = ClientImagen.GetStream();
        NetStreamNavegacion = ClientNavegacion.GetStream();
        //Preparamos la función para asignar al thread
```

```

        ThreadStart FunctionControl = new ThreadStart(Control);
        ThreadStart FunctionImagen = new ThreadStart(Imagen);
        ThreadStart FunctionNavegacion = new ThreadStart(Navegacion);
        //Se crean los threads y le asignamos la función
        ControlThread = new Thread(FunctionControl);
        ImagenThread = new Thread(FunctionImagen);
        NavegacionThread = new Thread(FunctionNavegacion);
        //Se inician los threads
        ControlThread.Start();
        ImagenThread.Start();
        NavegacionThread.Start();
        JoinTimer = 0;
    }
    catch (Exception)
    {
        IsConnected = false;
    }
    IsConnecting = false;
}

void Control()
{
    while (IsConnected)
    {
        if (SendClock > SendTimer)
        {
            if (!SocketConnected(ClientControl))
            {
                IsConnected = false;
            }
            else
            {
                try
                {
                    ControlComReady = true;
                    while (ControlComReady) { }
                    //Se envia el mensaje
                    byte[] SendBytes = Encoding.UTF8.GetBytes(message);
                    NetStreamControl.Write(SendBytes, 0, SendBytes.Length);
                    SendClock = 0;
                }
                catch (Exception)
                {
                    IsConnected = false;
                }
            }
        }
    }
}

void Imagen()
{
    byte[] Buffer = new byte[0];
    byte[] RecBytes = new byte[ClientImagen.ReceiveBufferSize];
    while (IsConnected)
    {
        if (!SocketConnected(ClientImagen))
        {

```

```

        IsConnected = false;
    }
    else
    {
        try
        {
            //Se recibe el mensaje
            if (NetStreamImagen.DataAvailable)
            {
                int BytesRecibidos = NetStreamImagen.Read(RecBytes, 0,
RecBytes.Length);
                byte[] Recibidos = new byte[1048576];
                Array.Copy(RecBytes, Recibidos, BytesRecibidos);
                Array.Resize(ref Recibidos, BytesRecibidos);
                Buffer = Buffer.Concat(Recibidos).ToArray();
                if (Encoding.UTF8.GetString(Buffer, Buffer.Length - 1, 1) == ";")
                {
                    MutexImagen.WaitOne();
                    ImageReady = true;
                    BytesImagen = Buffer;
                    MutexImagen.ReleaseMutex();
                    Array.Resize(ref Buffer, 0);
                }
            }
        }
        catch (Exception)
        {
            IsConnected = false;
        }
    }
}
}

void Navegacion()
{
    string Buffer = "";
    byte[] RecBytes = new byte[ClientNavegacion.ReceiveBufferSize];
    while (IsConnected)
    {
        if (! SocketConnected(ClientNavegacion))
        {
            IsConnected = false;
        }
        else
        {
            try
            {
                //Se recibe el mensaje
                if (NetStreamNavegacion.DataAvailable)
                {
                    int BytesRecibidos = NetStreamNavegacion.Read(RecBytes, 0,
RecBytes.Length);
                    Buffer = Buffer + Encoding.UTF8.GetString(RecBytes, 0,
BytesRecibidos);
                    if (Buffer[Buffer.Length - 1] == ';')
                    {
                        MutexNavegacion.WaitOne();
                        NavegacionDatos = Buffer;
                    }
                }
            }
        }
    }
}

```

```
        MutexNavegacion.ReleaseMutex();
        Buffer = "";
    }
}
catch (Exception)
{
    IsConnected = false;
}
}
}
}

void Reconnect()
{
    //Se desconectan los posibles clientes que no se hayan desconectado y se
reconecta
    IsConnecting = true;
    try
    {
        ControlThread.Join();
        ImagenThread.Join();
        NavegacionThread.Join();
    }
    catch (Exception) {
        IsConnected = false;
    }
    Connect();
}

public bool ImageIsReady()
{
    if (ImageReady)
    {
        ImageReady = false;
        return true;
    }
    else
    {
        return false;
    }
}

// Update is called once per frame
void Update()
{
    //Se controla si hay conexion y se da un timeout
    if (IsConnected == false && IsConnecting == false)
    {
        JoinTimer = JoinTimer + Time.deltaTime;
        if (JoinTimer > 5)
        {
            ThreadStart FunctionReconnect = new ThreadStart(Reconnect);
            ReconnectThread = new Thread(FunctionReconnect);
            ReconnectThread.Start();
            JoinTimer = 0;
        }
    }
}
```

```
//Se lee el comando y se reinicia el timer
else
{
    if (ControlComReady)
    {
        MutexControl.WaitOne();
        message =
GameObject.Find("Lector_USB").GetComponent<Lector_USB>().LeerComando();
        MutexControl.ReleaseMutex();
        ControlComReady = false;
    }
    SendClock = SendClock + Time.deltaTime;
    JointTimer = 0;
}
}

bool SocketConnected(TcpClient s)
{
    bool B3 = s.Connected; // True si la conexion se
realizo con exito en primer lugar
    return B3;
}

void OnApplicationQuit()
{
    MutexConnect.WaitOne();
    if (IsConnected)
    {
        //Se da la orden de desconexion y se cierran los threads
        IsConnecting = false;
        IsConnected = false;
        MutexConnect.ReleaseMutex();
        ReconnectThread.Join();
        ControlThread.Join();
        ImagenThread.Join();
        NavegacionThread.Join();
    }
    else
    {
        IsConnecting = false;
        IsConnected = false;
        MutexConnect.ReleaseMutex();
    }
}
```

## Jerarquía de escena



Imagen 56: Jerarquía de escena, terminal.

## Anexo: Código Dataset Generator (Unity C#)

### MyActivationTag.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Perception.Randomization.Parameters;
using UnityEngine.Perception.Randomization.Randomizers;
using UnityEngine.Perception.Randomization.Samplers;
using System.IO;

public class MyActivationTag : Randomizer
{
    private int Iter = 0;
    public Texture2D SavedTexture = null;

    // Se corre en cada iteracion
    protected override void OnIterationStart()
    {
        //Encuentra en el escenario de simulación el activador de objetos a detectar y lo
        desactiva cuando no se requieren mas objetos
        //if (Iter > 0)
        if (Iter > 950)
        {
            GameObject.Find("Simulation Scenario").transform.Find("Foreground
Objects").gameObject.SetActive(false);
        }
        Iter = Iter + 1;
        int RandomActivation = UnityEngine.Random.Range(0, 2);
        //Crea un numero aleatorio entero entre 0 y 2, sin tener en cuenta el 2 y si es 1
        activa la pared con una textura aleatoria
        if(RandomActivation==1)
        {
            if(!GameObject.Find("Fondo").GetComponent<Renderer>().isVisible)
            {
                GameObject.Find("Fondo").GetComponent<Renderer>().enabled = true;
            }
            int ImageNum= UnityEngine.Random.Range(0, 71);
            string ImageString = "C:/Users/MAXI/Dropbox/Proyecto final de
estudios/Unity/Dataset Generator/Assets/My Asset/Fondos/" + ImageNum.ToString() + ".png";
            Texture2D Textura = new Texture2D(640,480);
            byte[] ImagenBytes = File.ReadAllBytes(ImageString);
            Textura.LoadImage(ImagenBytes);
            if (Textura.height > 8)           //La textura de error es de 8x8
            {
                Textura.Apply();
                SavedTexture = Textura;
            }
            GameObject.Find("Fondo").GetComponent
            <Renderer>().material.mainTexture=SavedTexture;
        }
        else
        {
            if (GameObject.Find("Fondo").GetComponent<Renderer>().isVisible)
            {
                GameObject.Find("Fondo").GetComponent<Renderer>().enabled = false;
            }
        }
    }
}

```

}

## MyRotationRandomizerTag.cs

```

using System;
using System.Collections;
using UnityEngine;
using UnityEngine.Perception.Randomization.Parameters;
using UnityEngine.Perception.Randomization.Randomizers;
using UnityEngine.Perception.Randomization.Samplers;

[RequireComponent(typeof(Transform))] //Solo pueden usarse en objetos que contengan el tag

public class MyRotationRandomizerTag : RandomizerTag
{
    public float minAngle;
    public float maxAngle;
    public float minScale;
    public float maxScale;

    public void SetRotation(float RotationX, float RotationY, float RotationZ, float Scale)
    {
        //Se cambia la rotación del objeto y su escala en base a lo que deseamos
        var tagRot = GetComponent<Transform>();
        float QuatX = RotationX * (maxAngle - minAngle) + minAngle;
        float QuatY = RotationY * (maxAngle - minAngle) + minAngle;
        float QuatZ = RotationZ * (maxAngle - minAngle) + minAngle;
        float newScale = Scale * (maxScale - minScale) + minScale;
        tagRot.eulerAngles = new Vector3(QuatX, QuatY, QuatZ);
        tagRot.transform.localScale = new Vector3(newScale, newScale, newScale);
    }
}

[Serializable]
[AddRandomizerMenu("MyRotationRandomizer")]
public class MyRotationRandomizer : Randomizer
{
    // Se elige un valor entre 0 y 1
    public FloatParameter RotationX = new() { value = new UniformSampler(0, 1) };
    public FloatParameter RotationY = new() { value = new UniformSampler(0, 1) };
    public FloatParameter RotationZ = new() { value = new UniformSampler(0, 1) };
    public FloatParameter Scale = new() { value = new UniformSampler(0, 1) };

    // Se corre en cada iteracion
    protected override void OnIterationStart()
    {
        // Se buscan todos los objetos con el tag
        var tags = tagManager.Query<MyRotationRandomizerTag>();
        foreach (var tag in tags)
        {
            // Se toma un numero aleatorio entre 0 y 1 para realizar la rotación
            tag.SetRotation(RotationX.Sample(), RotationY.Sample(),
                RotationZ.Sample(), Scale.Sample());
        }
    }
}

```

```
        }
    }
}
```

### MyLightRandomizerTag.cs

```
using System;
using UnityEngine;
using UnityEngine.Perception.Randomization.Parameters;
using UnityEngine.Perception.Randomization.Randomizers;
using UnityEngine.Perception.Randomization.Samplers;

[RequireComponent(typeof(Light))] //Solo puede agregarse en objetos con una luz
public class MyLightRandomizerTag : RandomizerTag
{
    public float minIntensity;
    public float maxIntensity;

    public void SetIntensity(float rawIntensity)
    {
        //Se cambia la intensidad de la luz
        var tagLight = GetComponent<Light>();
        var scaledIntensity = rawIntensity * (maxIntensity - minIntensity) +
minIntensity;
        tagLight.intensity = scaledIntensity;
    }
}

[Serializable]
[AddRandomizerMenu("MyLightRandomizer")]
public class MyLightRandomizer : Randomizer
{
    // Se eligen valores entre 0 y 1
    public FloatParameter lightIntensity = new() { value = new UniformSampler(0, 1) };
    public ColorRgbParameter color;

    // Se corre en cada iteracion
    protected override void OnIterationStart()
    {
        // Toma todos los objetos con el tag en la escena
        var tags = tagManager.Query<MyLightRandomizerTag>();
        foreach (var tag in tags)
        {
            //Toma la luz en el objeto
            var tagLight = tag.GetComponent<Light>();
            tagLight.color = color.Sample();
            //Cambia la intensidad
            tag.SetIntensity(lightIntensity.Sample());
        }
    }
}
```

## Jerarquía de escena

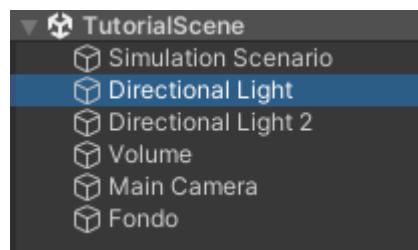


Imagen 57: Jerarquía de escena, Dataset Generator.