E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Mini Swarm Robots

Grado en Ingeniería Informática

Trabajo Fin de Grado

Luis José Llamas Pérez

Asier Ruperto Marzo Pérez

Josu Irisarri Erviti

Pamplona, 19/01/2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

A mis tutores, Asier Marzo y Josu Irisarri.

A mis compañeros de la carrera.

A Biko2 y 540.

A la Universidad Pública de Navarra

Resumen

Este proyecto recoge el desarrollo de una inteligencia colectiva que pueda ser ampliable, de fácil implementación y bajo coste. El desarrollo de dicha inteligencia colectiva se divide en diferentes fases que abarcan tanto el diseño del robot y la inteligencia colectiva a nivel de hardware, como la implementación software de los mismos que permita los objetivos expuestos anteriormente.

A nivel de hardware se ha realizado el diseño e implementación de un chasis impreso en 3D, así como los circuitos electrónicos utilizados en los robots. Cabe destacar el uso de componentes fáciles de obtener tanto en el robot como en el sistema centralizado.

En lo que al Software se refiere, se ha realizado desde cero con la mentalidad de que pueda ser fácilmente ampliable y configurable. Esto se ha conseguido utilizando técnicas de diseño de software como la inyección de dependencias, Mocks y TDD (Desarrollo guiado por pruebas).

Palabras clave

- Inteligencia colectiva
- Swarm robotics
- Robot
- ArUco
- TDD
- Inyección de Dependencias
- Mock

Índice

Introducción

A pesar de que la **inteligencia colectiva** no es algo nuevo, se ha trabajado poco en investigaciones para conseguir que sea de **bajo coste y ampliable** para **cubrir diferentes necesidades**.

Este proyecto surge para dar un primer paso que solucione esos inconvenientes con el fin de atraer la atención hacía el uso de la inteligencia colectiva como forma de resolver problemas.

Justificación y Objetivos

El proyecto parte de la premisa de que la inteligencia colectiva no tiene que ser ni cara, ni complicada, fácilmente replicable y debe ser capaz de analizar y comunicarse con el entorno.

A raíz de la premisa anterior, surgen los siguientes objetivos:

- Los robots deben poder ser construidos con diferentes componentes, es decir, no tiene por qué haber dos robots iguales.
- Añadir un robot que use otro tipo de componentes no debe complicar el desarrollo ni alterar el funcionamiento de los que ya existen.
- El número de robots que puedan participar en la inteligencia colectiva ha de ser alto.
- Escalabilidad
- Los robots cumplirán una de las siguientes opciones:
    - Tener inteligencia propia y ser ellos mismo los responsables de la toma de decisiones
    - Tener un sistema centralizado que se encargue de dichas decisiones.
    - Ambas.
- El sistema de comunicación entre robots y/o el sistema centralizado tendrá que ser flexible, es decir, que se podrán utilizar diferentes tipos de sistemas de comunicación
- Los robots podrán tener comportamientos/patrones y actuar en base al entorno

Debido al amplio campo que es la inteligencia colectiva, no se van a poder abordar todos los objetivos iniciales en su totalidad, sin embargo, sí que se ha creado el camino para poder aumentar en funcionalidad y diseño de manera que en un futuro otra persona pueda continuar con el proyecto sin gran dificultad.

Para ello se han tomado una serie de decisiones que simplifican y a su vez acotan el alcance del proyecto. Estas decisiones se listan a continuación:

- Uso de material fácilmente adquirible por cualquier persona.
- Dicho material ha de ser de bajo coste.
- Software que sea fácilmente ampliable y modificable.
- Los robots serán controlados por un sistema centralizado.
- Solamente se usará un sistema comunicación entre los robots y el sistema centralizado.
- Se crearán dos robots con distintos componentes para la flexibilidad del proyecto.
- El control que ejerce el sistema centralizado será únicamente el movimiento de los robots de un punto a otro.

En los siguientes puntos se hablará sobre el hardware y software concreto que se ha decidido usar en el proyecto.

Contexto tecnológico

La inteligencia colectiva es definida como el estudio del comportamiento colectivo cuando un sistema este compuesto de múltiples individuos que actúan en busca de un objetivo común. Esto permite abordar problemas de gran complejidad con individuos cuyas capacidades físicas/computacionales por separado sean inferiores a lo requerido.

Esto se consigue mediante el uso de comandos e interacciones simples que son elaboradas por un sistema central o individuos de mayor rango jerárquico.

Metodología empleada

ArUco

Para la detección y clasificación de los robots se van a utilizar marcadores **ArUco**, los cuales fueron diseñados por el grupo **Ava** de la **Universidad de Córdoba** (España) para proveer marcadores en tiempo real basándose en marcadores de **Realidad Aumentada**.

Estos marcadores se generan en base a un diccionario el cual tiene un **tamaño predefinido** de identificadores dependiendo del número de “bloques” que vaya a tener el marcador.

Estos marcadores tienen una característica muy importante y es que **tienen orientación**, por lo que podemos saber en todo momento el **ángulo del robot**.

Como detalle a tener en cuenta, estos marcadores tienen que tener un **borde** de otro color que **no sea negro** para funcionar correctamente.

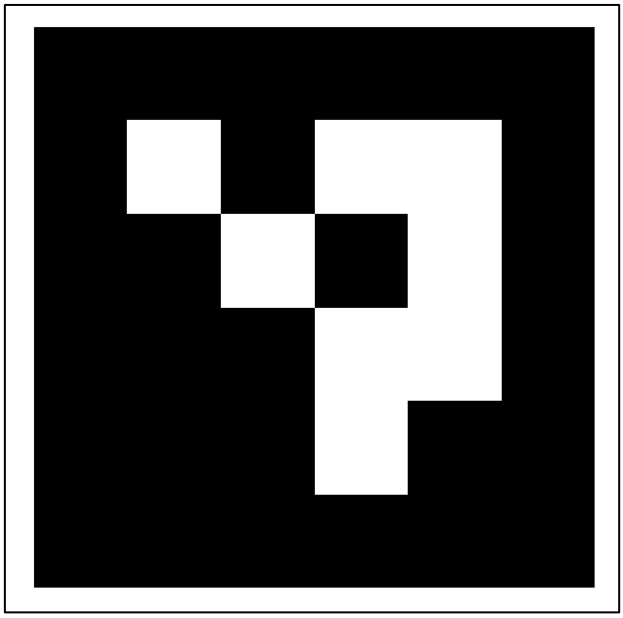

Figura 1: Marcador ArUco que representa el 0

Al ser un diseño muy simple y en blanco y negro, se puede **binarizar** la imagen antes de llegar a la fase de detección, lo cual hace que **el coste computacional** de detección y clasificación sea **relativamente bajo**, lo que convierte a este tipo de marcadores en una **solución ideal** para una aplicación como la que se propone en este proyecto.

Mocks y TDD

La metodología empleada para el desarrollo del Software de este proyecto ha sido una modificación de **TDD (Desarrollo guiado por pruebas)** junto al uso de **Mocks (Objetos simulados)**. Esto nos ha permitido comenzar a trabajar en el software sin conocer aún como va a ser el hardware, de esta forma, hemos conseguido desacoplar la dependencia del software y hardware.

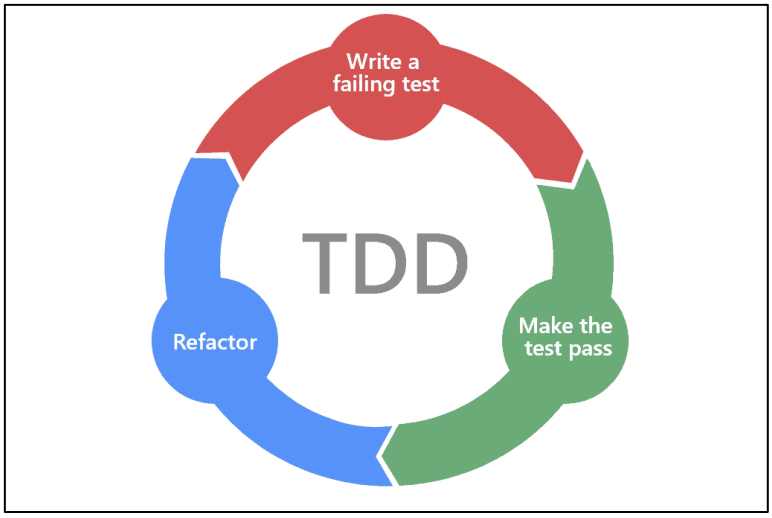

Figura 2: Diagrama de los estados en TDD (Rojo, Verde, Azul)

La manera en la que lo hemos logrado ha sido mediante la creación de Mocks de objetos que más adelante serán implementados de forma “real”.

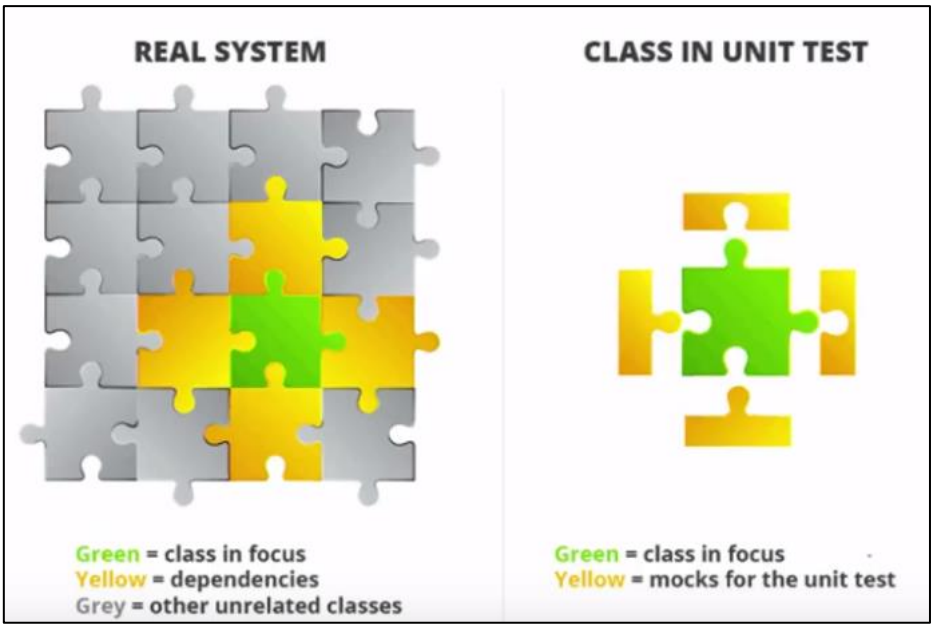

Figura 3: Ejemplo visual de Mocks

Usando la imagen de la Figura 3 como ejemplo, la parte en verde sería el **sistema centralizado**, mientras que las partes amarillas serían las dependencias del mismo, como, por ejemplo:

- Webcam (Para la obtención de posicionamiento de los robots).
- Sistema de comunicaciones (Para dar las órdenes a los robots).
- Robot (Para ejecutar las ordenes).

Si no hubiésemos usado **Mocks**, tendríamos que haber esperado a la implementación real de estos componentes antes de ponernos a trabajar en el sistema centralizado, ya que, no tendríamos **feedback** de si lo que estamos desarrollando es funcional o no.

Para solventar esto, se ha realizado un análisis previo de los componentes/actores necesarios por el sistema. Estos **actores** son candidatos a poder ser Mockeados, es decir, a crear una implementación "falsa" con las características y métodos que serán utilizados por la **implementación real**.

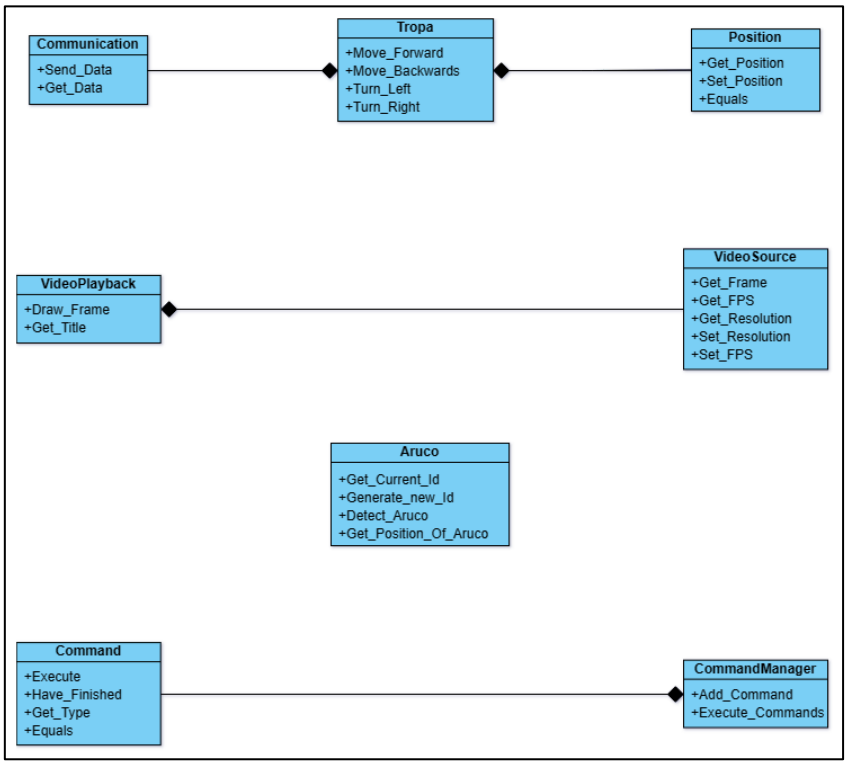

Figura 4: Actores del sistema centralizado

Los robots (**Tropas**) son un perfecto candidato a ser Mockeados, ya que nos permite desarrollar sin tener dicho robot. Esto genera un efecto mariposa, ya que, al no tener el robot, tampoco podemos obtener su posición usando una Webcam (**VideoSource**), por lo que el **VideoSource** también será mockeado. Lo mismo ocurre también con el sistema de comunicaciones (**Communication**).

El resto de componentes no tienen necesidad de ser **Mockeados**, al menos a primera vista, por lo que se puede proceder a la creación del sistema centralizado.

Inyección de Dependencias

Dado que hemos realizado el análisis de los componentes y sus acciones/métodos, podemos enfocar el desarrollo de estos usando la inyección de dependencias, lo cual nos permitirá el desacople de las implementaciones finales consiguiendo así, un software fácilmente mantenible y actualizable.

El primer paso es crear interfaces en donde antes eran actores simples. Estas interfaces declararan los atributos y métodos/acciones del actor. A raíz de esta interfaz, se crearán las implementaciones finales, las cuales deben cumplir con el “contrato” (interfaz) que hereden.

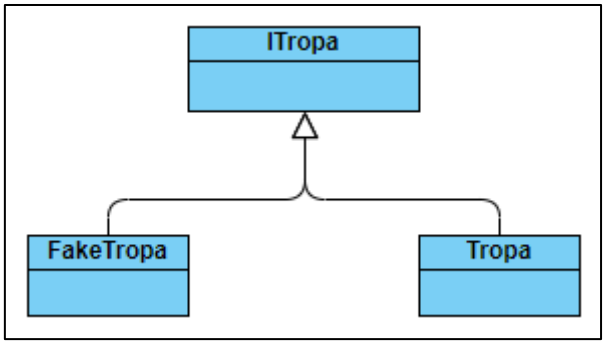

Figura 5: Primer paso para la Inyección de Dependencias

En el ejemplo de la Figura 5 se ha creado la interfaz **ITropa** la cual tendrá una serie de métodos/acciones que tanto **FakeTropa** como Tropa deberán de implementar, cada uno a su manera.

Los Robots (**Tropas**) como hemos visto en la Figura 4, dependen del sistema de comunicaciones (**Communication**), por lo que estamos ante de una dependencia que podemos romper y hacerlo más flexible aplicando la simplificación de la Figura 4 al actor Communication.

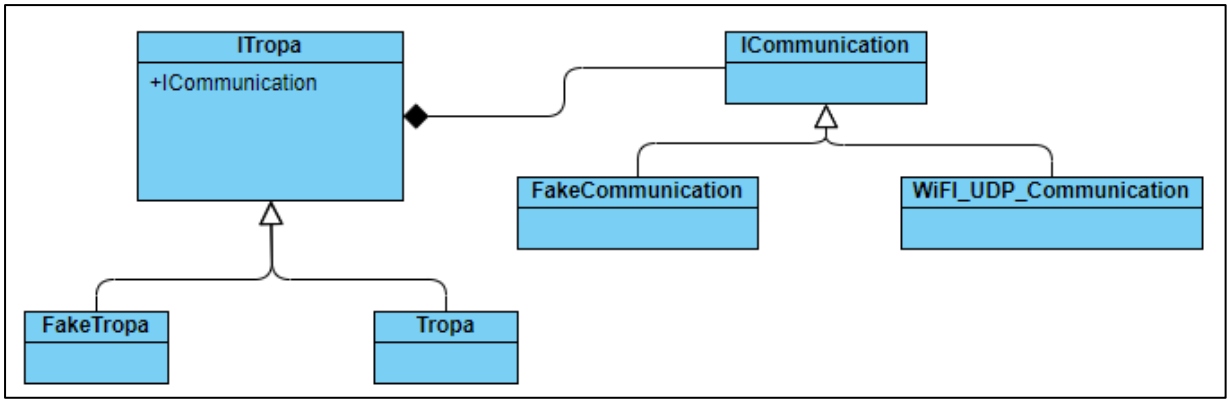

Figura 6: Inyección del sistema de comunicaciones como dependencia

Como podemos observar en la Figura 6, las **Tropas** ya no dependen de la implementación concreta del sistema de comunicaciones, si no que podemos mezclar, por ejemplo, el uso de una **Tropa falsa** y un **sistema de comunicaciones** basado en **UDP** por **WiFi**. Esto es lo que nos va a permitir añadir en un futuro de forma sencilla implementaciones concretas (ya sean reales o Mocks) sin tener que modificar el código que ya existía previamente.

Tras realizar este ejercicio sobre el resto de los actores del Sistema Centralizado, obtenemos el diagrama de clases de la Figura 7.

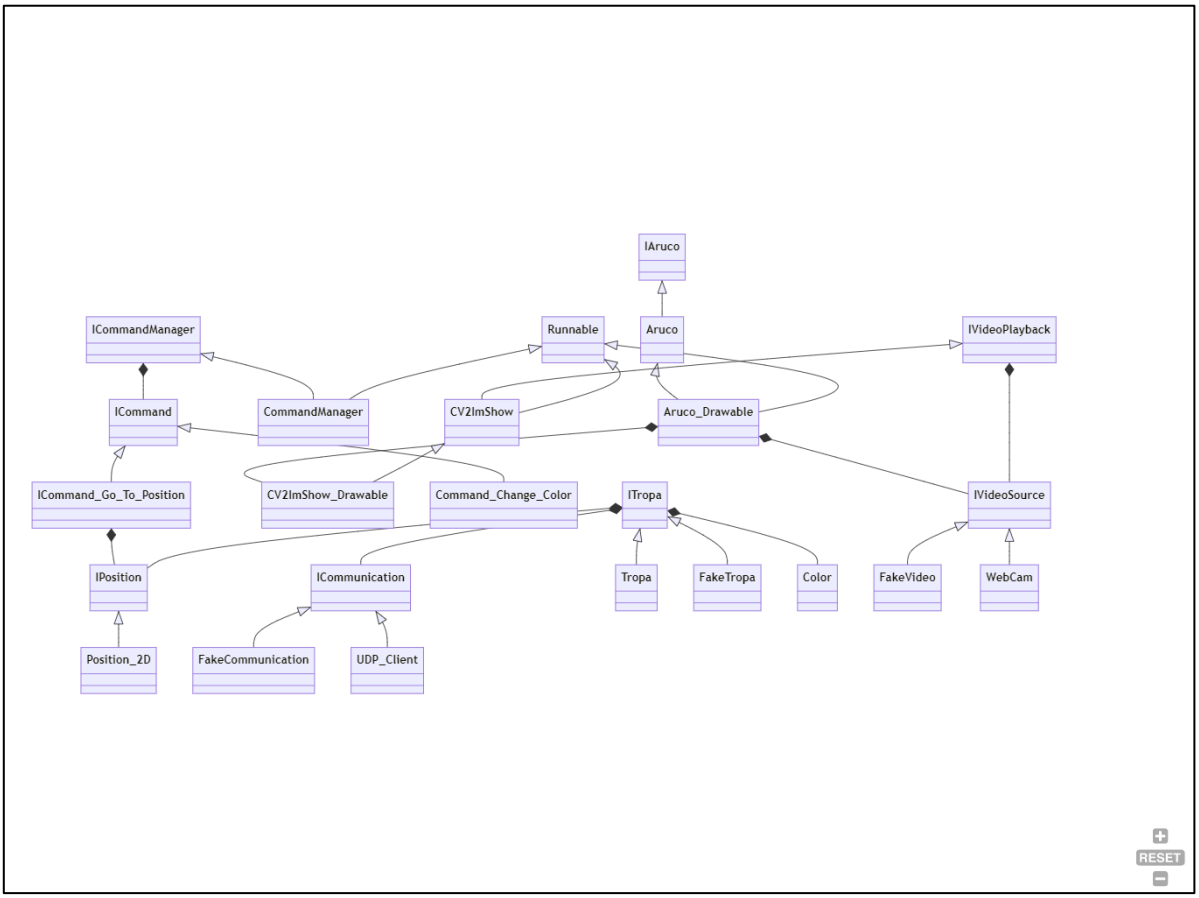

Figura 7: Diagrama de clases del Sistema Centralizado

Tal y como hemos visto en el sistema centralizado, también se pueden aplicar estas técnicas a la hora de la programación de los robots, donde la abstracción será en los componentes del Robot (Motor, Led, Sistema de Comunicación).

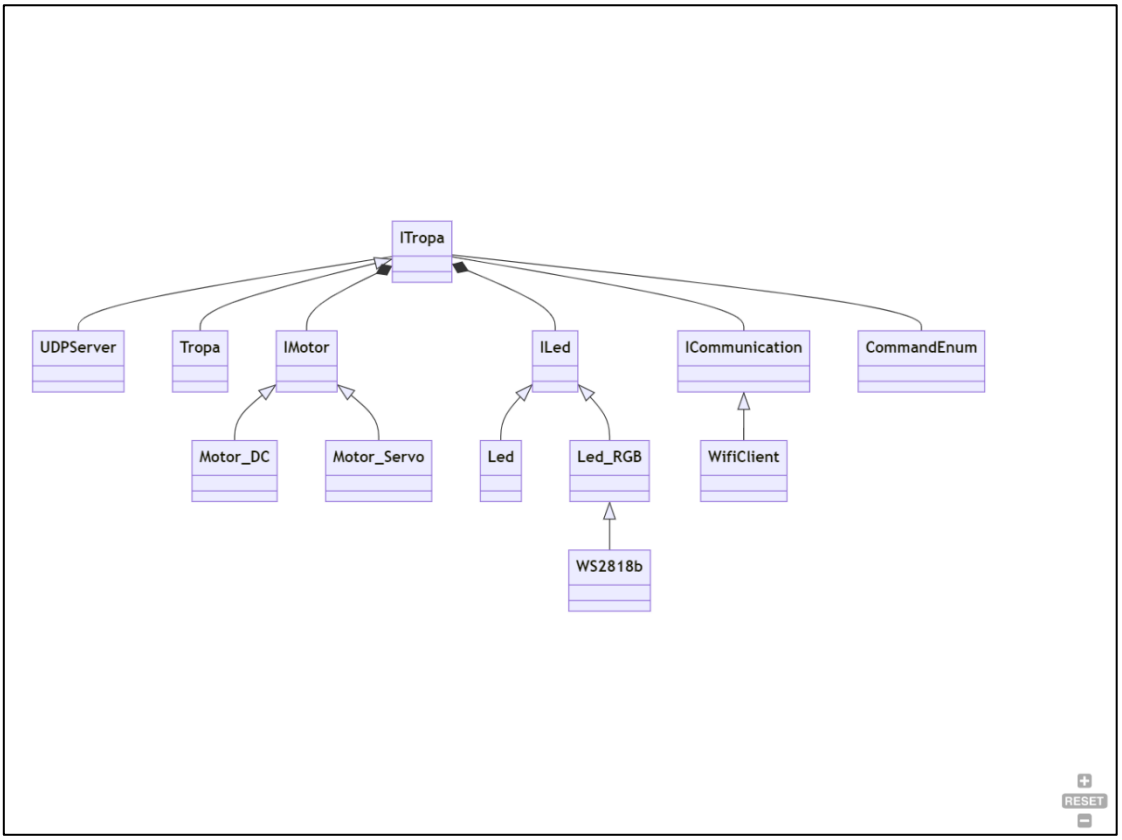

Figura 8: Diagrama de clases del Robot

Una vez se ha creado el software, pasar al hardware es trivial, ya que, se utilizan componentes fácilmente obtenibles, donde la única preocupación es la “conexión” entre ellos, como por ejemplo el control de los motores.

Mini Swarm Robots

Hardware

El hardware como se ha explicado anteriormente, es de bajo coste y accesible. En líneas generales hay dos grandes separaciones, el hardware de los robots y el del sistema centralizado.

Robots

En el caso de los robots, lo más importante para cumplir los objetivos, es el movimiento. En este caso se ha decidido usar un sistema de dos ruedas motorizadas encargadas de la tracción y una rueda “loca”.

Las ruedas motorizadas como su nombre indica, consisten en dos motores que al girar a distintas velocidades consiguen el movimiento y giro del robot. Para controlar dichos motores, se utiliza un **microcontrolador** que será el cerebro del robot y para alimentar el robot, un **Powerbank** de **5V** y **1000mAh**. Todo esto va integrado en un cuerpo impreso en 3D que permita que todos los componentes tengan un balance de peso optimo.

Microcontrolador

El microcontrolador que se ha utilizado es un **ESP32 WROOM-32**, el cual viene equipado con WiFi y Bluetooth.

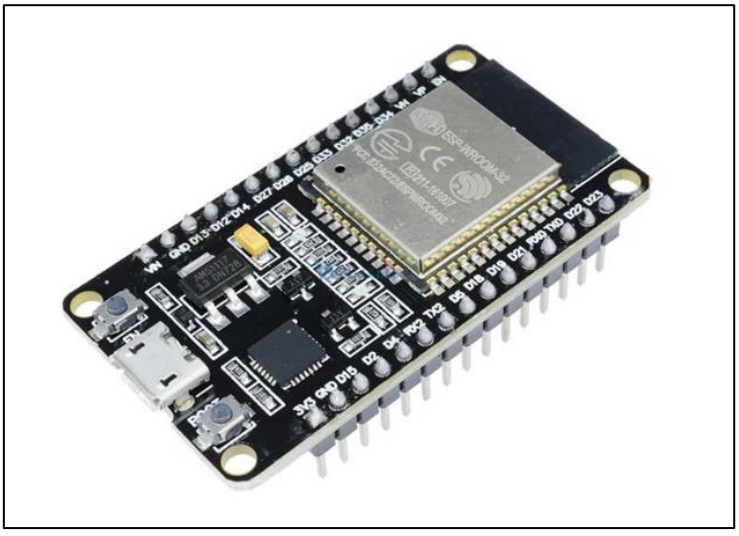

Figura 9: ESP32 WROOM-32

Este microcontrolador es más que suficiente para el proyecto y su elección radica en el hecho de que tiene un precio asequible, es fácil de obtener y tiene una gran cantidad de puertos de entrada/salida, así como comunicaciones WiFi y Bluetooth incluidas.

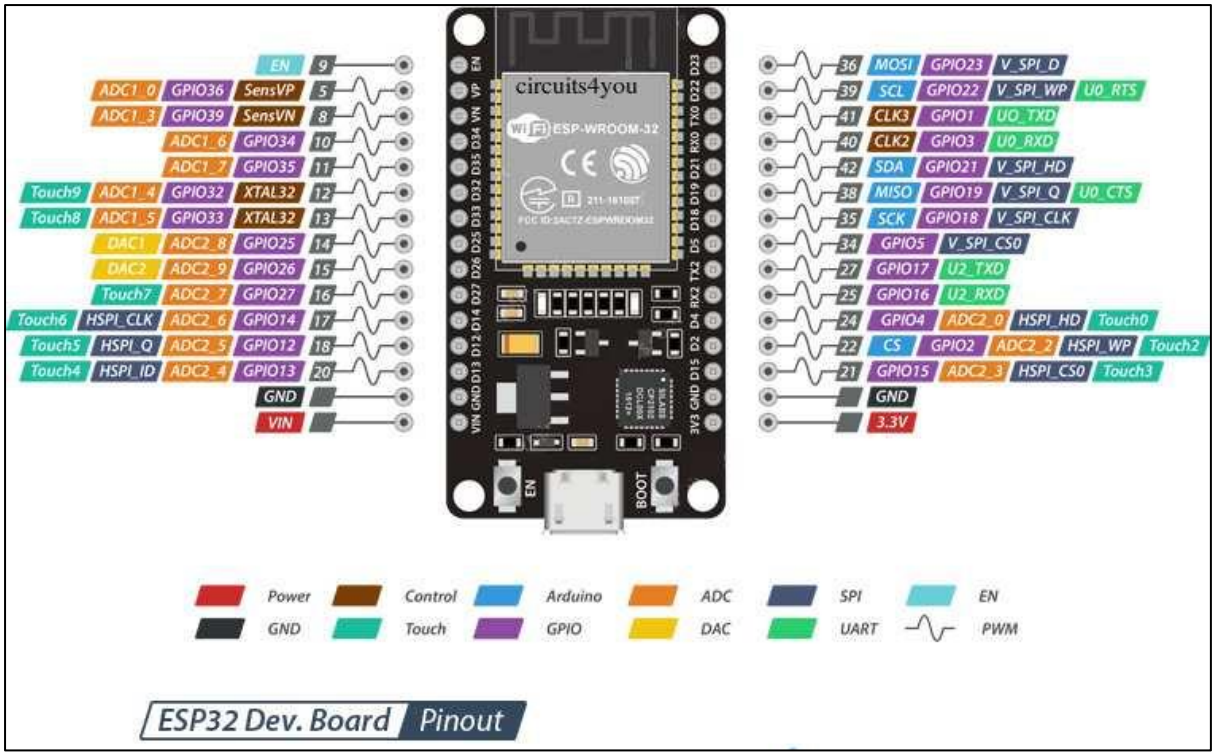

Figura 10: Pinout del ESP32 WROOM-32

Como características importantes, el microcontrolador cuenta con un regulador de voltaje que permite ser alimentado desde **12V hasta 3.3V**. Cabe destacar que el voltaje al que opera este dispositivo es de 3.3V, por lo que puede dar problemas con circuitos integrados cuyo umbral de activación sea superior a dicho voltaje. Esto puede remediarse con un transistor que actúe de Step-Up usando un pin de entrada/salida (**GPIO**) del microcontrolador.

Otra característica de este microcontrolador es la posibilidad de utilizar un sistema de comunicaciones propio basado en WiFi llamado **ESP-MESH**, el cual puede ser interesante de cara a aumentar tanto el número de robots, como la distancia entre estos y el sistema centralizado.

Motores

Los motores utilizados en los robots que se han prototipado son motores DC simples con una caja reductora y motores Servo modificados para permitir el giro completo.

Motor DC

Para el caso de los motores DC simples se ha utilizado el circuito integrado **L293D** para controlar hasta dos motores por circuito. Este circuito integrado permite controlar tanto el sentido de giro de los motores, como su velocidad.

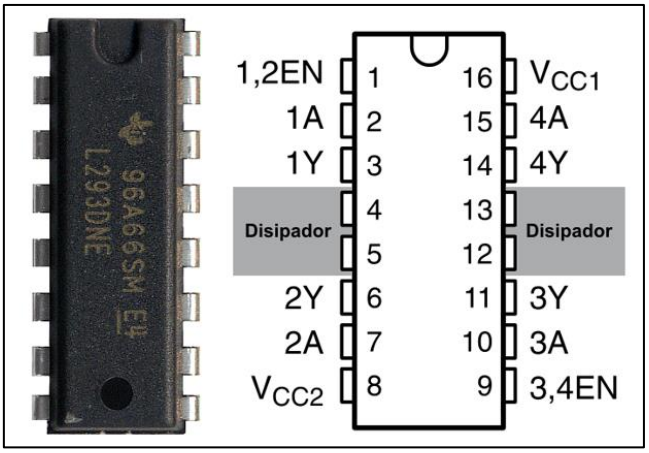

Figura 11: Pinout del IC L293D

Para obtener un movimiento más controlado se va a hacer uso de un sistema de engranajes que actúe de reductora. De esta forma, el movimiento del motor pasa a estar algo más controlado que si se dejase sin dicha reductora..

Figura 12: Sistema de engranajes reductor

Motor Servo

En el caso de los motores Servo, no se necesita el uso de ICs ya que, estos incluyen su propio controlador, así como un sistema de engranajes que actúa de reductora. La desventaja es que hay que modificar cada motor servo para permitir el movimiento continuo. Como ventaja, basta con conectar un único pin al microcontrolador por cada motor, mientras que en el caso del **L293D**, hay que conectar dos pines por cada motor.

Figura 13: Motor Servo

El proceso de modificación de un motor servo para permitir el giro continuo consta de cinco pasos:

1. Desarmar el servo.
2. Eliminar el limitador físico del sistema de engranajes.
3. Mover fuera el potenciómetro que se encarga de determinar la posición del servo.
4. Volver a armar el servo.
5. Calibrar el potenciómetro de forma que el servo piense que está siempre en el punto medio (90 grados).

Tras esta modificación, se tiene un control de dirección (por debajo del punto medio girará en un sentido, mientras que, por encima de dicho punto, girará en el sentido contrario) y de velocidad (conforme más nos alejemos del punto medio, a más velocidad girará el motor).

Chasis

El chasis empleado se ha diseñado para los componentes específicos que se han usado en los robots. Es un prototipo simple que se puede imprimir rápidamente y que tiene en cuenta el equilibrio de los pesos de los componentes para tratar de conseguir un movimiento menos errático.

El chasis se ha diseñado utilizando la herramienta **Autodesk Fusion 360** y se ha impreso en una impresora **Prusa MK3** utilizando **PrusaSlicer** para convertir del formato **STL** a **GCODE**.

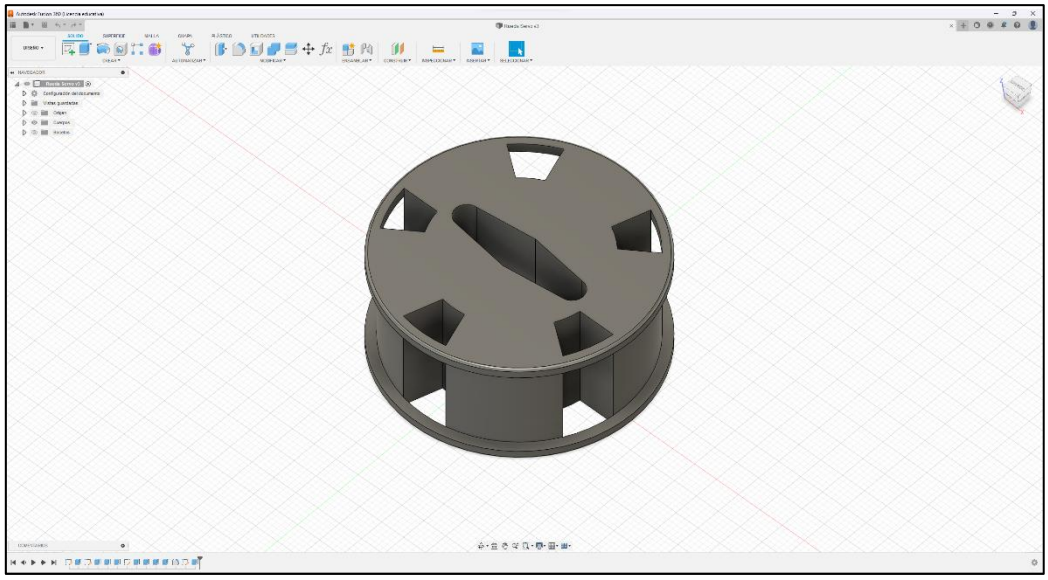

Figura 14: Diseño de las ruedas en Autodesk Fusion 360

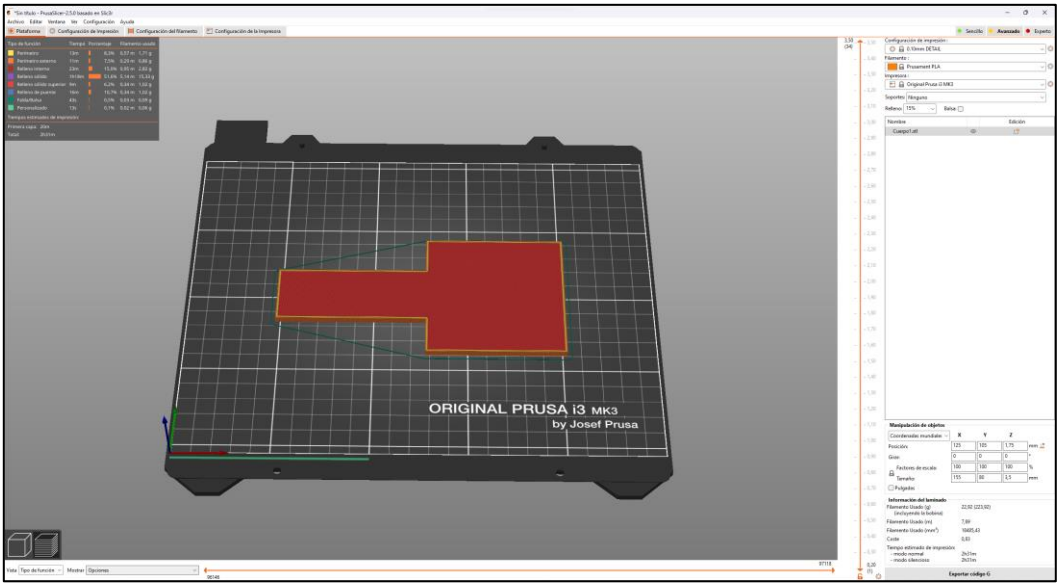

Figura 15: PrusaSlicer

Circuitos

La Figura 14 muestra como se ha realizado el conexionado del robot en el caso de los motores DC.

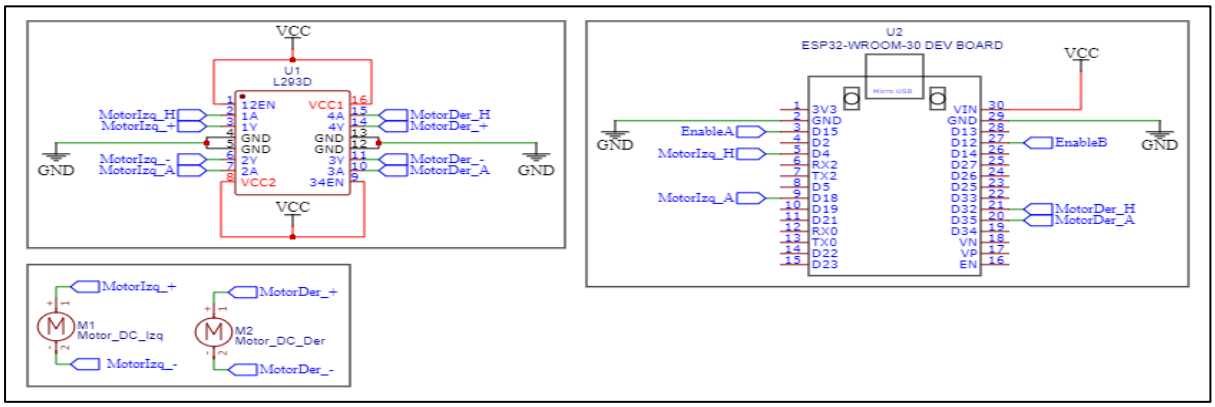

Figura 16: Circuito del Robot con motores DC

Sistema centralizado

El hardware del sistema centralizado se compone de las siguientes partes:

- PC donde ejecutar el código
- Webcam que irá conectada al PC
- Red WiFi (para el caso concreto que hemos implementado)

Los requisitos mínimos del PC son el poder ejecutar Python 3.9, al menos cuatro núcleos/hilos y 4 GB de RAM.

No se especifica arquitectura ya que, es válido tanto en un sistema convencional **x86/x64** como en ARM64 (**aarch64**).

El requisito de los cuatro núcleos/hilos viene debido a la utilización de multihilo para mejorar el rendimiento del código. Esto viene explicado con detalle en el apartado de Software.

Presupuesto

Ya que se ha hecho hincapié en que el proyecto sea accesible y barato, a continuación, se recoge una lista de los materiales necesarios para la recreación del proyecto junto a su precio (*).

Robot con motores DC

Componente	Cantidad	Precio unitario	Total
ESP32-WROOM	1	3.57€	3.57€
IC L293D	1	0.22€	0.22€
Motor DC	2	0.73€	1.46€
Powerbank	1	1.29€	1.29€
Chasis impresión 3D	1	1.00€	1.00€
Total	**7.54€**		

Tabla 1: Presupuesto del Robot con motores DC

Robot con motores servo

Componente	Cantidad	Precio unitario	Total
ESP32-WROOM	1	3.57€	3.57€
Motor Servo	2	1.48€	2.96€
Powerbank	1	1.29€	1.29€
Chasis impresión 3D	1	1.00€	1.00€
Rueda impresión 3D	2	1.09€	2.18€
Total	**12.00€**		

Tabla 2: Presupuesto del Robot con motores servo

*: Estos precios se han obtenido de Aliexpress en el momento de su redacción, por lo que pueden estar sujetos a cambios.

Resultados

Los resultados de esta implementación Hardware pueden ser vistos en la Figura 17.

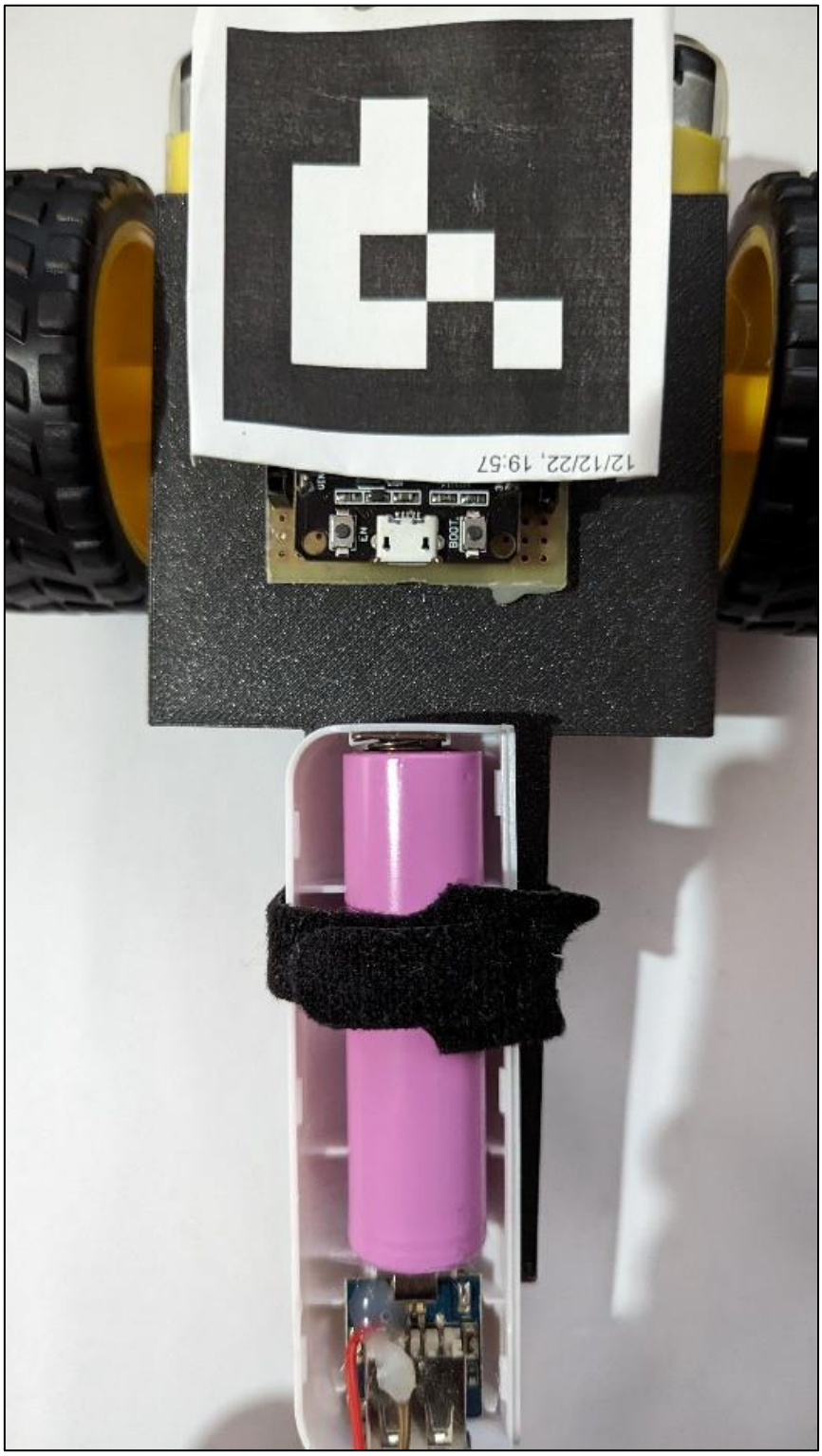

Figura 17: Prototipo del Robot con motores DC

Software

Como ya se ha explicado en el apartado de Metodología empleada, el Software se ha enfocado en el uso de **TDD** e **Inyección de Dependencias**.

En este apartado se va a mostrar en más detalle cómo se ha construido el Software tanto para los robots, como para el sistema centralizado.

Robots

La parte de los robots es la más sencilla debido a que se trata de una implementación ligada al Hardware en la que solamente hay que hacer el Software que sirva de intermediario entre el Hardware y el sistema centralizado.

Dado que ya se ha realizado el diagrama de clases de los robots en la Figura 8 y lo sencillo del software de los robots, se va a pasar al sistema centralizado.

Sistema Centralizado

El sistema centralizado es la parte más importante en cuanto a Software se refiere, ya que, es la parte encargada de gestionar las acciones/comportamientos, así como detección de los individuos.

La explicación se va a organizar en apartados donde se explicarán los elementos más relevantes de cada actor. También se van a dedicar apartados al **movimiento**, al **multihilo** y al **simulador**.

Tropas

En el caso de las Tropas, no hay nada destacable, ya que, las acciones de estas son realmente enviar la acción usando el sistema de comunicaciones al robot real.

Las acciones que se han definido son las siguientes:

0. Moverse hacía adelante
1. Moverse hacía atrás
2. Girar a la izquierda
3. Girar a la derecha
4. Cambiar el color del led

De estas acciones la única que debe llevar algo más de información a parte de la acción en sí, es el cambio de color. Por lo que se ha decidido que el sistema de comunicaciones va a enviar siempre primero la acción seguido en caso de ser necesario, de un valor/valores que acompañe/en a la acción.

Para ahorrar en tiempo y espacio, dichas acciones van a ser codificadas usando un **Enum** que será común tanto en la clase de la Tropa del sistema centralizado como en el robot.

ArUco

Se ha decidido usar la implementación de los marcadores **ArUco** que viene en **OpenCV**, esto permite desde la creación de los diccionarios de los marcadores hasta la detección y clasificación de los marcadores junto a sus coordenadas.

VideoSource y VideoPlayback

Para la entrada de video se ha decidido usar la librería de OpenCV la cual tiene ya implementados los métodos que necesitamos para el sistema central. Además, se ha creado un Mock que simula un frame con el fin de poder simular la entrada de video.

En el caso de la salida de video, usamos nuevamente la librería de OpenCV para mostrar los frames que vamos obteniendo desde la entrada de video.

Aprovechando la funcionalidad que da OpenCV, la salida de video puede dibujar por encima del frame formas simples (líneas, rectángulos, círculos y texto).

CommandManager y Command

Para facilitar la gestión de las acciones de los robots, se ha creado un gestor de comandos y los comandos. El gestor de comandos gestiona los comandos mediante una lista de estos. Se organiza utilizando el método **FIFO** (First In First Out) donde se ejecutarán a la vez todos los comandos que no vayan dirigidos a la misma Tropa y sean del mismo tipo.

Tropa	Tipo Comando	Estado
0	Go_To_Position	Ejecutando
0	Go_To_Position	En Espera
0	Go_To_Position	En Espera
0	Cambiar_Color	Ejecutando
1	Go_To_Position	Ejecutando

Tabla 2: Ejemplo de la cola FIFO del CommandManager

Los comandos que se han creado se dividen en dos categorías, el de ir a una posición y cambiar de color.

Movimiento

Uno de los problemas que nos encontramos al comenzar con este proyecto es la toma de decisión sobre qué movimiento deberá hacer el robot para llegar a su destino. En el caso que se ha planteado en el proyecto, el mundo tiene dos dimensiones, conocemos tanto las coordenadas del punto al que queremos ir, como la posición y orientación del robot (esto último gracias a los marcadores ArUco).

Con estos datos podemos aplicar un algoritmo determinista que haga lo siguiente:

1. Obtener la posición destino
2. Obtener la posición y orientación actual
3. Comprobar si el robot tiene enfrente o a su espalda el destino
4. Si lo tiene, avanzar hacía el en línea recta bien moviéndose hacía adelante o hacía atrás.
5. Si no lo tiene, girar a la izquierda o derecha dependiendo de que giro le vaya a costar menos hasta tenerlo de frente.

Lo complicado de este algoritmo es el **cálculo de las orientaciones**, así como el ajuste del **umbral** para cambiar la acción. Este umbral es el que va a **evitar** que podamos caer en **bucles infinitos** debido al movimiento no preciso del robot.

Para calcular primero la orientación del robot respecto al Frame, se ejecuta la siguiente formula:

$$Orientación\ del\ Robot = (360 - ((atan2(x_{Centro} - x_{Frente}, -(y_{Centro} - y_{Frente}))) * \frac{180}{\pi}))$$

El cálculo de las posiciones (x, y) tanto del centro como del frente del robot es trivial, ya que, el marcador ArUco devuelve las posiciones de las cuatro esquinas ordenadas.

El cálculo del Angulo que hay entre el robot y el punto se calcula de esta forma:

$$Angulo\ Robot - Destino = (360 - ((atan2(x_{Objetivo} - x_{Real}, -(y_{Objetivo} - y_{Real}))) * \frac{180}{\pi}))$$

A los resultados hay que llevárselos al **rango de 0 – 360** y **sustraerles 90** para compensar el cambio de plano que efectúa OpenCV en las imágenes.

A continuación, se va a explicar con un ejemplo el proceso del algoritmo de movimiento de los robots.

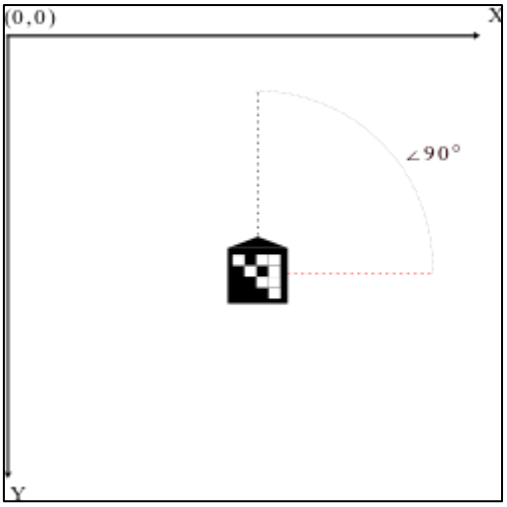

Figura 18: Primer paso de la explicación del movimiento de los robots

En la Figura 18 se puede ver al robot quieto sin ningún objetivo. Utilizando la fórmula para calcular la **orientación del robot**, obtenemos que está mirando a 90 grados.

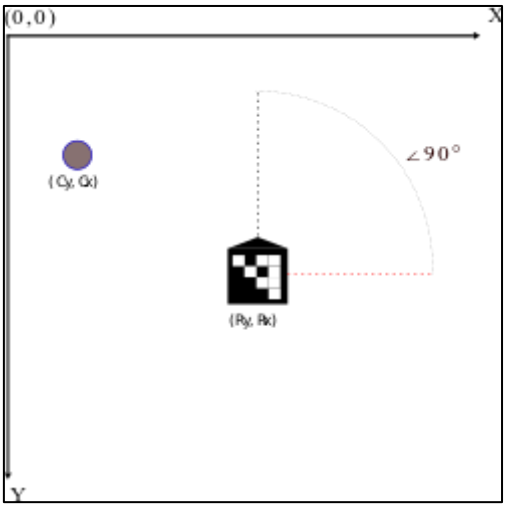

Figura 19: Segundo paso de la explicación del movimiento de los robots

Ahora se le ha generado un destino al robot, el cual es un punto que se encuentra en las coordenadas (Cx, Cy) y el robot está en (Rx, Ry).

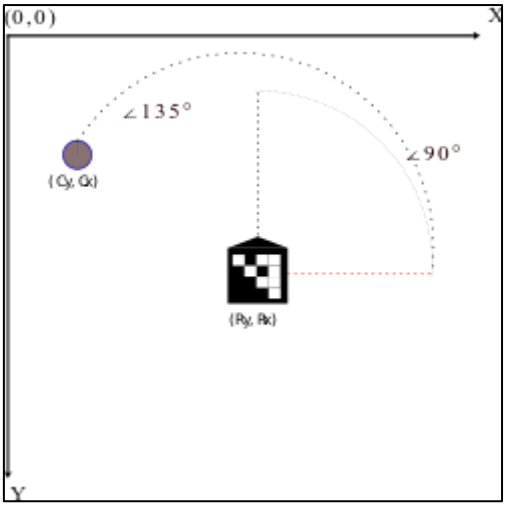

Figura 20: Tercer paso de la explicación del movimiento de los robots

Utilizando la fórmula para calcular el Angulo que forman el robot con el destino, obtenemos que el punto está a 135 grado. Eso significa que la diferencia entre los ángulos es de 135-90 = 45. Suponiendo que el robot previamente se estuviese moviendo hacía adelante y que el umbral para él cambio de acción sea igual o inferior a 45, el robot se girara en sentido antihorario para tratar de posicionarse en frente del destino.

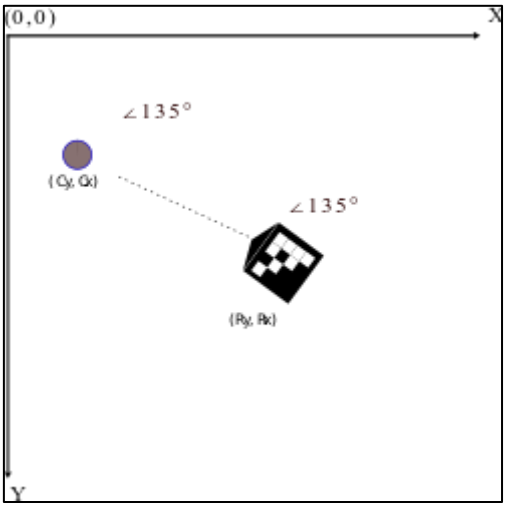

Figura 21: Cuarto paso de la explicación del movimiento de los robots

El robot ha girado a la izquierda (sentido antihorario) hasta posicionarse enfrente del objetivo.

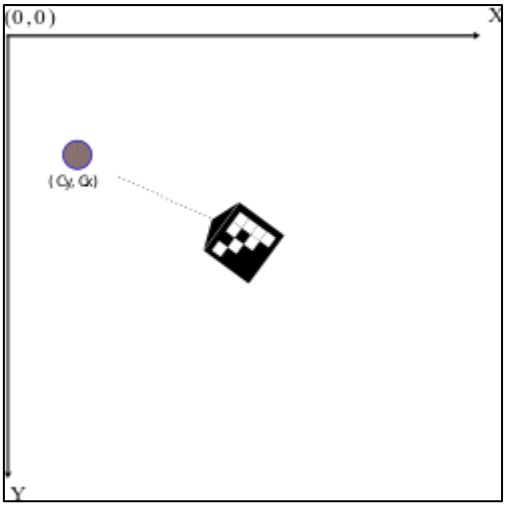

Figura 22: Quinto paso de la explicación del movimiento de los robots

Como el ángulo que forman el punto con el frente del robot es 0, esté avanza hacía el punto.

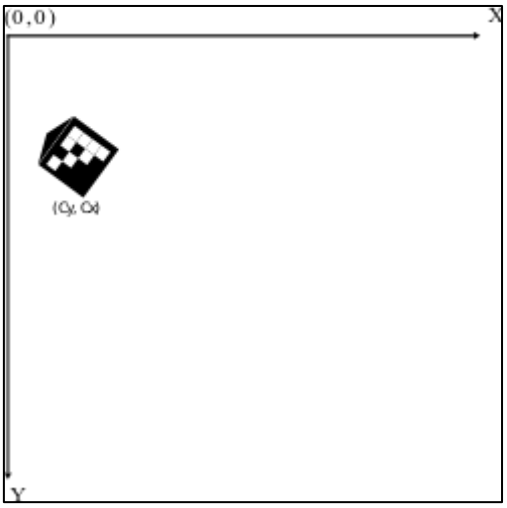

Figura 23: Último paso de la explicación

Por último, una vez que el robot llega al destino, el comando se deja de ejecutar y el robot se para.

Multihilo

Dada la naturaleza del proyecto, el coste computacional puede crecer rápidamente. Sin embargo, hay varias formas de afrontar este problema, una de ellas es el multihilo.

Como se puede observar en la Figura 4, hay cuatro grupos de “actores” en el sistema. Esto significa que podemos llegar a trabajar de forma separada en **cuatro hilos** diferentes, ya que, **ninguno** de los grupos **depende fuertemente de otro**.

De los cuatro grupos que hay uno de ellos **no tiene sentido paralelizarlo**, ya que, no interviene directamente en el sistema. Este grupo son las **Tropas** (robots), las cuales solamente envían comandos y no tienen coste computacional per se.

Los otros tres grupos pueden ser paralelizados, aunque realmente trabajen de manera **secuencial**. El flujo del programa es el que aparece en la Figura 24.

En dicho diagrama podemos observar cómo existe un tiempo de procesamiento en el actor **Aruco**, el cual no interacciona con **VideoPlayback**, por lo que al ejecutar en paralelo estos dos obtenemos un sistema más **responsive**.

Por otro lado, está el **CommandManager**, el cual hemos visto que no depende de ninguno de los actores del sistema.

El resultado final es que el sistema centralizado va a utilizar cuatro hilos, el principal, uno para el **VideoPlayback**, otro para **Aruco** y el ultimo será utilizado por el **CommandManager**

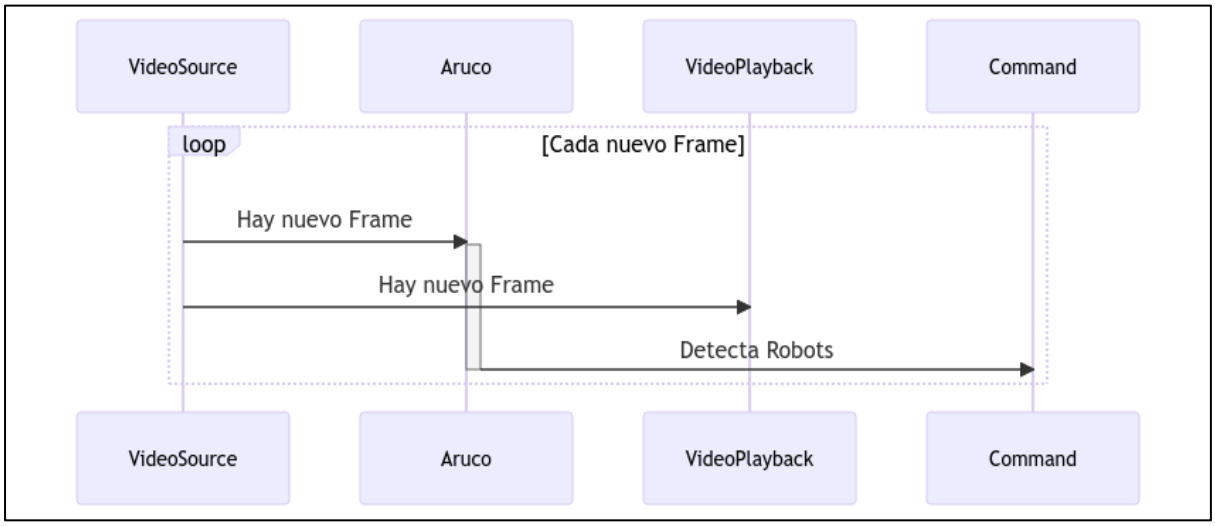

Figura 24: Diagrama de secuencia del funcionamiento del sistema central

Simulador

El simulador no es más que el uso de Mocks para simular las Tropas y el VideoSource.

Para ello se han creado las siguientes implementaciones:

- **FakeTropa**
- **FakeVideo**
- **FakeCommunication**
- **Command_Go_To_2D_Position_Fake**

La forma de simular que se ha implementado consiste en generar una matriz que simulará el **Frame** de la **WebCam**.

Las **Tropas** serán matrices que se pintan de forma programática y el movimiento de las tropas será el resultado de mover las matrices dentro del **Frame** simulado. Como se ha visto en el apartado de Movimiento, las **FakeTropas** tendrán un **umbral** de 90 grados debido a la naturaleza del **Frame**.

Esto da lugar al resultado de la Figura 25.

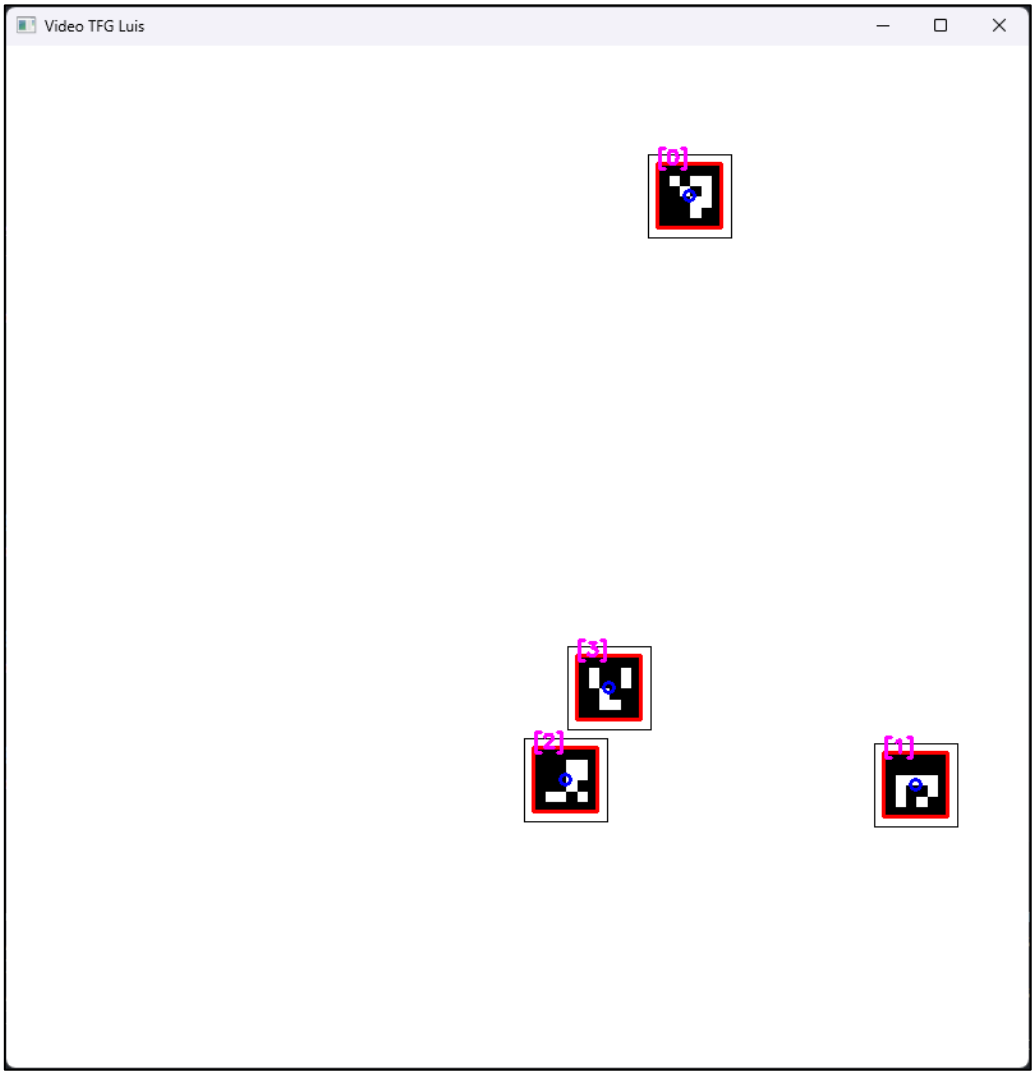

Figura 25: Simulador de Tropas

Gracias al **simulador** se ha podido **desarrollar y depurar de forma más eficaz** el proyecto. Alguna de las **limitaciones** del simulador son las siguientes:

- Rango de giro de las **Tropas** simuladas limitado a **90 grados**
- **Movimiento exacto**
- **Coste computacional**

Líneas futuras

Debido a la naturaleza del proyecto, los caminos por los que continuar son muy variados, pero se pueden definir en dos grandes categorías, **Software** y **Hardware**.

Dentro del apartado Software aparecen las siguientes propuestas:

- Creación de comandos que aporten nuevos comportamientos, como, por ejemplo, que todos los robots se reúnan en un punto
- Espacio de configuraciones para hacer más preciso el movimiento del robot
- Algoritmo de Path Finding que permita a los robots esquivar obstáculos
- Mapeado dinámico de la zona de forma que los robots sean capaces de recordar que caminos han ido tomando
- Simulador en Unity u otro motor para mejorar las capacidades de depuración e implementación del proyecto.
- Añadir de manera dinámica nuevos robots según vayan apareciendo en el plano y eliminarlos cuando salgan del mismo (posibilidad de encadenar varias cámaras una detrás de otra)

Por su parte, dentro del Hardware se propone continuar con las siguientes:

- Minimizar el tamaño del robot
- Pasar de un robot que se mueve en 2D a uno que se mueva en 3D
- Mejorar el consumo del robot
- Explorar otras formas de alimentación (energía solar, por ejemplo)
- Explorar otras formas de posicionamiento (odometría, uso de balizas, etc.)
- Pasar a una arquitectura jerárquica
- Pasar a una arquitectura descentralizada
- Pasar a una arquitectura hibrida

Bibliografía y referencias

https://docs.opencv.org/4.x/

https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

https://docs.opencv.org/4.x/db/deb/tutorial_display_image.html

https://es.wikipedia.org/wiki/Inteligencia_colectiva

https://github.com/Wonnie180/Trabajo_Fin_Grado

Anexos