TRABAJO FINAL DE CARRERA: DISEÑO DE SOFTWARE DE ROBOT MÓVIL UTILIZANDO ARDUINO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Realizado por Daniel Soto Hernández

Tutorizado por Leopoldo Armesto, Andrés Conejero y Miquel Cañada

Curso académico 2020/2021

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

RESUMEN

A lo largo de este documento se va a exponer como se ha planteado el diseño de un robot móvil de cuatro ruedas, con configuración diferencial, construido a partir de electrónica económica accesible para todo el mundo sobre una estructura dada por terceros, con el fin de ser un prototipo que simulara un producto comercial, con una carcasa que dejaría inaccesible la electrónica en su interior.

Dentro de las funcionalidades a realizar se encuentra el modo sigue-líneas, un modo autónomo evita obstáculos, la posibilidad de controlar las matrices leds que constituyen sus ojos y un control remoto manual.

Para ello se ha planteado una estrategia que engloba y relaciona tres puntos clave que componen el robot:

Software, en la parte de la App para el móvil con el que podremos manejar el robot de forma inalámbrica utilizando Bluetooth, donde se han implementado diferentes menús de control, uno para cada funcionalidad del robot, con un planteamiento muy intuitivo.

Firmware, programando el microcontrolador que compone el cerebro del robot a través de Arduino, éste será el encargado de controlar todos los sensores y actuadores que componen al robot (servomotores, matrices de leds, ultrasonidos, etc.), crear un servicio Bluetooth capaz de conectar con un smartphone y gestionar el flujo de comandos y algoritmos necesarios para cumplir con las funcionalidades

Hardware, en el seleccionado de componentes, testeo y conexionado, además de la colaboración en el ajuste de detalles del diseño planteado.

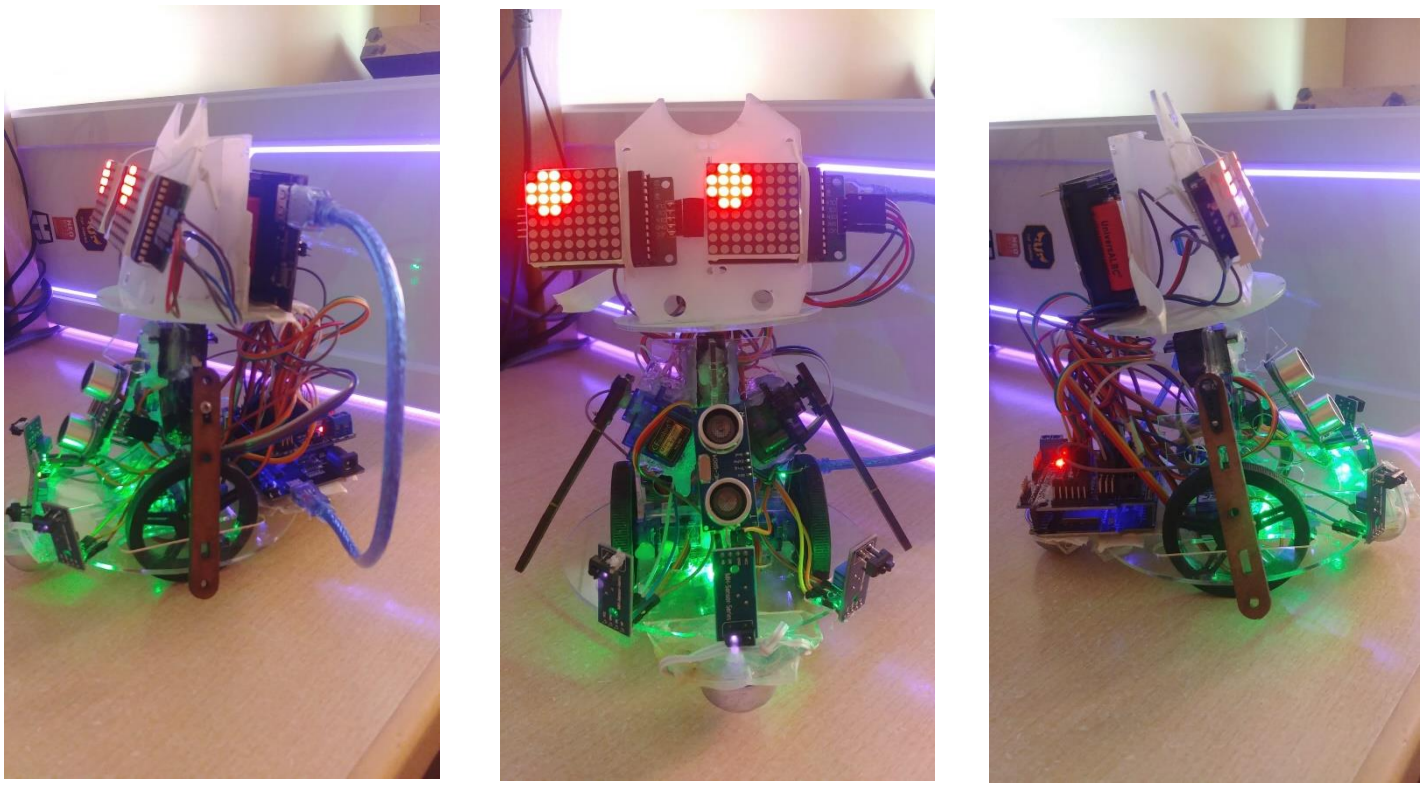


Ilustración 1: Presentación robot completo.

INDICE

BLOQUE I: MEMORIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo final de carrera

Realizado por Daniel Soto Hernández

Tutorizado por Leopoldo Armesto

Curso académico 2020/2021

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

I. TABLA DE ILUSTRACIONES

I. INDICE DE TABLAS

I.1. OBJETO

El trabajo a desarrollar consiste en el diseño, testeo, montaje y puesta en marcha de un robot móvil de bajo presupuesto, que sea controlado de forma remota a través de un smartphone y que se adecue al diseño estructural y necesidades dadas por terceros.

La parte del diseño que concierne a este trabajo es el seleccionado de la electrónica a utilizar y la programación del software de la aplicación móvil, así como la programación del firmware. Queda excluido por tanto el diseño de la estructura del robot, el cual es realizado por terceros.

I.2. ANTECEDENTES

La robótica está cada vez más integrada en nuestros hogares, presente de formas a veces más evidentes que otras, ya pueda ser en forma de aspiradora, electrodoméstico o incluso juguete.

La presencia de microcontroladores en los objetos más cotidianos del hogar, así como mi curiosidad por la programación y la informática industrial impulsaron mi deseo de profundizar en este ámbito con la realización de un proyecto que esté relacionado, pero en un sector no industrial, de forma que me pueda servir como punto de partida para la comprensión de cómo se desarrolla un producto, para posteriormente extrapolar la información en los futuros proyectos de mi carrera profesional.

La propuesta de realizar este TFG viene dada por un grupo de ingenieros de desarrollo de productos, los cuales me presentan el chasis que habían diseñado para un robot móvil humanoide y me exponen su deseo de crear un proyecto lo más cercano posible a un producto comercial, es decir que la electrónica y cableado no estén a la vista, ni se tenga fácil acceso a ellas desde fuera, estando encerradas en una carcasa atornillable. Además, en la estructura dada solo hay presente un botón, necesario para conectar y desconectar la batería, por lo que el control de los diferentes modos o funcionalidades del robot debería de ser remoto.

I.3. ESTADO DEL ARTE

Durante la realización de este proyecto el punto de investigación que más he desarrollado ha sido todo lo relacionado a los protocolos de comunicación inalámbricas, conocer las diferentes opciones posibles entre Wifi y los diversos métodos de Bluetooth. En ambos protocolos se transfieren datos entre dispositivos, en el caso del Bluetooth los datos se envían y reciben a través del puerto de comunicaciones serie del microcontrolador.

En cuanto a la placa, una imitación de la ESP32, “Wemos D1 Wifi and Bluetooth”, que comparte todas las características, se han descubierto todas las posibilidades que ofrece, como integra el “BLE” sin necesidad de módulos externos, igualmente ocurre con el Wifi, dispone de variedad de puertos tanto analógicos como digitales. Dispone de un procesador más potente en comparación a microcontroladores de precio similar. En definitiva, se trata de una placa que lo tiene todo, haciendo infinita la variedad de proyectos que se pueden realizar con ella, crear servidores web, recoger información de internet, conectarse con cualquier dispositivo Bluetooth, crear tu propia red privada de ESP32 y más.

La sencillez que ofrecen diferentes páginas web para realizar apps de móvil online, sin necesidad de descarga era un campo desconocido, en esencia ofrecen métodos de programación orientada a objetos, que a su vez se organizan en bloques, lo que simplifica aún más la tarea al programador.

I.4. FACTORES A CONSIDERAR

I.4.1 NECESIDADES DE PRODUCTO

I.4.1.1. FACTORES TÉCNICOS

El robot, al ser móvil debe de tener autonomía en cuanto a fuentes de alimentación fijas, por lo que funcionará a partir de una batería.

Robot en configuración diferencial, lo que supone dos ruedas móviles junto con sus motores y al menos una rueda libre.

Movilidad de tres articulaciones: sus dos brazos y cuello, con el único objetivo de darle un aspecto humanoide, por lo que los motores encargados de esos movimientos no tendrán apenas carga adicional, únicamente será necesaria la suficiente potencia como para mover el propio peso de los brazos y cuello respectivamente, fabricados en plástico.

Control de expresiones del robot a través de displays o pantallas, simulando los ojos de este.

Control remoto para gestionar las diferentes funcionalidades del robot, por lo que será necesario un método de comunicación inalámbrica y un mando o herramienta de control.

Desarrollo de las diferentes funcionalidades dichas, entre ellas: modo sigue-líneas, modo autónomo evita-obstáculos, modo control remoto y modo control de expresiones.

Únicamente botón para conectar o desconectar la alimentación a nuestro sistema.

I.4.1.2. FACTORES ECONÓMICOS

Al tratarse de un robot de propósito educativo, que podría estar presente en el ámbito doméstico y sin requerimiento de una gran potencia ni precisión no será necesario contar con una electrónica de mucha calidad, sino que las opciones de bajo coste serán más interesantes para obtener un producto que cumpla las funciones básicas de forma aproximada y facilite la competitividad de precio en caso de salir al mercado.

I.4.2. LIMITACIONES Y CONDICIONAMIENTOS

El proyecto se podría categorizar como un producto electrónico de emisión de radio frecuencia debido a que utiliza Bluetooth, que trabaja con el estándar de conexión inalámbrica IEEE 802.15, el cual permite la transmisión de voz y datos entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM de 2,4 GHz, según dicta la norma EN 300 328.

Categorizando nuestro proyecto como como un producto electrónico de emisión de radio frecuencia debemos de considerar el **Real Decreto 188/2016, de 6 de mayo**, por el que se aprueba el Reglamento por el que se establecen los requisitos para la comercialización, puesta en servicio y uso de equipos radioeléctricos, y se regula el procedimiento para la evaluación de la conformidad, la vigilancia del mercado y el régimen sancionador de los equipos de telecomunicación.

Siguiendo con el ANEXO I el cual dicta que equipos no están sujetos a las preinscripciones del reglamento, acogiéndonos a las definiciones del punto 1.a y 1.c donde se excluyen equipos electrónicos radioeléctricos que sean construidos por radioaficionados particulares con fines experimentales y científicos o que sean montados por radioaficionados a partir de kits.

Al tratarse de un producto de propósito general, simplemente un proyecto destinado a la evaluación de un TFG, la experimentación y el aprendizaje, sobre éste queda excluida de aplicarse esta normativa, siendo la única relacionada.

I.5. PLANTEAMIENTO DE SOLUCIONES ALTERNATIVAS

I.5.1 SISTEMA DE COMUNICACIONES

I.5.1.1. BLUETOOTH CLÁSICO

Sistema de comunicaciones inalámbrica orientado a intercambiar y manejar gran cantidad de datos, a cambio de un consumo de energía superior al “BLE”, lo que no lo hace el más adecuado para nuestra aplicación ya que el Bluetooth siempre estará activo y nuestro robot se alimentará a partir de baterías.

Este tipo de protocolo de comunicaciones tiene aplicaciones como la conexión entre dispositivos de audio, altavoces y auriculares inalámbricos.

I.5.1.2. BLUETOOTH LOW ENERGY

A pesar de compartir nombre con el clásico, no son compatibles entre ellos ya que disponen de protocolos completamente diferentes. Es utilizado para manejar cantidades de datos más reducidas a la misma velocidad, pero con un consumo más reducido. A partir de la versión 4.0 los módulos Bluetooth tienen ambos modos incorporados.

Esta presente de forma frecuente en dispositivos “wereables” y relacionados con el “loT”, dispositivos que funcionan con una pequeña pila, como termómetros y sensores.

I.5.1.3. WIFI

Este método de comunicaciones posee un ancho de banda superior a al Bluetooth, además de mayor velocidad en la transmisión de datos y más alcance efectivo. El consumo de energía con el uso de este protocolo es muy superior al Bluetooth.

Como desventaja hay que apuntar que el uso del robot se restringiría a puntos con una conexión Wifi, procedente de un rúter cercano. Además, no se aprovecharía todas las ventajas de este método, ya que no necesitamos tanto ancho de banda ni gran velocidad de transmisión para nuestra aplicación.

I.5.2. PLACA DE DESARROLLO

Para el control del robot y el firmware de la electrónica es necesario un microcontrolador. Entre tanta variedad existente hoy en día se requiere de uno que tenga compatibilidad con algún método de comunicaciones inalámbrica y una capacidad de procesamiento y memoria razonable, para llevar tareas a cabo como el sigue-líneas con cierta precisión, mientras continúa escuchando el puerto de comunicaciones inalámbricas, además de no sobrecargar el almacenamiento debido a la cantidad de código y variables.

Por otro lado, se tiene que encargar de distribuir la potencia suficiente a los diversos actuadores, los de mayor consumo son: 5 servomotores, 2 de ellos encargados de las ruedas por lo que pasarán

la mayor parte del tiempo en funcionamiento y con una carga significativa y dos matrices de iluminación para dar forma a los ojos del robot.

I.5.2.1. ARDUINO UNO

Arquitectura basada en Atmega328, destaca por su versatilidad y compatibilidad con módulos periféricos. Una de sus ventajas se ve reflejada en la facilidad de programación usando Arduino y en la variedad de ejemplos, tutoriales y foros orientados a esta placa.

En su contra para la realización de este proyecto estarían su velocidad de reloj de 16 MHz, su EEPROM de 1Kb, su SRAM de 2Kb y su corriente máxima de salida total entre los pines, 300mA, la cual podría condicionar el funcionamiento de los actuadores en momentos críticos, donde todos estén funcionando simultáneamente. Por otro lado, no dispone de conexiones inalámbricas, aunque podrían añadirse módulos con tal fin.

I.5.2.2. ESP32

Basada en el procesador y arquitectura ESP32, con una potencia/precio superior a la placa de Arduino, ya que a pesar de compartir el precio dispone de un reloj de 80MHz, lo que significa 5 veces más velocidad y mantiene su compatibilidad con Arduino. Integra Wifi y Bluetooth sin necesidad de periféricos. En cuanto a la memoria posee una SRAM de 520Kb y diversas subdivisiones más respecto a su competidora, dedicando cada una de ellas a una tarea más específica en el funcionamiento de la placa, lo que aumenta la velocidad de lectura/escritura de datos. Ofrece una corriente de salida por pin de 12mA, sin máximo total entre la suma de pines, lo que nos permite llenar los pines de pequeños actuadores, la potencia máxima que puede ofrecer son 4W.

Como desventajas, dispone de sus propias librerías, pudiendo variar el funcionamiento en alguna parte y no se trata de una placa de desarrollo tan simple como su competidora.

I.5.3. MOTORES

I.5.3.1. MOTOR CORRIENTE CONTINUA

Los motores de corriente continua son una solución analógica controlados únicamente por la alimentación que se le suministra. Hay de diversas potencias, capaces de mover cargas considerables.

Realizar un control preciso de estos motores requeriría de hacer unos cálculos previos para traducir voltaje de alimentación suministrado con revoluciones por segundo dadas, no sería posible controlar los grados desplazados por los motores, habría que añadir un encóder a cada motor y aun así no dispondríamos de precisión exacta.

I.5.3.2. MOTOR PASO A PASO

Motores ideales para aplicaciones de baja velocidad y aceleración y poca precisión, debido a que no es posible controlar los grados desplazados. Para el control de estos actuadores se debería de realizar desde el software, ofreciendo alimentación al par de polos correspondiente en cada momento para hacerlo girar, lo que consumiría recursos del procesador. Son los más económicos a cambio de un bajo rendimiento.

I.5.3.3. SERVOMOTOR

Solución normalmente aplicada a brazos robots, donde la precisión del movimiento desplazado de cada servo es fundamental para controlar la posición final del actuador. Tienen gran rendimiento además de un fácil control a través de librerías, lo que descargan trabajo a nuestro microcontrolador. Los servos contienen en su interior un motor de corriente continua, un sistema de engranajes, un controlador integrado y un sensor de posición.

Los hay muy económicos con un par muy reducidos, suficientes para mover las articulaciones sin apenas carga de nuestro proyecto, en un rango de 0º a 180º grado a grado.

Como alternativa para las ruedas existen los servomotores 360, los cuales permiten una rotación continua. A través de librerías es posible controlar la velocidad de giro.

I.5.4. SENSORES

En términos generales no se requiere de ningún sensor con especificaciones restrictivas debido a que nuestro proyecto es de aplicación doméstica. Por tanto, en términos de sensorización optaremos por las opciones más económicas.

Con el fin de implementar el sigue-líneas comúnmente se utiliza 1 o 2 sensores de infrarrojos, dependiendo del grosor de la línea y de si el circuito a recorrer tiene curvas cerradas. En el caso del modo autónomo del robot la opción más viable es un sensor de ultrasonidos, polivalente con un rango de detección suficiente para situaciones indoors.

I.5.5. DISPLAYS

A partir de algún tipo de pantallas se pretende simular unos ojos para el robot, los cuales puedan mostrar diferentes dibujos o formas, y controlar la expresividad de éste

I.5.5.1. DISPLAY LCD

Una posible solución se trata de imprimir a través de un display lcd, los cuales son capaces de configurar pixeles sobre una luz reflectora. Suelen estar orientados a imprimir texto, aunque existe la posibilidad de crear caracteres personalizados parea poder imprimir variedad de dibujos y formas. Destacan por su bajo consumo de energía y bajo precio.

I.5.5.2. MATRIZ DE LEDS

Alternativamente se presentan las matrices de led, las cuales son capaces de encender cada led de forma independiente, dando más flexibilidad a la hora de formar dibujos. Es posible controlar el nivel de brillo y los hay de diferentes colores, por lo que a nivel visual resulta más impactante. Tienen una vida útil muy larga, gran eficiencia y no disipa apenas calor.

I.5.6. SOFTWARE

Para la realización de la aplicación móvil se requiere que sea capaz de escanear y conectarse al dispositivo que se desea y una vez conectado que tenga la capacidad de implementar botones que se traduzcan en un envío vía inalámbrica de un comando a nuestro robot, posteriormente este interpretará el mensaje para realizar la acción correspondiente.

I.5.6.1. APP INVENTOR II

Incorpora métodos de comunicación con Wifi, Bluetooth Clásico y Low Energy, se trata de una aplicación sencilla que permite una programación orientada a objetos organizada en bloques, de esta manera se puede distribuir a lo largo de nuestra pantalla móvil diferentes elementos como botones, listas, etiquetas, etc. los cuales podemos hacer que interactúen de diferentes maneras al ser utilizados. Es una aplicación con largo recorrido por lo que es fiable y los ejemplos foros y tutoriales son de fácil acceso.

I.5.6.2. THUNKABLE X

Se trata de una aplicación muy similar a “App inventor 2”, la cual lo lleva a un paso más e incorpora avanzados servicios externos, los cuales son fácilmente incorporables como el control por voz o la IA. No se incorpora Bluetooth clásico.

I.5.6.3. ARDUINO

Enfocado a la parte de programación de microcontroladores, Arduino es una solución polivalente en lenguaje C y C++, la cual engloba una gran cantidad de placas de desarrollo y un administrador de bibliotecas que permite gestionarlas cómodamente, todas in situ en el programa.

I.5.7. ALIMENTACIÓN

Disponemos de diferentes formas de alimentar nuestro sistema, desde utilizando una power bank estándar para móviles de 5V conectada al puerto usb de nuestro microcontrolador o bien utilizando baterías de 3.7V conectadas directamente a los pines de alimentación del microcontrolador. La placa funciona a 3.3V, y ambas alimentaciones son igual de correctas.

Por otro lado, se debe de pensar en la autonomía de la que se quiere disponer, la placa únicamente haciendo la gestión Bluetooth consume alrededor de 200mA más la corriente que puedan consumir los periféricos, los únicos realmente significativos serán los motores de las ruedas, ya que tendrán que soportar la carga del robot completo, su consumo rondará los 200mA por motor.

Tanto las matrices leds, las cuales rondarán los 120uA cada una, tanto los servos, los cuales no tienen carga apenas y no están en continuo movimiento, así como los sensores no significarán un gran consumo para las baterías.

En conclusión, si queremos disponer de una autonomía aceptable y poniéndonos en la situación más desfavorable, donde nuestro robot consuma 1A, necesitaremos aproximadamente una batería que ofrezca a partir de 3000mA, para tener como mínimo dos horas de uso sin necesitar carga. Además, cumpliremos con el límite máximo de potencia para alimentar a nuestra placa el cual son 4W, siguiendo la premisa planteada nuestra potencia consumida serán 3W en el caso más exigente. El consumo real medido con el robot ensamblado rara vez supera los 700mA.

I.5.7.1. MÓDULO 18650

Este módulo se compone de dos baterías de 9.800mA en paralelo, lo que haría disponer de una gran autonomía al robot, a cambio de comprometer un poco el espacio debido a su tamaño, la única forma de la que dispone para alimentar nuestro micro es a trasvés de un cable usb de salida. Su peso podría comprometer el comportamiento dinámico de nuestro robot, sería necesario situar esta batería lo más cerca del suelo posible y alineada con el centro de gravedad de nuestro robot.

Ilustración 2:Módulo 18650 de baterías.

I.5.7.2. MÓDULO 16340

El módulo presentado se compone de dos baterías de 3000mA en paralelo, lo que nos supliría de una autonomía más que aceptable de aproximadamente tres horas, además su peso y tamaño es significativamente reducido comparado con el módulo anterior. Como punto a favor añadido tiene la posibilidad de ser conectado a nuestro microcontrolador a través de la disposición de pines (como alternativa al cable micro-usb), al igual que la “Arduino sensor shield”, lo que nos permitiría tener el microcontrolador, la batería y la placa de expansión todo en un mismo bloque, haciendo más sencillo el diseño y la distribución de la electrónica dentro del robot.

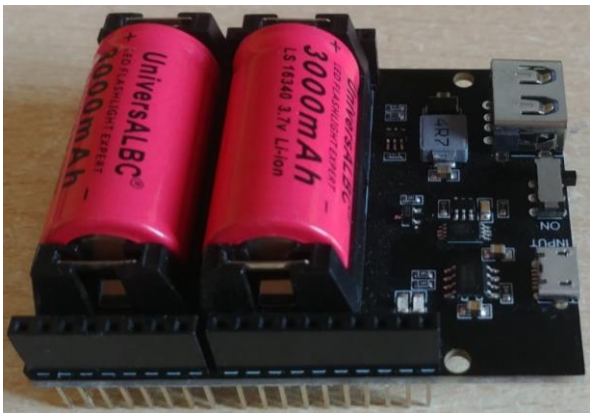


Ilustración 3: Módulo 16340 de baterías.

I.6. DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN ADOPTADA

I.6.1. ESP32

Debido a la clara superioridad en características relacionadas con potencia de procesamiento, integración de módulos de comunicación inalámbrica y su competitivo precio, la ESP32 se proclama como la mejor opción para resolver un proyecto de estas características.

Dentro de la variedad de versiones y fabricantes de placas de desarrollo con este procesador se ha elegido la “WeMos D1” (ver referencias en el “Anexo A: datasheets”) debido a su aparente similitud con el “Arduino UNO”, compartiendo su disposición de pines. También es posible programar esta placa desde el propio programa Arduino, el cual permite administrar las librerías propias de esta placa.

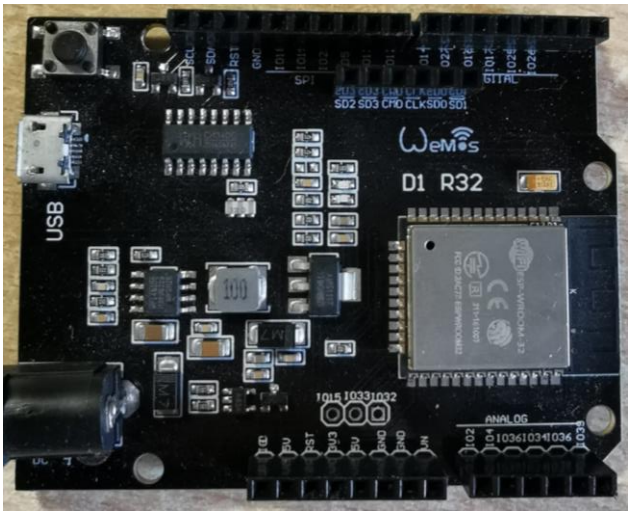


Ilustración 4: Placa “Wemos D1 Wifi and Bluetooth”.

Para organizar el conexionado de todos los puertos se ha utilizado una placa de expansión, la cual nos permite acceder a los puertos GPIOs tanto analógicos como digitales más cómodamente. Se ha utilizado la “Arduino Sensor Shield V.5.0”, compatible con la placa (ver referencias en el “Anexo A: datasheets”).

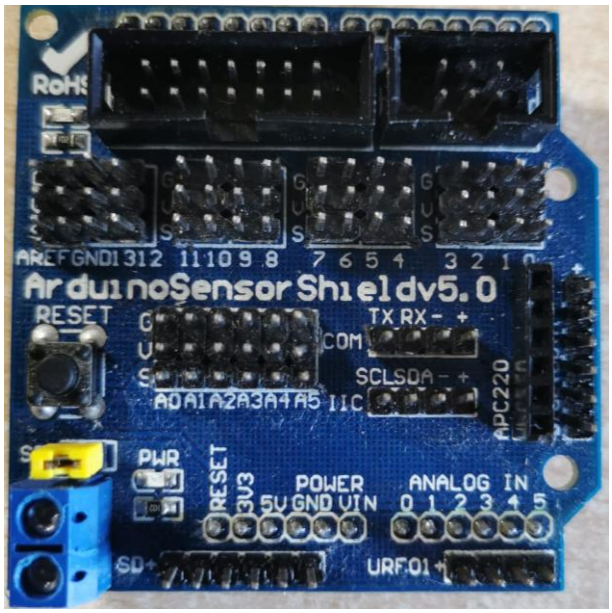


Ilustración 5: Placa expansión “Arduino Sensor Shield v.5.0”

Visualizando la Ilustración 2, se disponen a continuación la tabla de distribución de pines y su conexionado sobre la placa de expansión. Para referirse en el código a estos pines se debe de hacer una traducción, ya que los pines de la placa de expansión no corresponden con el mismo número de pines en el microcontrolador

Tabla 1: Mapa de conexionado de pines.

Pin microcontrolador	**Pin placa expansión**	**Dispositivo**	**Pin dispositivo**
16	**D5**	Matriz de leds	Din
27	**D6**		CS
14	**D7**		Clk
2	**A0**	Servo brazo izquierdo	
34	**A3**	Sensor ultrasonidos	Echo +Trigger

4	**A1**	Servo cuello	
18	**D13**	Sensor sigue-líneas	
39	**A5**	Sensor infrarrojos centro	
36	**A4**	Sensor infrarrojos D.	
24	**D3**	Sensor infrarrojos I.	
23	**D11**	Rueda derecha	
5	**D10**		
13	**D9**	Rueda izquierda	
19	**D12**	Servo brazo derecho	
18	**D13**	Sensor infrarrojo	

I.6.2. BLUETOOTH LOW ENERGY

El Bluetooth Low Energy recoge las características que más se aproximan a nuestro proyecto, es decir el manejo de pequeños datos, en nuestro caso únicamente se trata de caracteres individuales o como mucho en parejas, el bajo consumo energético, fundamental para nuestro proyecto ya que la placa permanecerá en escucha de forma permanente, debido a que es la única manera que tiene de recibir órdenes y ejecutar acciones. De esta manera podemos alargar la duración de nuestra batería y hacer nuestro robot más autónomo. Para implementar esta funcionalidad no será necesario añadir ningún módulo, debido a que viene integrado en nuestra ESP32, únicamente se debe de saber configurarlo.

Para explicar brevemente cómo funciona el protocolo Bluetooth se puede decir que tiene una estructura cliente/servidor, siendo el cliente el encargado de mandar inputs al servidor y recibir respuestas si fuese necesario, los teléfonos móviles son un ejemplo de clientes Bluetooth. Por otro lado, el servidor tiene la tarea de procesar esos inputs y convertirlos en respuestas o ejecutar las acciones ligadas, un ejemplo serían los dispositivos inalámbricos como ratones o auriculares y en nuestro caso particular el microcontrolador.

Estos servidores previamente citados se componen a su vez de atributos, los cuales tienen una estructura jerárquica y pueden ser de tres tipos: servicios, características o perfiles. Un servicio es un conjunto de características y declaraciones que se agrupan para satisfacer una necesidad. Siguiendo la línea jerárquica, las características representan información con el objetivo de ser expuesta al cliente e incluye de qué forma puede ser expuesta, lectura, escritura, notificación, etc.

Todos los atributos de un servidor deben de tener una UUID que los identifique, este identificador puede adoptar o bien un valor genérico para ciertas finalidades (GAP), el cual dará información al cliente de cómo proceder en el protocolo, por ejemplo, existe un identificador exclusivo para el intercambio de audio, o bien se puede crear un identificador propio. Una práctica común que

hemos seguido en nuestro proyecto es tras darle una UUID generada aleatoriamente a través de una web a nuestro servicio, y utilizar esa misma UUID variando un bit del 4º byte más significativo para las características contenidas en su interior.

```
SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

Ilustración 6: UUIID de los atributos Bluetooth.

Para integrarlo en el proyecto se crea un servidor abierto al que sea posible conectarse desde un móvil como cliente, dentro de éste se crea un servicio con UUID única, el cual a su vez se compone de una característica, la cual se habilita para que cuando se reciba un dato por el puerto en serie (proveniente del móvil a través del Bluetooth) se ejecute una función de Callback, dentro de esta función el microcontrolador se encarga de comparar el comando recibido con una lista propia, de forma que interpreta el comando para desarrollar la función ordenada.

En nuestro caso el servicio de recepción de datos se trata de una UART, ya que es capaz de cargar una secuencia de datos recibidas carácter a carácter por el puerto serie a un vector, el cual es utilizado posteriormente para identificar el comando usado. Es decir, permite el envió los datos en serie y los carga en un vector.

La forma de configurar el módulo Bluetooth es a través de las funciones de la librería “ESP32_BLE_Arduino[1]”, se muestran en la Tabla Nº3 las principales funciones utilizadas.

Tabla 2: Leyenda colores para tablas de funciones.

Tipo de variable Variable Función (Tipo de variable Variable); Tipo de objeto Objeto;

Tabla 3: Funciones principales librería ESP32_BLE_Arduino.

Lugar	Función	Parámetros
Inicio	BLEServer *NombreServidor; BLECharasteristic *NombreCharasteristic; BLEService *NombreService;	- NombreServidor: nombre del puntero a objeto de tipo BLEServer.

[1] Kolban, N. and Kliem, S., 2019. *Librería BLE*. [online] GitHub. Available at: < https://github.com/nkolban/ESP32_BLE_Arduino > [Accessed 14 March 2021].

	Crea un puntero a objeto de tipo BLEServer, BLECharasteristic y BLEService para posteriormente invocar sus funciones.	- NombreCharasteristic: nombre del puntero a objeto de tipo BLECharasteristic. - NombreService: nombre del puntero a objeto de tipo BLEService.

	Class NombreMyCallbacks: public BLETipoCallbacks {
	
	Void FuncionEvento (){}
	
	}
	
	Declaración de función Callback. En su interior se declarán las funciones evento que dispararán el Callback. En el interior de las funciones se encontrará el código a ejecutar cuando el evento ocurra.
		- NombreMyCallbacks: nombre que se ha dado al callback en la función setCallbacks().
		
		- BLETipoCallbacks: tipo de callback que va a ser lanzado, puede ser de Server o de Characteristic. Dependiendo del tipo tendrá diferentes funciones evento.
		
		- FuncionEvento: se refiere al tipo de evento que hará disparar el callback, en su interior se verá reflejado el código a ejecutar en caso de que el evento ocurra.

Setup()	BLEService *NombreService NombreServidor→createService (UUID); Crea un servicio dentro del servidor bluetooth.	-UUID: la UUID[2] de un servicio bluetooth es un número de serie, único para cada dispositivo. Es posible generar estos identificadores de forma online. -*NombreService: devuelve un puntero a objeto de tipo BLEService, inicializándolo dentro de su servidor.
	NombreServidor->setCallbacks (new MyServerCallbacks ()); Crea una función de Callback para el servidor. Esta función asíncrona será llamada automáticamente cuando un evento en el servidor de bluetooth ocurra, por ejemplo, en el caso de conectar/desconectar un dispositivo.	- MyServerCallbacks (): nombre de la función que queremos que sea llamada cuando un evento ocurra. Se deberá definir previamente y especificar que queremos que pase en su interior. Función clave en la gestión bluetooth.
	BLEDevice::init (NombreDispositivo); Crea un dispositivo bluetooth y le da nombre para posteriormente invocar sus funciones.	- NombreDispositivo: nombre que queremos darle a nuestro dispositivo bluetooth, será el nombre visible al escanear dispositivos.

[2] Uuidgenerator.net. 2021. *Online UUID Generator Tool*. [online] Available at: < https://www.uuidgenerator.net/ > [Accessed 14 May 2021].

	BLEServer *NombreServidor BLEDevice::createServer(); Crea el servidor.	-*NombreServidor: devuelve un puntero a objeto de tipo BLEServer, inicializándolo.
	BLECharasteristic *NombreCharasteristic; NombreServidor→createCharasteristic (UUID_Charasteristic, BLECharacteristic::PROPERTY_N) Crea una característica dentro del servicio del servidor.	-*NombreCharasteristic: devuelve un puntero a objeto de tipo BLECharasteristic, para posteriormente llamar al identificador PROPERTY_N. - UUID_Charasteristic: se tratará de un identificador para esta característica. - PROPERTY_N: identificador propio de la librería, dependiendo de cual se escoja se define la característica, ésta puede ser de lectura o escritura entre otras. Según que tipo escojamos nuestro dispositivo será capaz de leer o escribir valores.
	NombreCharasteristic→ setCallbacks (new MyCallbacks ()); Guardas la función de Callback correspondiente a la característica, esta será llamada automáticamente cuando un evento de la característica ocurra, por ejemplo si se ha creado una característica con PROPERTY_N= PROPERTY_WRITE, este MyCallbacks() es llamado cuando un dato haya sido recibido.	- MyCallbacks (): nombre de la función que queremos que sea llamada cuando un evento ocurra. Se deberá definir previamente y especificar que queremos que pase en su interior. Función clave en la gestión bluetooth.
	NombreServidor→start () NombreService→getAdvertising ()→start () Se lanzan todas las configuraciones preestablecidas con las anteriores funciones.	

A continuación, se muestra la parte del código del proyecto donde se aplica estas funciones para la configuración y gestión BLE, la cual es una estructura asíncrona debido a las funciones de Callback, funciones que son ejecutadas en el momento en el que un evento o bien del Servicio o bien de Característica ocurra. Estas funciones de Callback son las encargadas de avisar de que ha llegado información nueva vía puerto COM, lugar donde se reciben los datos en el protocolo Bluetooth.

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <sstream>
#include <string>

BLEServer *pServer = NULL;
BLECharacteristic * pTxCharacteristic;
bool deviceConnected = false;
bool oldDeviceConnected = false;
uint8_t txValue = 0;


#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

Ilustración 7: Código inicialización dispositivo BLE.

En el inicio del programa se incluyen las librerías, se crean los objetos necesarios para la configuración y se declaran las variables de las UUID para atribuirlas a nuestro dispositivo.

```
//Main part of funcionality in bluetooth
//Function that is called when a msg is received(rxValue),
class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();
      if (rxValue.length() > 0) {
        Serial.println("*********");
        Serial.print("Received Value: ");
        //Max rx.lenght are 22 characteres
        for (int i = 0; i < rxValue.length(); i++)
          Serial.print(rxValue[i]);
```

Ilustración 8:Código de la función de Callback para lectura de puerto BLE.

Se muestra la implementación de la función Callback perteneciente a la Característica de nuestro dispositivo dedicada a escuchar datos entrantes, por lo que en su interior leerá los datos disponibles en el puerto y los cargará a una variable. para posteriormente realizar un

procesamiento y compararla con los comandos asignados a funciones de nuestro robot, con el objetivo de ejecutar la acción correspondiente, en la Tabla Nº4 se representa toda la información referida a los comandos y las funciones que invocan.

```
// Create the BLE Device
BLEDevice::init("Nanobot jeje");

// Create the BLE Server
pServer = BLEDevice::createServer();
pServer->setCallbacks(new MyServerCallbacks());

// Create the BLE Service
BLEService *pService = pServer->createService(SERVICE_UUID);

// Create a BLE Characteristic
//First TX service for send data
pTxCharacteristic = pService->createCharacteristic(
                  CHARACTERISTIC_UUID_TX,
                  BLECharacteristic::PROPERTY_NOTIFY
                );
pTxCharacteristic->addDescriptor(new BLE2902());

//RX service for receive data
BLECharacteristic * pRxCharacteristic = pService->createCharacteristic(
                     CHARACTERISTIC_UUID_RX,
                    BLECharacteristic::PROPERTY_WRITE
                  );

pRxCharacteristic->setCallbacks(new MyCallbacks());

// Start the service
pService->start();

// Start advertising
pServer->getAdvertising()->start();
```

Ilustración 9:Código de funciones de BLE en setup().

En esta parte del **setup()** de nuestro programa se muestra la invocación de funciones a partir del objeto “pServer” creados en la inicialización, las cuales crean el Servicio y las Características con los parámetros UUID anteriormente definidos, además se definen los Callbacks correspondiente a cada uno. Posteriormente se lanza la configuración.

Estos es la estructura fundamental para configurar un dispositivo BLE, debido a que funciona de forma asíncrona no es necesario código en la parte **loop()** del programa. Ver Plano Nº1 para ver el diagrama de flujo de la estructura asíncrona.

I.6.3. SERVOMOTORES

Tras tener en consideración la ligera carga de las articulaciones del robot, que no tienen funcionalidad o requerimiento de precisión, la solución óptima es el uso de servomotores, debido a su facilidad de programación y su económico precio.

Para las articulaciones hemos elegido unos servomotores de 9g con modelo “SG90”, un servo pequeño y con poca potencia, mientras para las ruedas hemos elegido unos servomotores de rotación continua con modelo “FS90R” (ver referencias en el “Anexo A: datasheets”).

Ilustración 11: Servomotor “SG90”

Ilustración 10: Servomotor "FS90R"

Para realizar su control utilizaremos las librerías “ESP32Servo[3]”, la cual nos ofrece variedad de funciones que nos simplifican la tarea, aquí las más utilizadas:

Tabla 4: Funciones principales librería ESP32Servo.

Lugar	Función	Parámetros
Inicio	Servo NombreServo; Crea objeto de tipo servo para posteriormente invocar sus funciones.	- NombreServo: nombre del objeto clase Servo.
Funciones que	NombreServo.attach (int Pin, int minUs360, int maxUs360);	- Pin: número de pin donde está el servo a conectar.

[3] Harrington, K. and Bennet, J., 2020. *Librería ESP32Servo*. [online] GitHub. Available at: < https://github.com/madhephaestus/ESP32Servo > [Accessed 5 March 2021].

ordenan movimiento	“Conecta” el servo a su pin y ajusta parámetros.	- minUs360: mínimo ancho de pulso admitido. - maxUs360: máximo ancho de pulso admitido.
	NombreServo.write (int num); Ordena al servo moverse.	-num: en el caso de los servos es la posición en grados entre 0 y 180 a la que quieres llegar. En el caso de los servos continuos es la V a la que quieres que gire siendo: 0 máxima velocidad inversa, 90 servo parado, 180 velocidad máxima.
	NombreServo.detach (); “Desconecta” el servo. Útil para evitar movimientos indeseados por derivaciones.	

I.6.4. SENSOR INFRARROJO

Para la implementación del sigue-líneas es fundamental el uso de un sensor infrarrojo, el cual será el encargado de estar transmitiendo si el robot está situado encima de la línea o no, hemos utilizado el “TCRT5000” (ver referencia en el “Anexo A: datasheets”) debido a que es de los más sencillos y económicos. Para usarlo simplemente tendremos que leer el pin en el que esté conectado y posteriormente interpretar los valores devueltos.

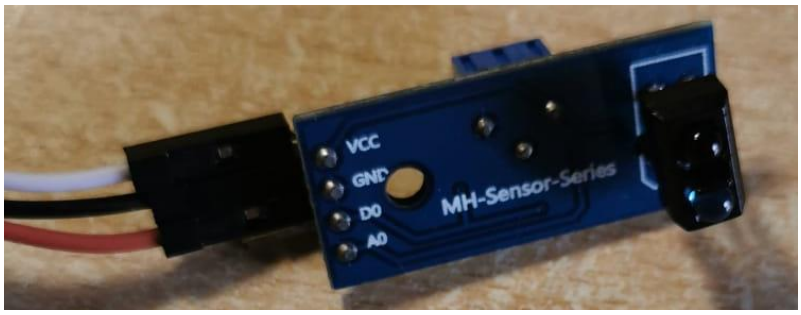


Ilustración 12: Sensor infrarrojo TCRT5000

Tiene dos modos de funcionamiento, dependiendo de si se conecta el pin analógico o el digital: conectando el pin analógico se obtiene un sensor que mide distancias de entre 1 y 10 centímetros,

devolviendo valores desde los 260 a 800 respectivamente, por otro lado si se conecta el pin digital se podrá leer medidas inferiores a 1 centímetro, las cuales corresponderán a un nivel alto si se trata de un color oscuro o un nivel bajo si es un color claro, debido a si la luz infrarroja rebota en la superficie o es absorbida.

Tanto como para el modo sigue-líneas como el autónomo es necesario “abandonar” el modo escucha activa de comandos entrantes Bluetooth, para pasar a un bucle repetitivo donde se ejecute el algoritmo a seguir.

Para ello en Arduino se crea un bucle cronometrado de 10s, es decir durante 10 segundos las funciones de Callback no serán disparadas en ese momento si un comando entra por el puerto de comunicaciones, por lo que los comandos se irán acumulando en el puerto y al salir del bucle se disparará la función Callback tantas veces como comandos hayan llegado. Esto podría saturar el canal e inducir a una pérdida de conexión Bluetooth, por lo que para acompañar esto, durante el tiempo que Arduino vaya a estar sin escuchar, la App bloquea las acciones que envíen comandos, y las libera al acabar el tiempo.

Además, desde la App se envía un nuevo comando cada 10 segundos, durante el tiempo total seleccionado. A través de esta estrategia impedimos que el canal se sature o se bloquee, y también nos permite cortar el algoritmo cada 10 segundos, ya que Arduino saldrá del bucle y solo hay dos posibilidades, que haya otro comando que le meta de nuevo en el algoritmo o que no lo haya. Ver Planos Nº3. En el Plano Nº7 se refleja el diagrama de flujo del algoritmo sigue-líneas.

I.6.5. SENSOR ULTRASONIDOS

Para la implementación del modo autónomo, en el que el robot tiene que ser capaz de detectar obstáculos a cierta distancia y tener la capacidad de esquivarlos utilizaremos un sensor “HC-SR04” (ver referencia en el “Anexo A: datasheets”), el cual tiene un rango de hasta medio metro teórico.

Ilustración 13: Sensor HC-SR04

En el caso de este sensor vamos a utilizar la librería “NewPing[4]”, la cual nos facilitará la lectura y gestión de el sensor. Se exponen aquí las funciones principales:

Tabla 5: Funciones principales librería NewPing.

Lugar	Función	parámetros
Inicio	NewPing NombreSensor (int TriggerPin, int EchoPin, int MaxDistance); Crea objeto de tipo NewPing e inicializa, para posteriormente invocar sus funciones.	- NombreSensor: nombre del objeto clase NewPing. - TriggerPin: pin donde está conectado el puerto de disparo del sensor. - EchoPin: pin donde está conectado el puerto de lectura del sensor. Estos dos últimos pines pueden ser el mismo. - MaxDistance: distancia máxima de lectura del sensor.
Función que realiza medida	Int medida NombreSensor.ping_cm() Función que devuelve la distancia medida en centímetros, devolverá la medida máxima si no se detecta nada.	-medida: distancia medida en centímetros.

Para desarrollar nuestro algoritmo evita obstáculos se implementará que el robot avance en línea recta siempre y cuando ningún objeto sea detectado por el sensor de ultrasonidos, en caso de que una medida sea inferior a una distancia cautelar, el robot retrocederá un poco y comenzará a girar hasta que encuentre una zona despejada.

Durante las pruebas de este sensor en el algoritmo evita obstáculos ser ha decidido complementarlo con tres sensores de infrarrojos en modo analógico, los cuales actúan en caso de que el ultrasonido no este midiendo correctamente, o bien porque no esté completamente en perpendicular apuntando hacia la superficie a detectar o porque se esté apuntando hacia una esquina, donde normalmente se toman medidas erróneas. De esta manera nuestro ángulo de detección aumenta considerablemente debido a la distribución de los sensores infrarrojos a los lados del robot.

[4] Eckel, T., 2014. *Librería NewPing*. [online] GitHub. Available at: < https://github.com/KurtE/NewPing > [Accessed 5 April 2021].

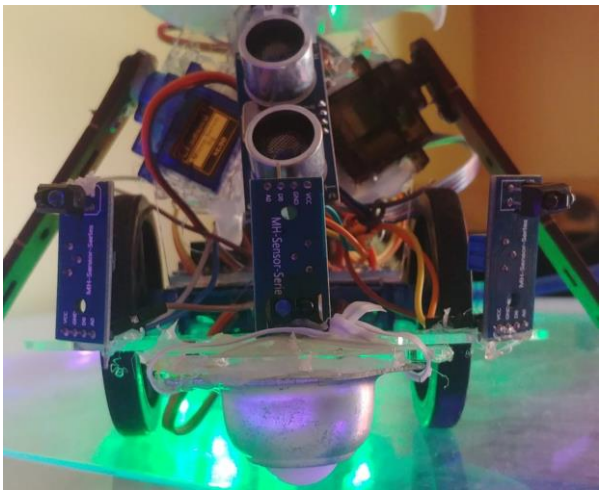


Ilustración 14: Sensorización final para modo autónomo.

I.6.6. MATRIZ LEDS

Con el objetivo de dar expresividad al robot, imitando unos ojos humanos con motivo estético, finalmente se han usado dos matrices de led de 8x8 con el integrado “MAX7219”, conectadas en cascada (ver referencia en el “Anexo A: datasheets”).

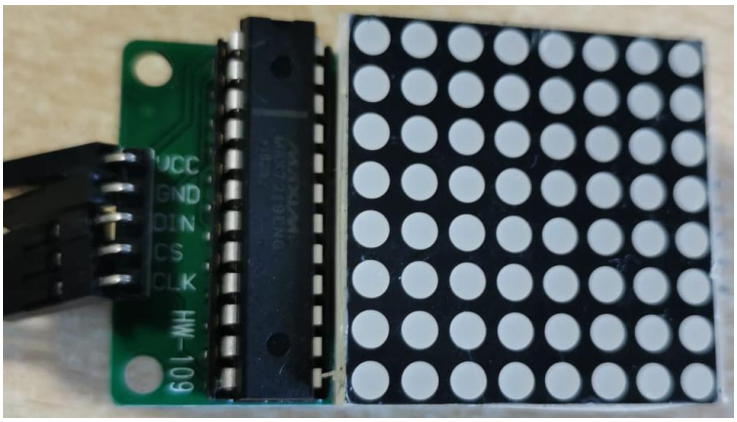


Ilustración 15: Matriz de led 8x8

Además, se ha implementado una funcionalidad en la que se puede imprimir frases por las matrices, de hasta 22 caracteres, las cuales deslizarán de un lado a otro con velocidad seleccionable, pasando por ambas matrices, esto se suma al conjunto de 8 dibujos para darle expresividad y personalidad al robot, los cuales parpadean de forma continua, independientemente en el modo en el que esté. Ver Planos Nº5 y Nº6 para ver las representaciones y las matrices de datos que las contienen.

Es decir, en cualquier modo en el que este el robot la app nos da la opción de modificar al tipo de ojo que queramos, el cual permanecerá parpadeando de forma continua, mientras que el modo de imprimir cadenas de texto es único y no puede ser compartido con otra funcionalidad.

Para facilitar el manejo de estos periféricos se ha utilizado la librería “MD_MAX72xx[5]”, la cual nos facilita la programación, la implementación de dibujos y acceder a diferentes funcionalidades, siendo las principales utilizadas en este proyecto:

Tabla 6: Funciones principales librería MD_MAX72xx.

Lugar	Función	parámetros
Inicio	MD_MAX72XX NombreMatriz MD_MAX72XX (MD_MAX72XX.type Hardware_type, int DataPin, int ClkPin, int NumofMatrix); Crea objeto de tipo MD_MAX72XX e inicializa, para posteriormente invocar sus funciones.	- NombreMatriz: nombre del objeto clase MD_MAX72XX, es decir la función de creación de objeto devuelve un objeto con la configuración indicada. - Hardware_type: tipo de hardware utilizado, se trata de un tipo de parámetro interno de la librería. - DataPin: pin donde está conectado el puerto data de la matriz. - ClkPin: pin donde está conectado el puerto clock de la matriz. - NumofMatrix: numero de matrices que vamos a conectar en cascada.
Función donde se dibujan formas	NombreMatriz.setRow(int NumofMatrix, int NumofRow, int RowData);	- NumofMatrix: número de matriz que queremos modificar. - NumofRow: número de columna que queremos modificar. - RowData: dato de 8 bits, representando un led por bit

[5] Colli, M. and Humfrey, N., 2021. *Librería MD_MAX72XX*. [online] GitHub. Available at: < https://github.com/MajicDesigns/MD_MAX72XX > [Accessed 20 April 2021].

		de la columna, siendo 1 encendido y 0 apagado.
Función donde se dibujan caracteres	NombreMatriz.setChar(int columna, char letra);	- columna: número de columna a partir del que se imprime el carácter. - letra: carácter a imprimir.
Setup()	NombreMatriz.control(MD_MAX72XX intensity, int IntensidadBrillo);	- intensity: caractrística propia de la librería. - IntensidadBrillo: intensidad con la que queremos que brille nuestras matrices, será un número entre 0 y 15.

A continuación, se van a mostrar las principales funciones implementadas para nuestro proyecto. Para ver el diagrama de flujo en detalle ver Plano Nº2.

```
unsigned long currentMilis=millis();
 if (currentMilis-previousMilis>=intervalPl){
   previousMilis=currentMilis;
   if(intervalPl==4000)
     intervalPl=500;
   else if(intervalPl==500)
     intervalPl=4000;
     if(TypeEye=='d'||TypeEye=='e'){
       }
     else if(SaveText==false)
       Parpadeo(intervalPl,TypeEye);
 }
```

Ilustración 16: Código gestión de parpadeo.

En este extracto del **loop()** se hace un cronometraje que se va comparando con dos periodos de tiempo, 4 segundo donde los ojos están abierto, contra medio segundo en el que están “parpadeando” y se llama a una función que gestiona el cambio de dibujo dependiendo que tipo de ojos hayan en ese momento gracias a una sencilla estructura de “swich case”. Hay que añadir que no todos los tipos de ojo parpadean.

```
void slide_text(int ms_delay,char text[], int num){
  int col = 0;
  int last_pos;
  bool complete = false;
  mx.clear();
  while( complete == false ){
    last_pos = printText(col, text,num);
    delay(ms_delay);
    col++;
    if( last_pos > (int)mx.getColumnCount() )
      complete = true;
  }
}
```

Ilustración 17: Código función “slide_text”.

Esta función se encarga llamar a la función “printText” en bucle con cierto delay(), que marca la velocidad de deslizamiento, la cual puede ser regulada, este bucle finalizará cuando el último carácter de la cadena de texto haya deslizado hasta el final.

```
int printText(int pos, char text[], int num){
  int w;
  for( int i = 0; i < num; i++ ){
    // imprimir letra
    w = mx.setChar( pos, text[i] );
    // la proxima letra empieza donde termina esta
    pos = pos - w;
    // Se deja una columna entre letras.
    mx.setColumn(pos, B00000000);
    pos = pos - 1;
    if( pos < 0 )
      break;
  }
  mx.update();
  return pos;
}
```

Ilustración 18: Código función printText.

Esta función se encarga de imprimir el carácter que toque de la cadena, poner una columna de espacio entre caracteres e ir deslizando todo lo que haya impreso una columna hacia un lado, es llamada de forma continua por “slide_text” hasta que se acabe la cadena.

I.6.7. BATERÍA

Finalmente hemos seleccionado el módulo ”16340” como fuente de alimentación para nuestro robot (ver referencias en el “Anexo A: datasheets). Debido a que la autonomía es correcta y nos da la posibilidad de hacer más compacto el diseño, aunque para ello se ha tenido que realizar las siguientes modificaciones de hardware.

Debido a que los pines de la placa de expansión machos no llegaban a insertarse en el circuito se han fabricado unos “alargadores” de estos pines, soldando una tira de pines macho y una de hembras.

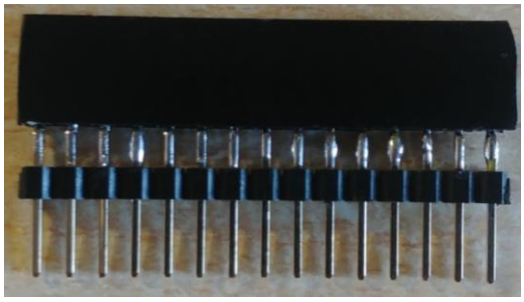


Ilustración 19: Alargador de conectores para la shield de Arduino.

Se muestran a continuación la problemática inicial y el resultado de la solución adoptada:

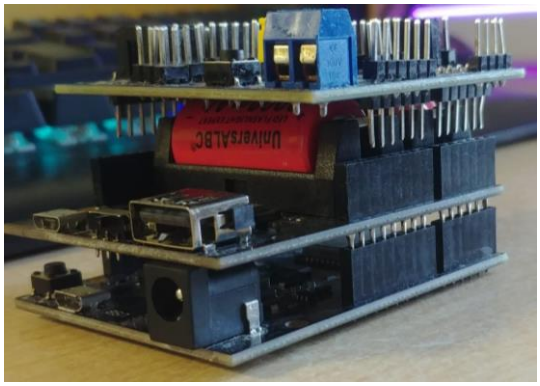


Ilustración 20:Problemática al conectar los tres módulos a través de sus pines.

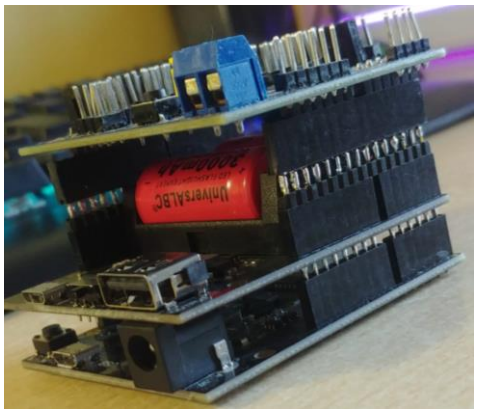


Ilustración 21: Resultado de la solución adoptada en el conexionado d ellos tres módulos.

A parte de esta forma de conexión entre la batería y el micro también se ha probado a hacerlo a través del cable micro-usb, con el objetivo de comprobar si la cabeza fuese capaz de soportar el peso de la batería sin que se comprometiese el comportamiento dinámico del robot, como alternativa en caso de que no sea posible físicamente emplazarlo todo junto en el interior del robot.

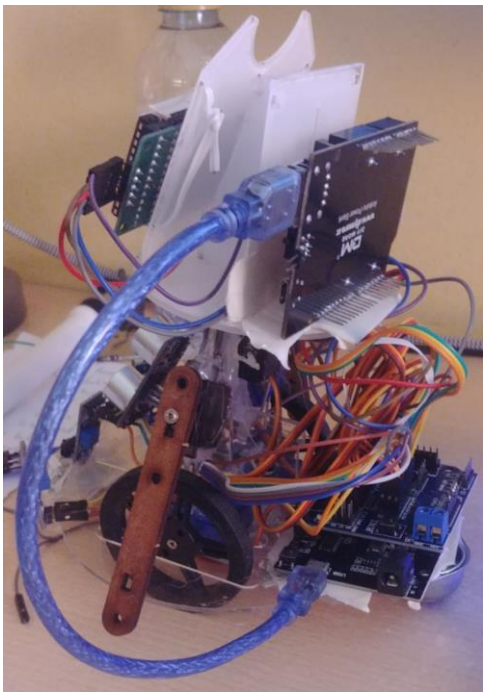


Ilustración 22:Solución alternativa conexionado de batería.

Además, en las especificaciones técnicas se remarca que será necesario tener un botón que permita conectar o desconectar la batería, disponible desde el exterior de la estructura, por lo que con tal fin se ha realizado la siguiente modificación al módulo de baterías.

Ilustración 23: Soldadura de pines machos en swich del módulo 16340.

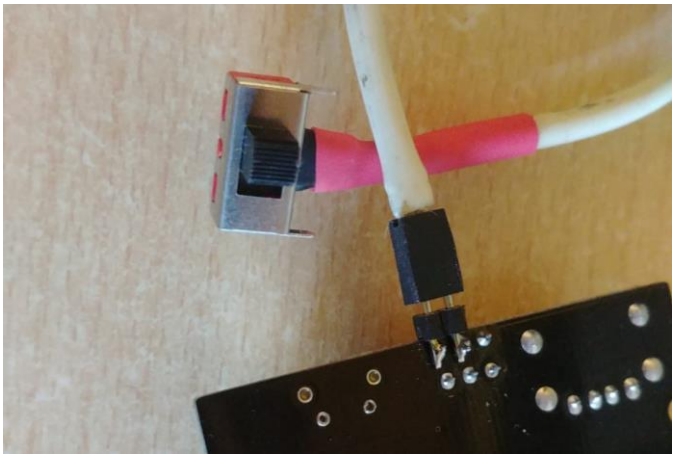


Ilustración 24: Conexionado de cable con extensión del swich del módulo 16340.

Para crear esta extensión del swich que ya integraba el módulo 16340 de por sí se ha ubicado con el tester los pines del circuito que cerraba este swich, posteriormente hemos soldado dos pines macho. A continuación, se ha soldado otro swich en un extremo de un cable y una pareja de pines hembras al otro.

I.6.8. ARDUINO

Se ha adoptado como solución utilizar Arduino IDE para la programación del microcontrolador, mientras que para el desarrollo de la aplicación móvil y su API se ha optado por App Inventor II. En el próximo apartado se mostrará las configuraciones de nuestro entorno de trabajo.

Para que el Arduino pueda saber que placa se está usando, y de esa manera que arquitectura tiene, lo primero que hay que hacer es incluir este enlace https://dl.espressif.com/dl/package_esp32_index.json en el gestor de tarjetas, dentro del apartado “Archivo > Preferencias”. Este enlace es un fichero Json, el cual encapsula la información en forma de objetos, los cuales se componen a su vez de propiedades. Este tipo de archivos es muy utilizado para intercambiar datos entre diferentes programas, debido a que normalmente todos tienen un método para decodificar la información, además tiene como ventaja que es fácilmente interpretable para el programador.

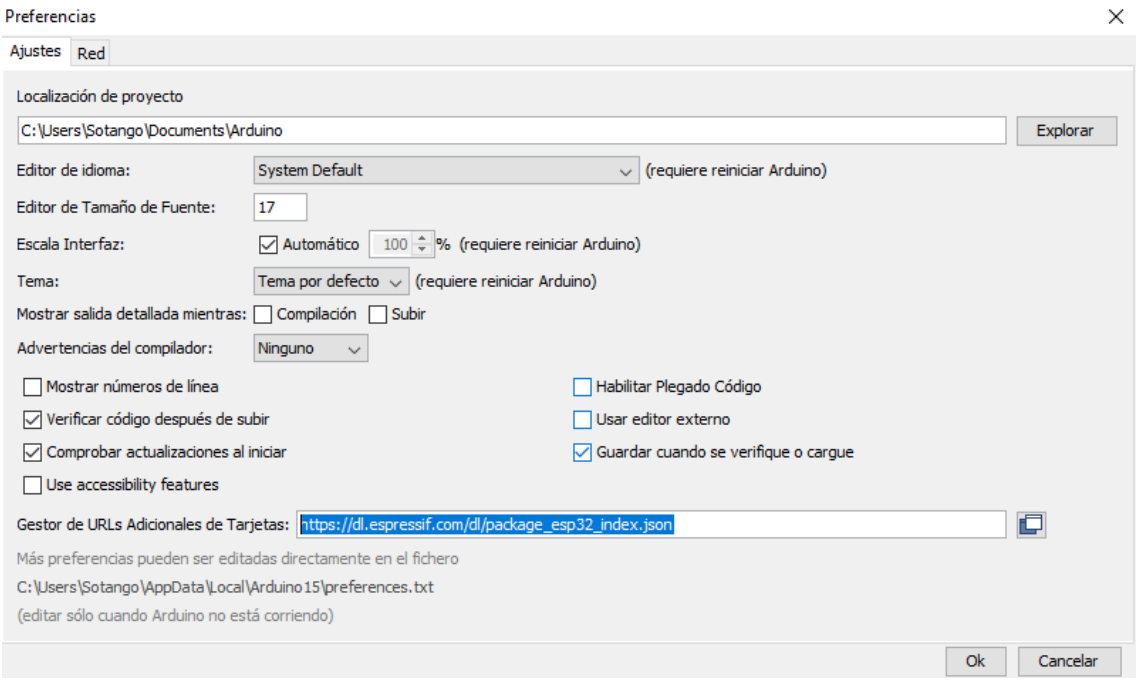


Ilustración 25: Configurando gestor de tarjetas.

Este fichero contiene varias placas que añadirá al gestor de tarjetas de Arduino, lo que nos permitirá descargar el paquete y elegir la opción “Wemos LOLIN32”

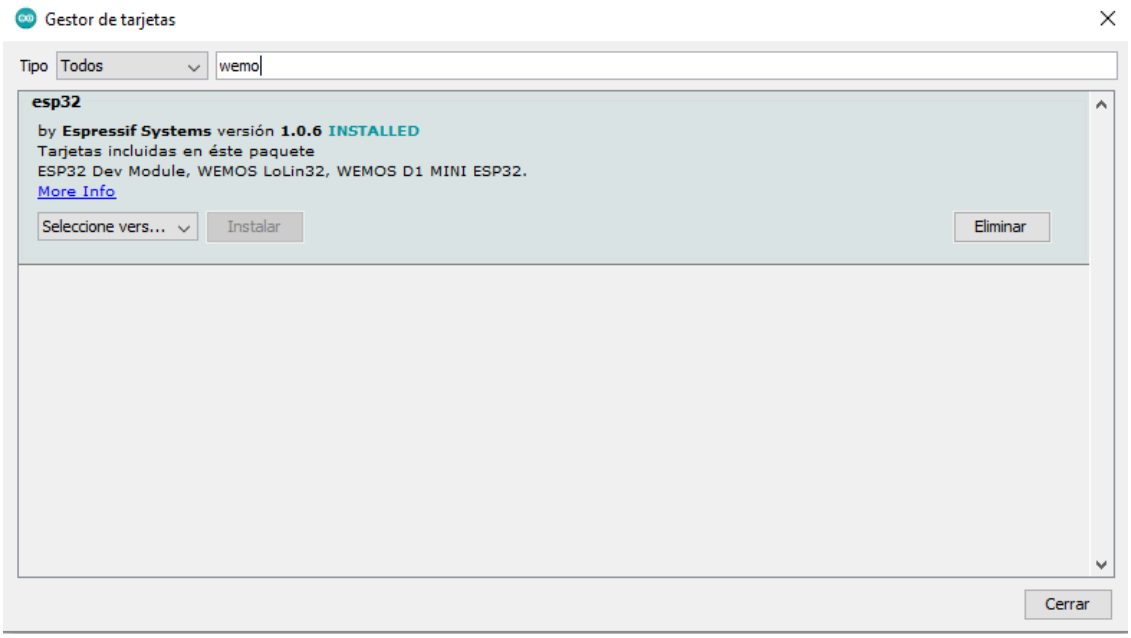


Ilustración 26: Descarga de paquete con las tarjetas.

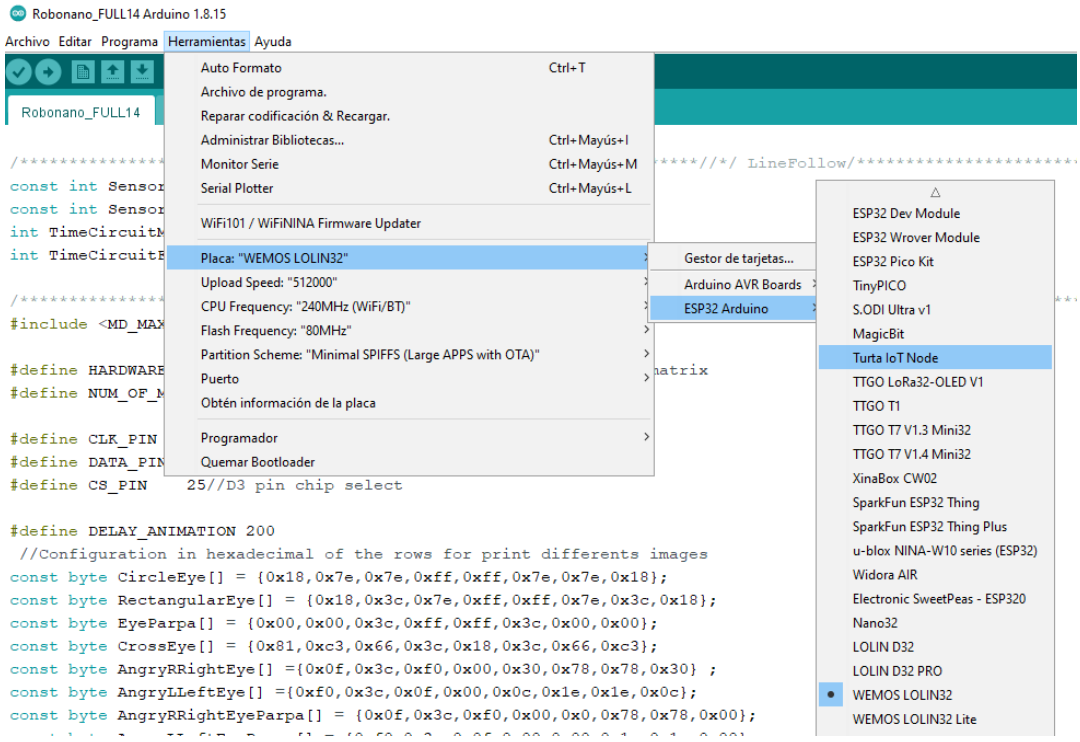


Ilustración 27: Selección de placa en Arduino Ide.

A pesar de no ser nuestra placa, “WeMos D1”, utiliza el mismo esquemático y funciona perfectamente, además es necesario elegir esta alternativa porque nos da la opción de aumentar

la memoria para el código, reduciendo el apartado de memoria SPIFFS, la cual no utilizamos en nuestro proyecto, de otra manera el código excedería la memoria disponible y no sería posible cargar el programa en la placa. Se muestra en la siguiente imagen.

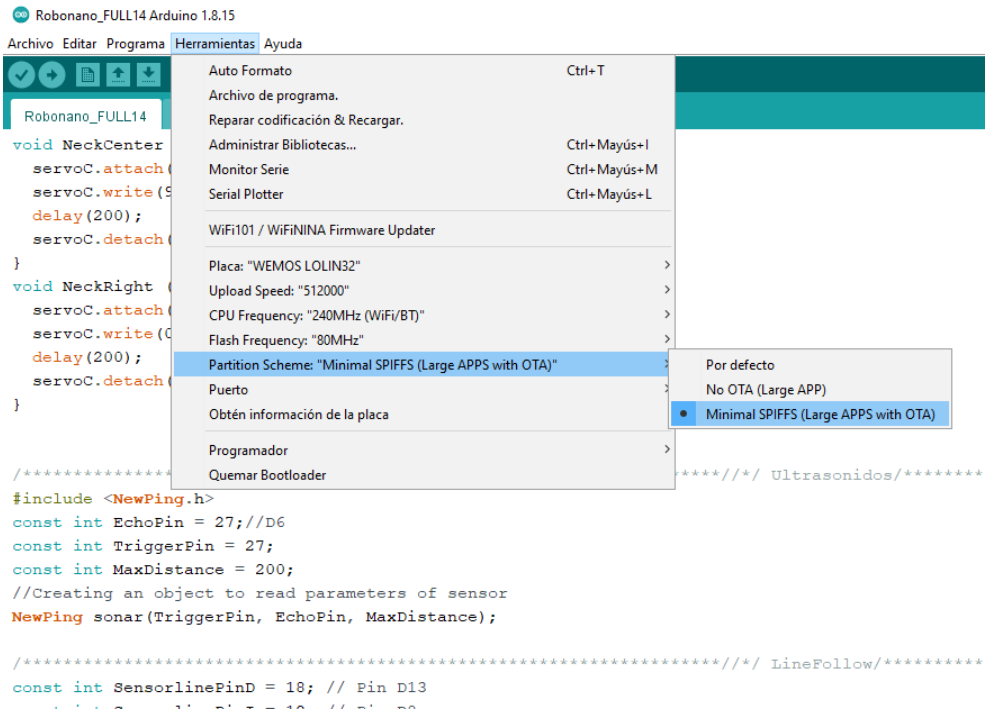


Ilustración 28: Seleccion de partición de memoria.

Tras este punto ya se tiene la configuración básica desarrollada, únicamente se debe conectar la placa al ordenador a través del cable micro-usb, seleccionar el puerto COM al que la hemos conectado y cargar el programa deseado.

I.6.9. LIBRERÍA OTA

La librería OTA[6] permite la carga de datos en el microcontrolador de forma inalámbrica, es decir se podrá prescindir del cable micro usb como canal de comunicaciones entre el ordenador y el micro. Esta funcionalidad es realmente interesante al tratarse de un prototipo con una carcasa que estará cerrada por completo, dejando toda la electrónica inaccesible en su interior. Con la utilización de esta librería será posible reprogramar el robot sin tener que abrirlo, siempre que haya conexión Wifi disponible, utilizando la comunicación por Socket a través de la red y las

[6] Andrássy, J., 2016. *Librería ArduinoOTA*. [online] GitHub. Available at: < https://github.com/jandrassy/ArduinoOTA > [Accessed 2 February 2021].

funciones de la librería OTA, las cuales se encargan de todo el proceso, se exponen a continuación un resumen del proceso a seguir.

Ilustración 29: Instalación de Python 3.9.5

En primer lugar, para el correcto funcionamiento de la librería es necesario instalar en nuestro equipo “Python” e incluir la opción que no viene predefinida de “Add Python to PATH”.

```
void ConnectWiFi_STA(bool useStaticIP = false)
{
  Serial.println("");
  //WiFi.config(local_IP, gateway, subnet);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());
}
```

Ilustración 30: Código función conexión a Wifi.

El primer paso será conectarse a la red para que la librería OTA pueda hacer su trabajo, para ello llamaremos al principio del **setup()** a esta función, la cual sencillamente necesita los parámetros de la “ssid” del Wifi, que es el nombre visible en cualquier dispositivo al escanear las redes disponibles y por otro lado la contraseña de la red. Una vez estemos conectados ya podemos iniciar las funcionalidades de la OTA

La primera vez que carguemos el programa utilizando funciones de esta librería será necesario hacerlo a través del cable, una vez el programa este subido en nuestra placa y reiniciemos Arduino IDE, si se han seguido los pasos correctamente aparecerá un nuevo puerto en “Herramientas >> Puerto” correspondiente a un puerto virtual de red, en este caso ya se puede desconectar el cable del ordenador, alimentar la placa con una batería y cargar el programa de nuevo seleccionando este puerto.

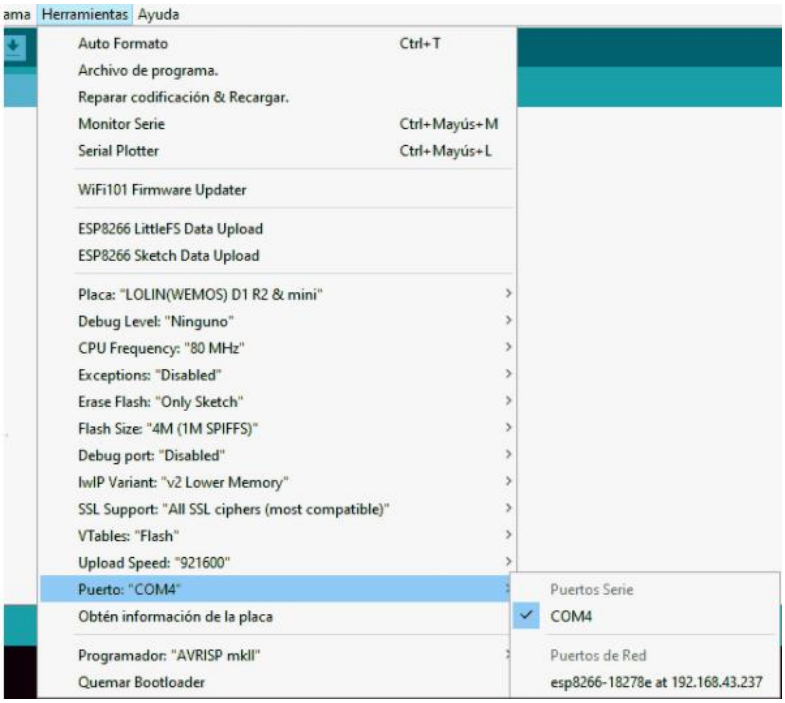


Ilustración 31:Puerto en red correspondiente a OTA.

I.7. POSIBLES MEJORAS

En cuanto a mejorías del hardware principalmente se podría apuntar a la mejora de los sensores, debido a que estos son de muy bajo coste y tienen muchas limitaciones como errores de medición y un campo de detección pequeño.

Otra alternativa sería añadir más sensores de bajo coste, cubriendo todos los ángulos de nuestro robot, lo que nos permitiría tener un control completo de su posición en todos los sentidos, además nos ofrecería la capacidad de implementar algoritmos más elaborados. Sería factible la combinación de diferentes tipos de sensor, incluyendo en nuestro diseño por ejemplo sensores laser, los cuales tienen un alcance bastante mayor a los ultrasonidos, aunque no poseen tanto campo de visión.

Añadir más matrices a nuestro proyecto sería una opción viable, la cual nos permitiría poder imprimir mensajes más largos sin necesidad de deslizamiento y la implementación de animaciones más elaboradas.

En cuanto al firmware se podría haber añadido funcionalidades como servidores web donde almacenar los últimos comandos enviados por nuestra App, un registro de todas las medidas realizadas por nuestro sensor en tiempo real e incluso la posibilidad de establecer control del robot por estas vías como alternativa al bluetooth, haciendo híbrida la forma de control.

Establecer un control PID para los motores de las ruedas con la finalidad de que en el momento del arranque o frenada, independientemente de la velocidad seleccionada para las ruedas, sea progresivo y suave, evitando así desequilibrios del robot o movimientos bruscos que puedan hacer derrapar las ruedas.

En cuanto al software se podría haber mejorado el entorno de trabajo para programar, usando aplicaciones más complejas como Qt o Thunkable-X, las cuales nos permitiría ampliar las posibilidades para navegar entre los menús e implementar nuevas funcionalidades.

I.8. PRUEBAS Y AJUSTES FINALES

En este apartado se va a realizar una descripción detallada de las pruebas realizadas al robot para verificar su correcto funcionamiento, además se va a explicar cómo se han realizado los ajustes finales, dependiendo de las imperfecciones del diseño, montaje o del propio hardware.

Antes de realizar este apartado se debe de haber verificado que la comunicación bluetooth funciona perfectamente, y que todos los comandos enviados por la App son recibidos correctamente por el microcontrolador y asociados a sus funciones correspondientes. Este apartado se limita a los ajustes de parámetros dentro de esas funcionalidades deseadas.

I.8.1. CONTROL REMOTO

I.8.1.1. AJUSTE DE PARÁMETROS

Para ajustar las velocidades de las ruedas en las diferentes trayectorias debemos de saber que velocidad en una rueda equivale a la otra, ya que, aunque por firmware se ordene girar a la misma velocidad es probable que esto no ocurra. Por otro lado, es necesario considerar que, por cuestiones de diseño de la estructura, el motor derecho tendrá el sentido invertido respecto al izquierdo, ver Tabla Nº4 para ver cómo hacer girar las ruedas a cierta velocidad.

Para realizar los ajustes relacionados con las velocidades de ruedas tenemos dos formas de ajustarlo:

Por un lado, ajustando las velocidades de las ruedas que se guardan al seleccionar una nueva velocidad de movimiento desde la App, ver Ilustración Nº29, las cuales nos tenemos que asegurar de que siempre realicen una trayectoria recta, teóricamente estas velocidades serán iguales, pero de sentido inverso, siendo (0>V>110) sentido inverso de giro y (110<V<180) sentido directo, aunque esto difícilmente ocurrirá y será necesario calibrar los valores para que funcione correctamente.

El siguiente paso, una vez nuestro robot sea capaz de avanzar en línea recta será ajustar las constantes de giro del robot, estas constantes están presentes en cada tipo de giro (arriba izquierda,

derecha y abajo izquierda, derecha) y son las encargadas de hacer frenar en mayor o menor medida a la rueda en la dirección en la que se vaya a ejecutar el giro, ver Ilustración Nº30. Hay que tener en cuenta que para la rueda derecha una constante mayor de 1 aminorará la velocidad, mientras que para la rueda izquierda la aumentaría; por tanto, dependiendo del sentido en el que nos queramos mover y de la rueda que vayamos a mover la constante deberá de ser mayor de 1 o menor.

```
//Control of Velocity of wheels in
if (command[0] == 'W'){
    if (command[1] == '0'){
      V=98;              //********
      Viz=123;
    }
    else if (command[1] == '1'){
      V=93;              //********
      Viz=129;
    }
    else if (command[1] == '2'){
      V=82;
      Viz=142;
    }
    else if (command[1] == '3'){
      V=73;
      Viz=155;
    }
    else if (command[1] == '4'){
      V=60;
      Viz=160;
    }
    else if (command[1] == '5'){
      V=53;
      Viz=170;
    }
```

Ilustración 32: Ajuste de diferentes velocidades de ruedas al recibir comando de variar velocidad

```
//Motor derecho gira alreves
//On Turns only the whhel that is going to be the fina
void UPRight1 (void){
    int Vfinal;
    //K for slow down Right wheel
    //Keep between 1.1 and 1.9 on this function
    float kR=1.8;
    if(kR*V>102) Vfinal=103;
    else Vfinal=kR*V;
    servo360R.attach(servo360RPin,minUs360,maxUs360);
    servo360L.attach(servo360LPin,minUs360,maxUs360);
    servo360R.write(Vfinal);
    servo360L.write(Viz);
}
```

Ilustración 33: Ejemplo de función de giro donde se aplica constante de aminoramiento.

La otra prueba necesaria será visualizar los limites reales de las articulaciones, es decir una vez con los brazos ensamblados en los motores observar cual es el ángulo máximo tanto hacia arriba como hacia abajo, en este caso igualmente debemos de considerar que los sentidos de giro serán inversos un lado de otro.

I.8.1.2. PRUEBAS DE VERIFICACIÓN

TRAYECTORIA RECTA SENTIDO DIRECTO

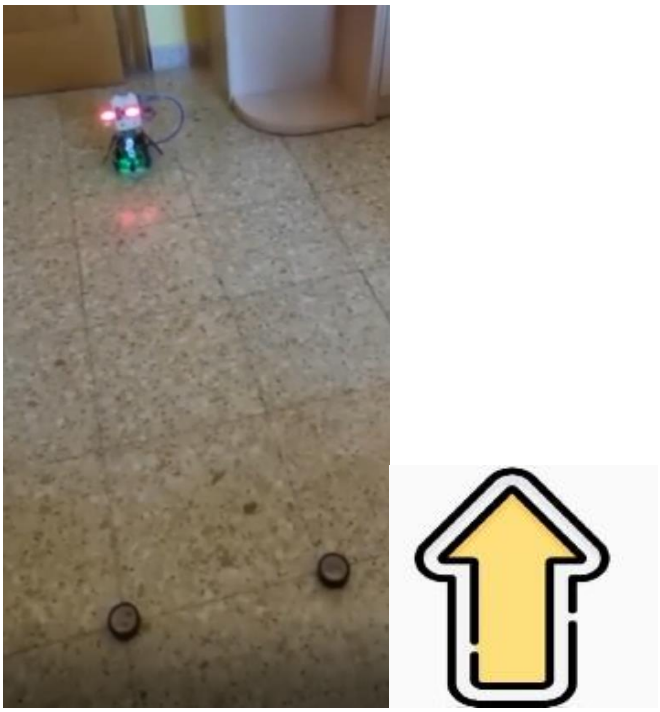


Ilustración 34:Prueba verificación trayectoria recta I.

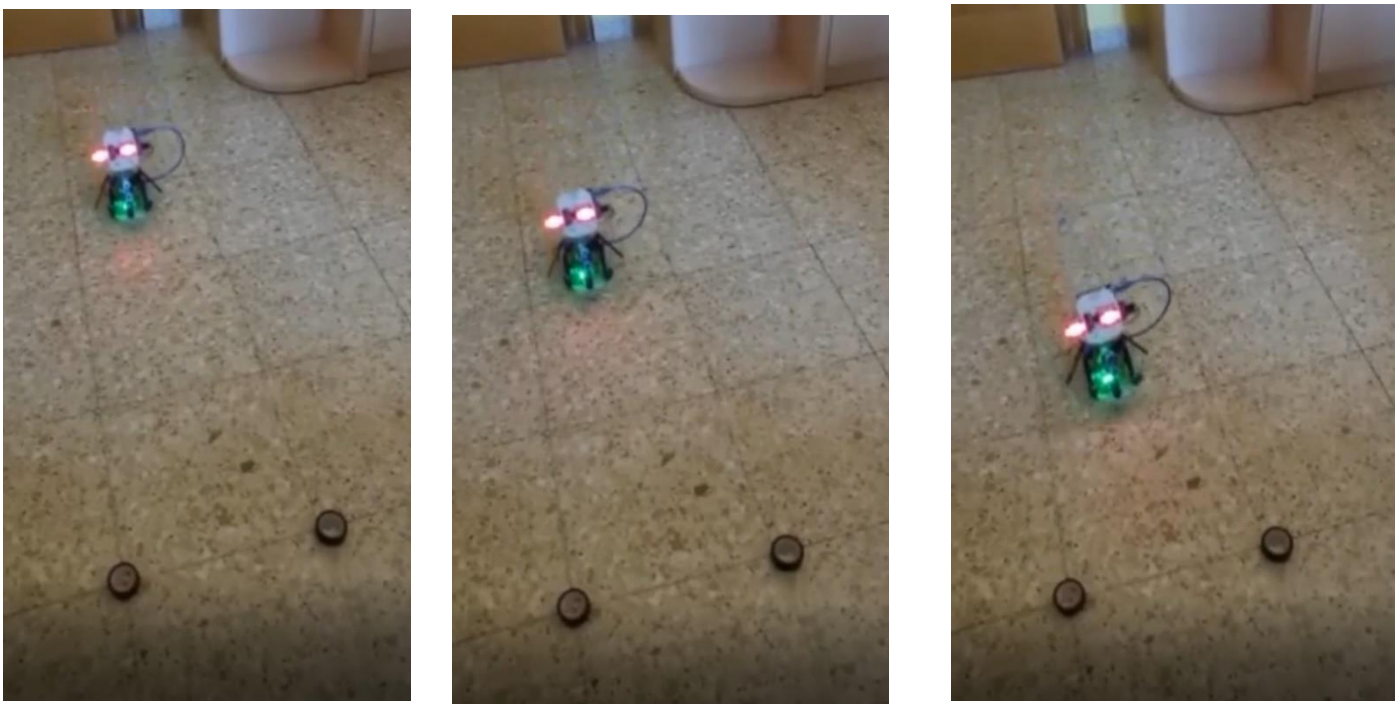


Ilustración 35: Prueba verificación trayectoria recta II.



Ilustración 36: Prueba verificación trayectoria recta III.

TRAYECTORIA RECTA SENTIDO INVERSO

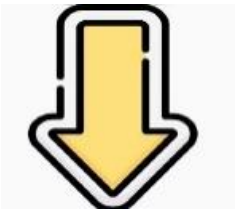



Ilustración 37:Prueba verificación trayectoria recta atrás I.

Ilustración 38: Prueba verificación trayectoria recta atrás II.

Ilustración 39: Prueba verificación trayectoria recta atrás III.

GIRO VELOCIDAD MÁXIMA

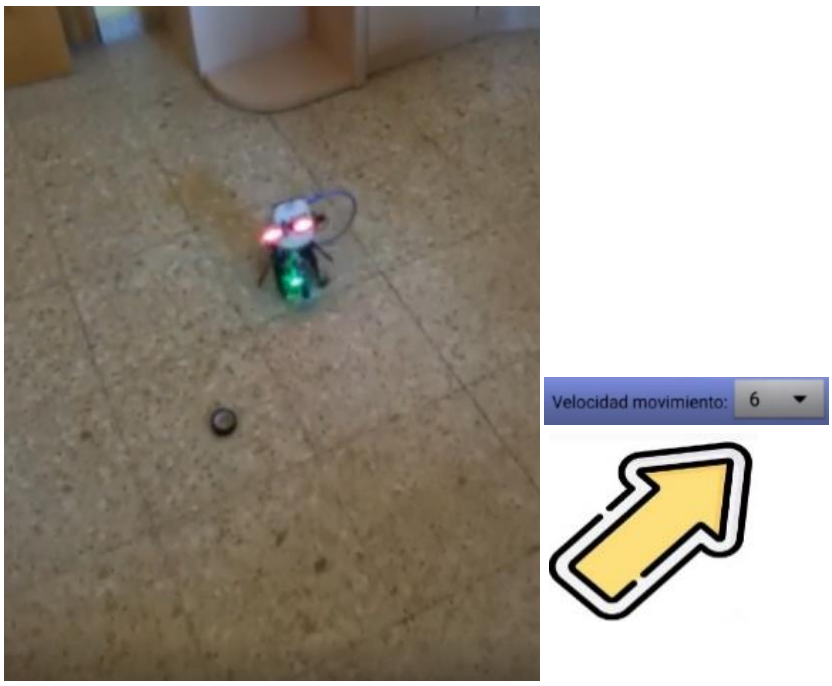


Ilustración 40: Prueba verificación trayectoria curva velocidad máxima I.

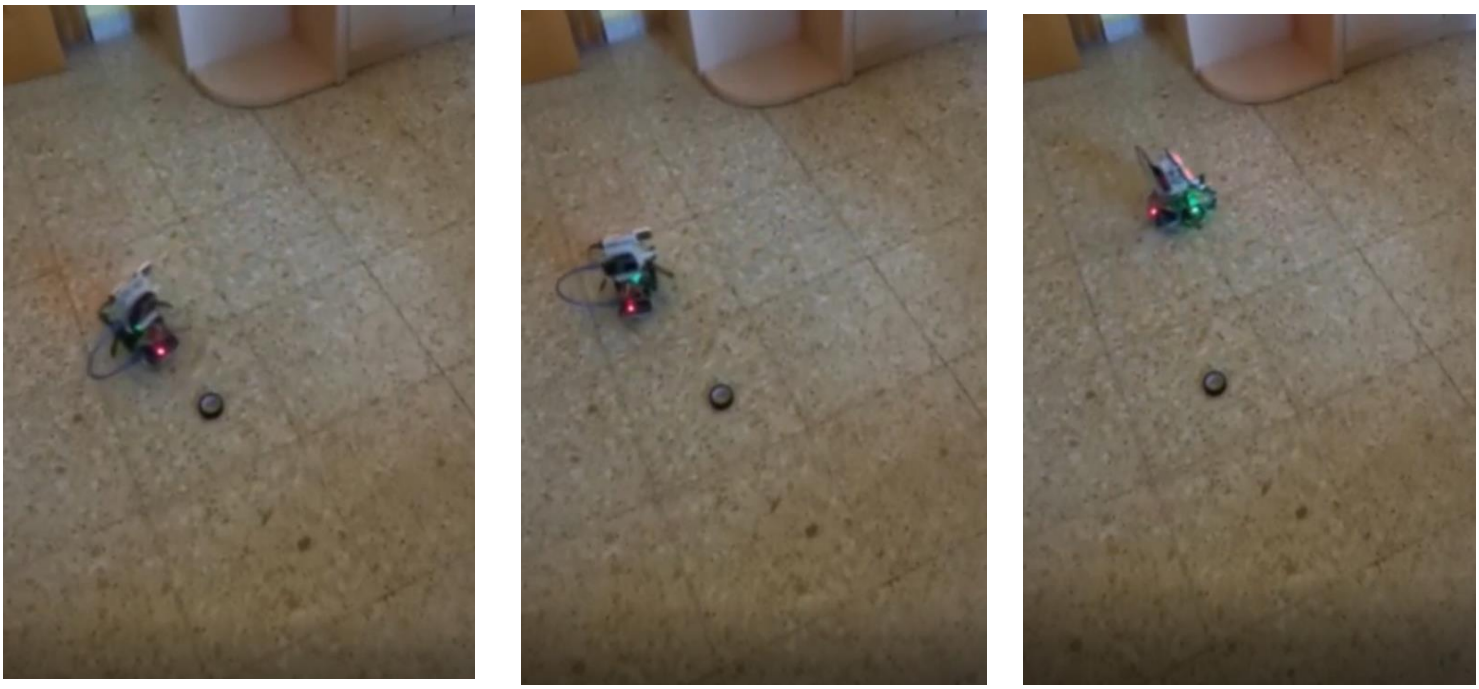


Ilustración 41: Prueba verificación trayectoria curva velocidad máxima II.

GIRO VELOCIDAD MÍNIMA

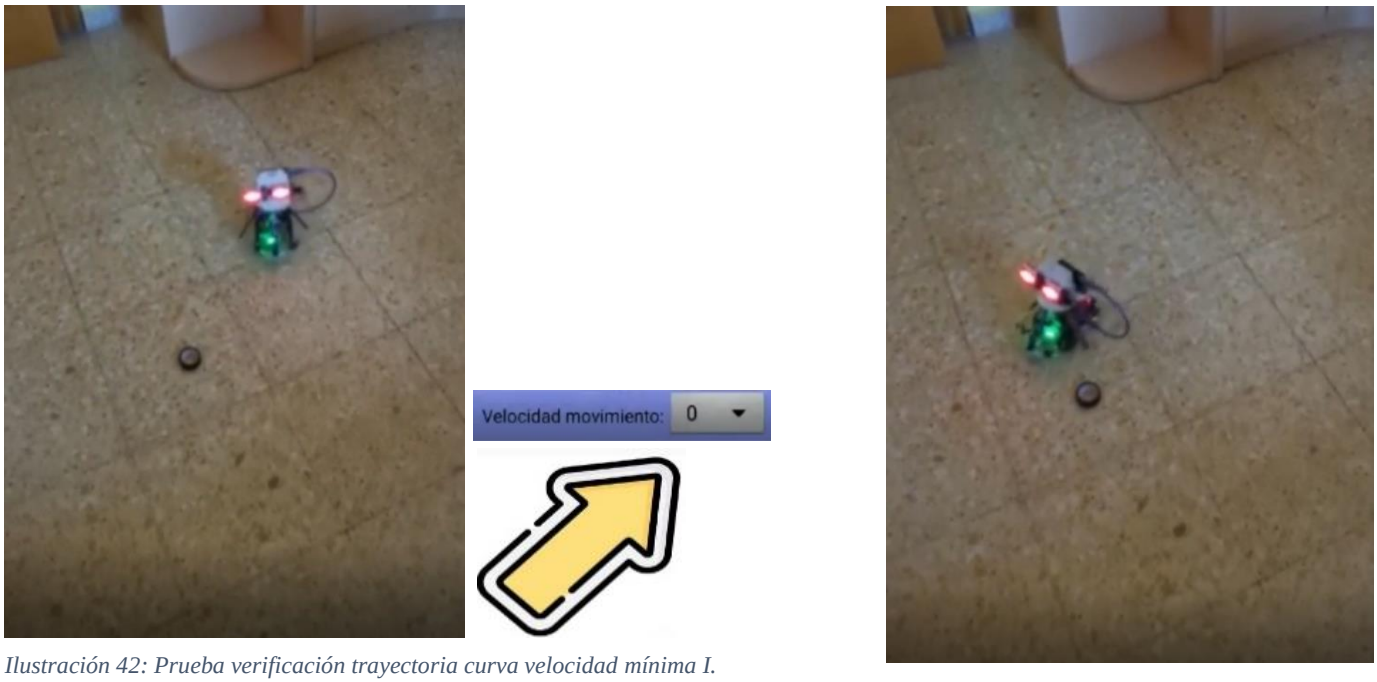


Ilustración 42: Prueba verificación trayectoria curva velocidad mínima I.

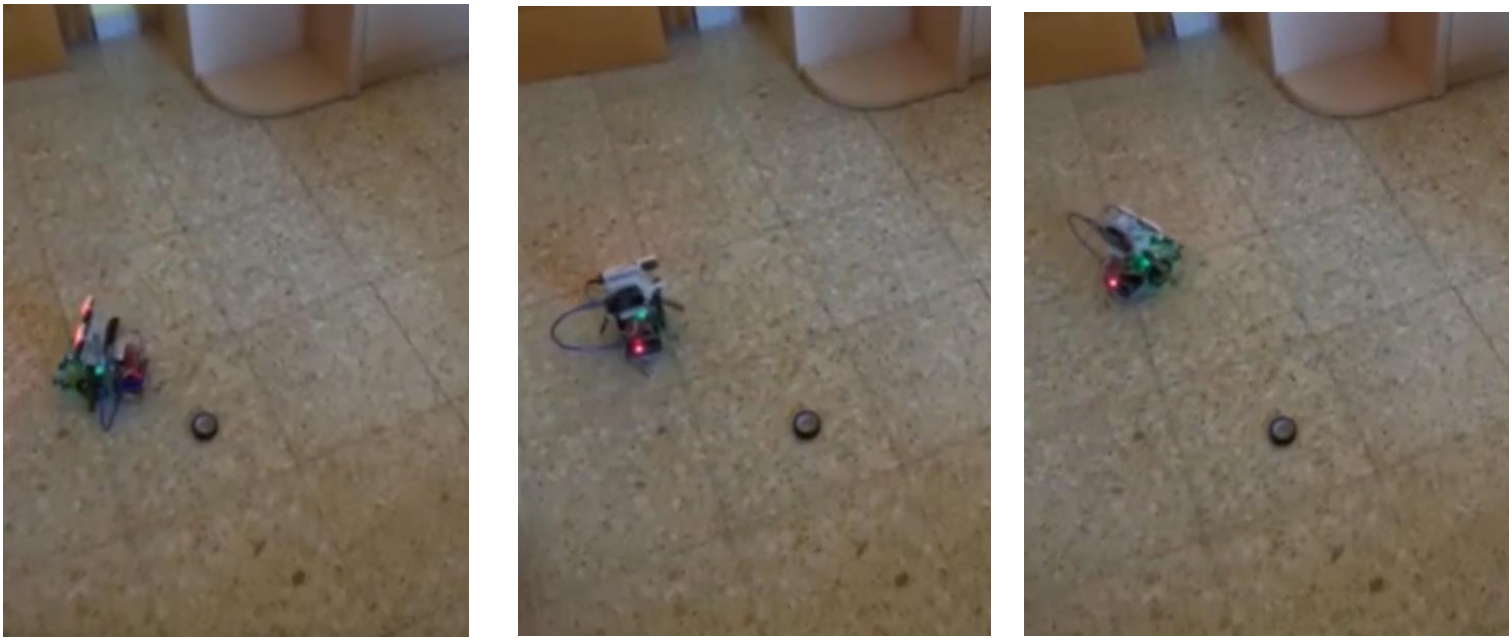


Ilustración 43: Prueba verificación trayectoria curva velocidad mínima II.

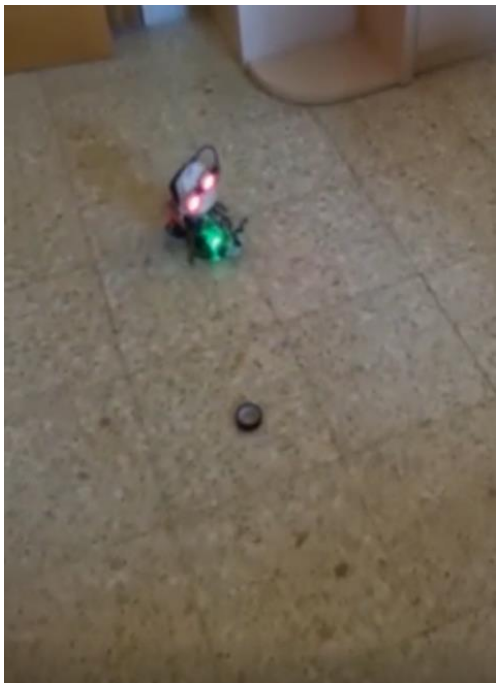
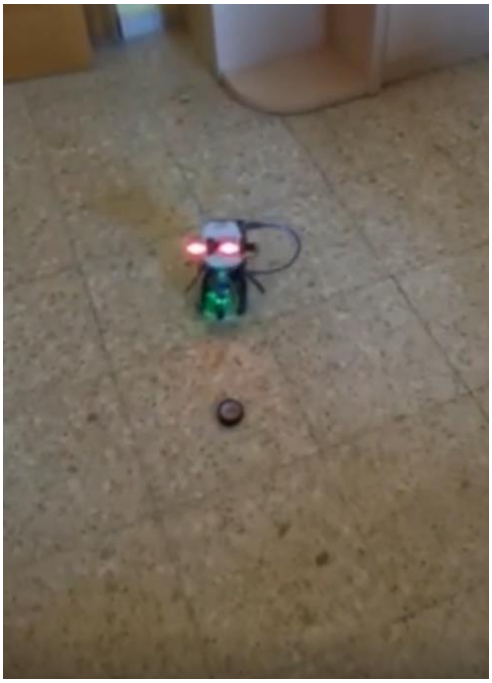




Ilustración 44: Prueba verificación trayectoria curva velocidad mínima III.

LIMITES DE ARTICULACIONES

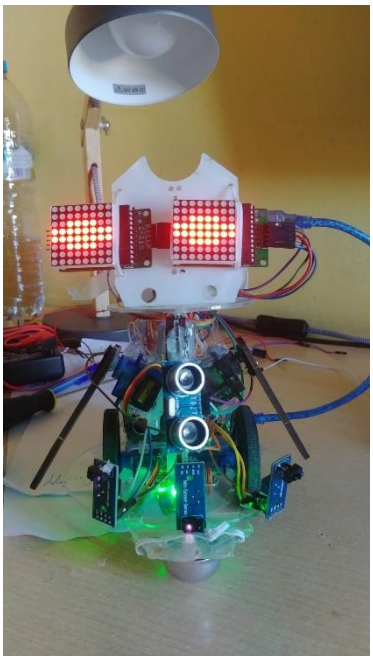
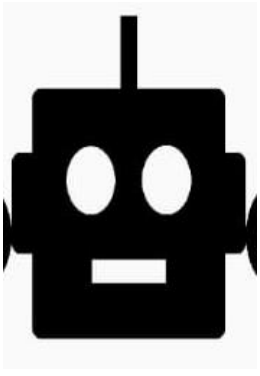



Ilustración 45: Cuello del robot centrado.

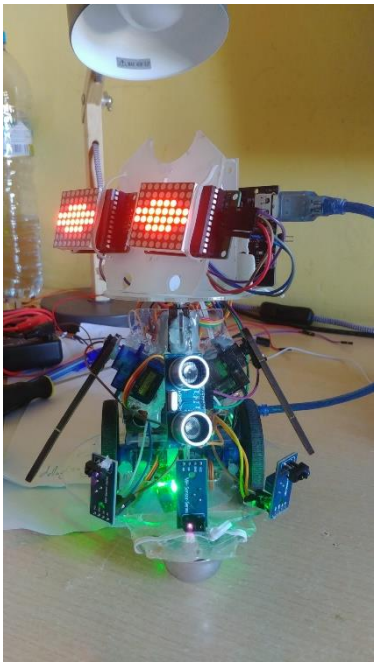
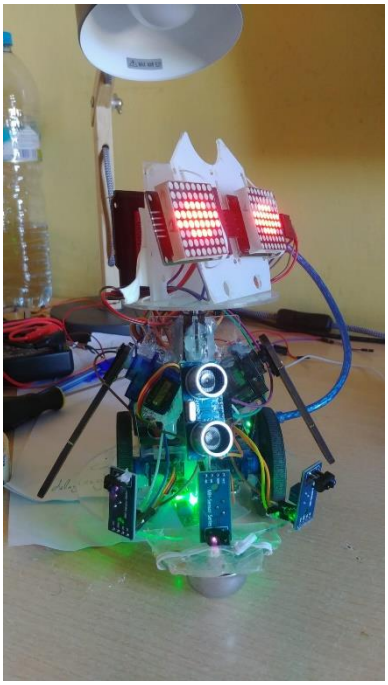


Ilustración 46: Cuello del robot derecha.



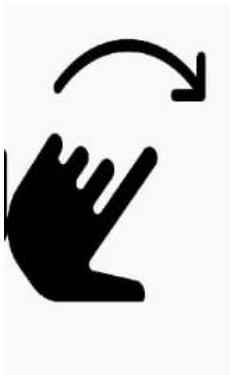
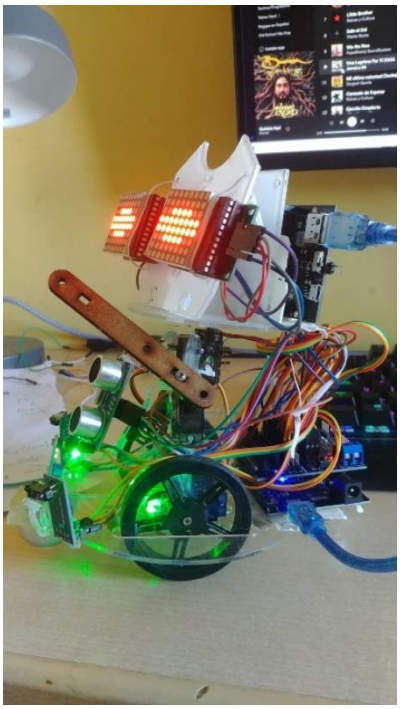


Ilustración 47: Cuello del robot izquierda.



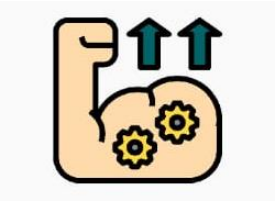
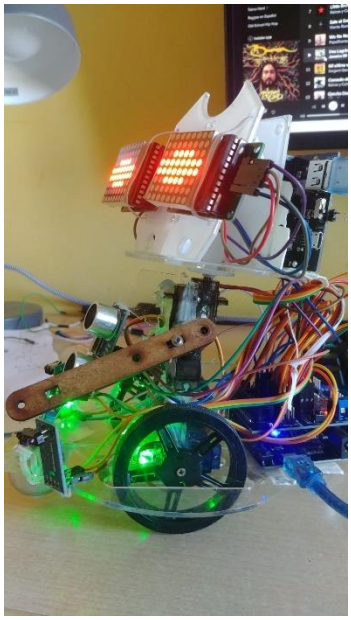


Ilustración 48:Brazo del robot arriba.



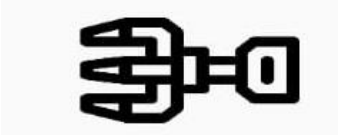


Ilustración 49:Brazo del robot al medio.

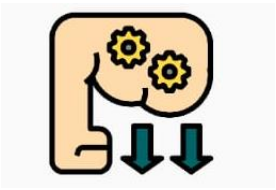


Ilustración 50: Brazo robot abajo.

I.8.2. SIGUE-LINEAS

I.8.2.1. AJUSTE DE PARÁMETROS

En primer lugar, se diseñará el circuito a realizar, para ello trazaremos sobre una superficie blanca y plana, como una cartulina o una lona estirada, una línea que corresponderá con la trayectoria deseada para el robot, esta trayectoria debe de comenzar y acabar en el mismo punto, es decir creará una superficie cerrada en el papel. Para dibujar la línea utilizaremos cinta de color completamente negro de grosor de al menos 3cm, de lo contrario no sería detectada por el robot, además en el trazado de la línea no deberá de haber giros de más de 90º.

Ilustración 51: Circuito sigue-líneas.

Dependiendo del lado en el que se coloque el robot de la línea se deberá de implementar el algoritmo que se detalla en el Plano Nº 7 en una dirección u otra, es decir, si la línea queda a la derecha del robot, su estado natural será avanzar en línea recta con cierta desviación hacia la derecha, y en el momento que se detecte la línea girar sobre sí mismo hacia el lado contrario, hasta que la línea deje de ser detectada, por otra parte se ha implementado una función que en caso de no haber detectado la línea en cierto tiempo el robot se reorientará en su dirección.

Tendremos tres parámetros a ajustar: el primero es que desviación hay al avanzar en su estado natural en línea recta, partiendo de la mínima velocidad disponible en línea recta restaremos poco a poco velocidad a la rueda hacía queremos desviarnos. El segundo es la brusquedad deseada en el giro sobre sí mismo, se recomienda partir desde las velocidades más pequeñas. El último será el número de desbordes de un timer de 5ms, según el tiempo deseado antes de reorientar el robot en dirección a la línea si esta no ha sido detectada.

I.8.2.2. PRUEBAS DE VERIFICACIÓN

CIRCUITO

Se incluye únicamente una tira de fotogramas de una parte del circuito.

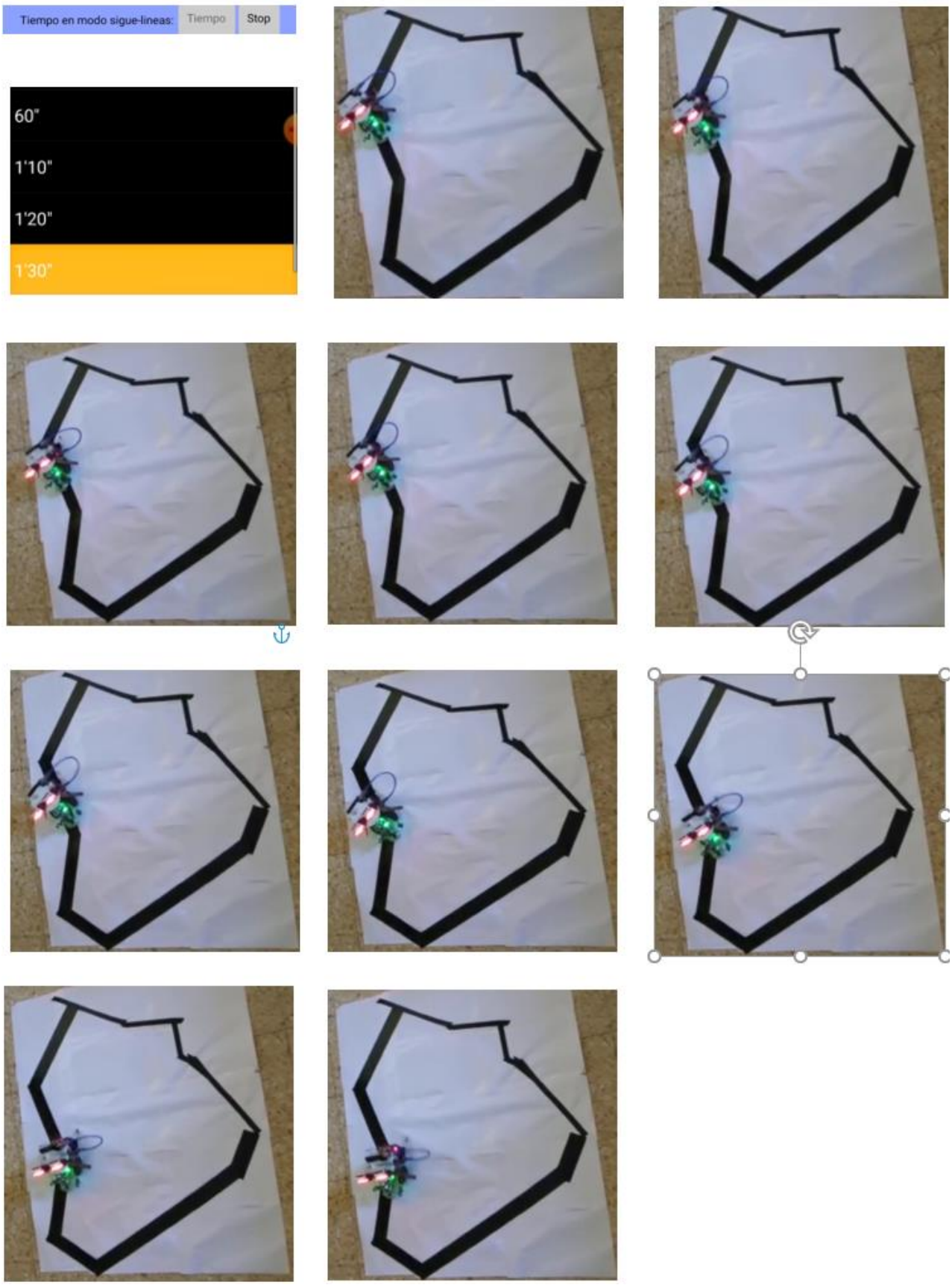


Ilustración 52:Tira de fotogramas de robot siguiendo el circuito I.

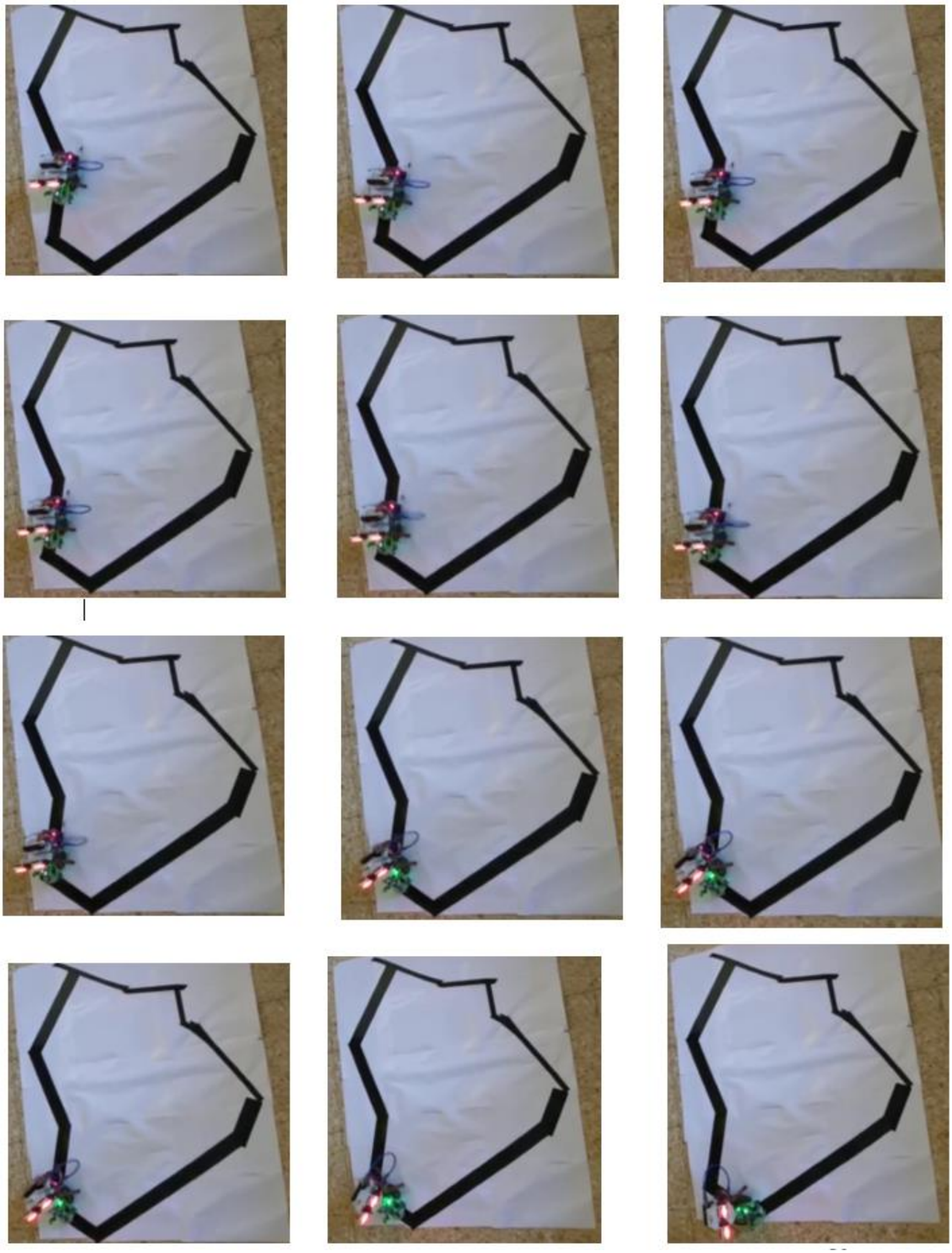


Ilustración 53:Tira de fotogramas de robot siguiendo el circuito II

I.8.3 MODO AUTÓNOMO

I.8.3.1. AJUSTE DE PARÁMETROS

Los parámetros a modificar en el modo evita-obstáculos o autónomo son muy similares a los del modo sigue-líneas, ya que el algoritmo se compone de dos estados, el estado natural en el que el robot avanza en línea recta, donde las velocidades de las ruedas serán el único parámetro modificable, y su estado cuando detecta el obstáculo, en el que el robot retrocederá un poco y girará sobre sí mismo durante el tiempo necesario hasta que no se detecte un obstáculo en su rumbo, dónde básicamente se podrá modificar la velocidad de rotación sobre sí mismo.

I.8.3.2. PRUEBAS DE VERIFICACIÓN

TIEMPO SIN BLOQUEO

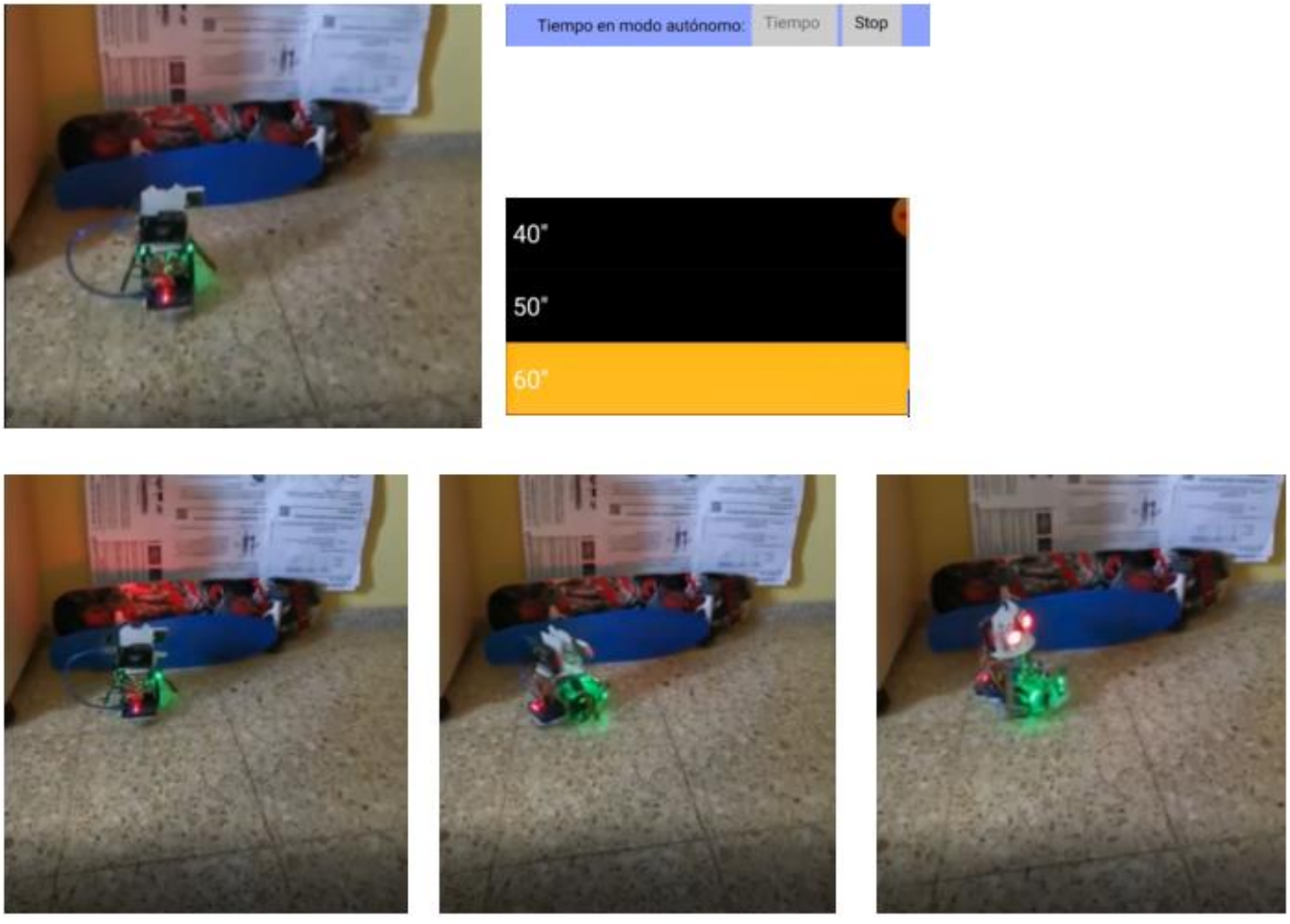


Ilustración 54: Tira de fotogramas modo autónomo I.

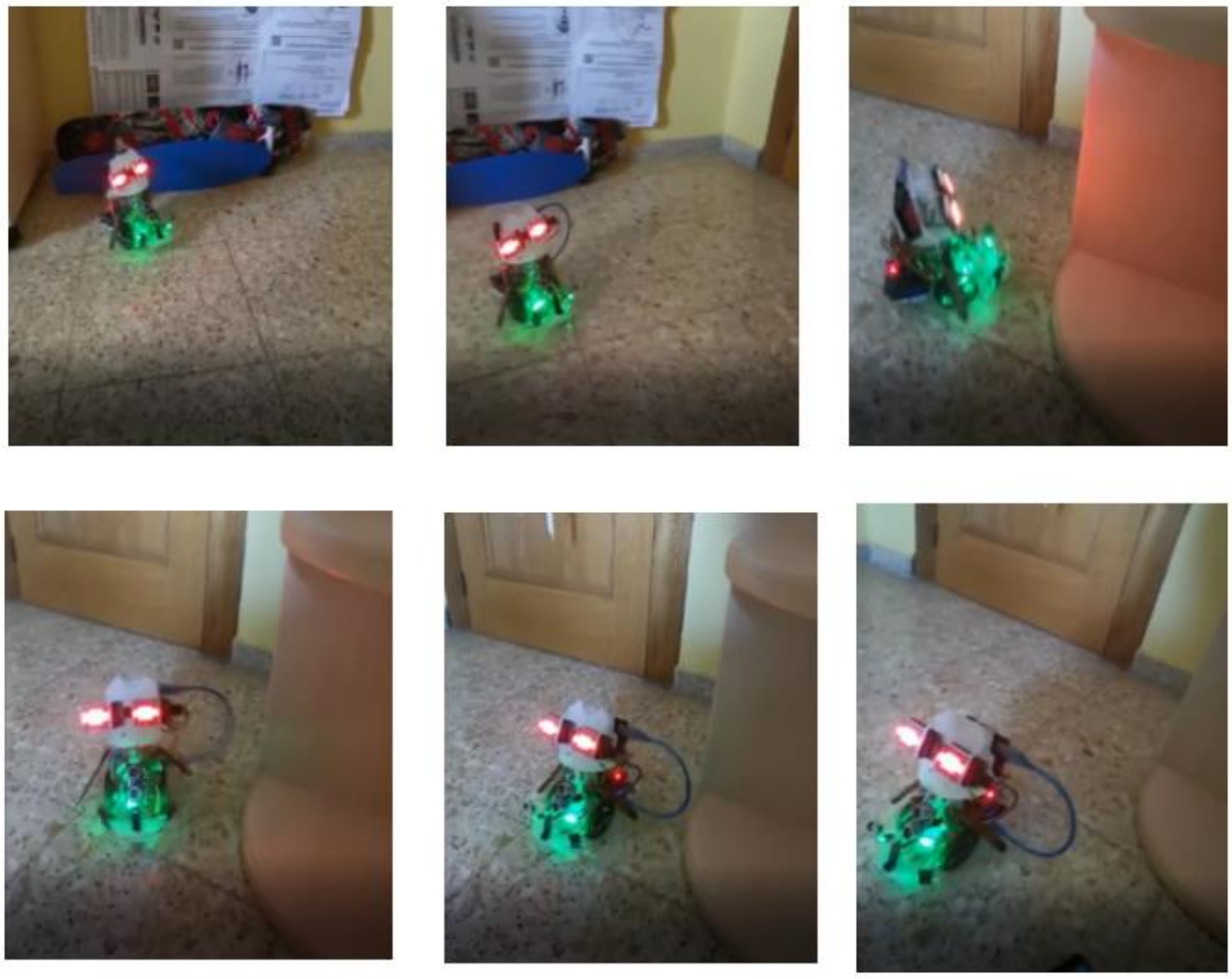


Ilustración 55:Tira de fotogramas modo autónomo II.

I.9. ANEXOS A LA MEMORIA

I.9.1. ANEXO A: DATASHEETS

1. ESP32
   Roboticafacil.es. 2017. *Datasheet ESP32*. [online] Available at: < https://roboticafacil.es/datasheets/esp-32.pdf > [Accessed 5 March 2021].

2. Arduino Sensor Shield

Curtocircuito.com.br. 2021. *Datasheet Arduino Sensor Shield V.5*. [online] Available at: < https://curtocircuito.com.br/datasheet/arduino_sensor_shield.pdf > [Accessed 10 February 2021].

3. Servomotor FS90R

   Roboticafacil.es. 2010. *Datasheet servomotor FS90R*. [online] Available at: < https://roboticafacil.es/datasheets/FS90R.pdf > [Accessed 5 January 2021].

4. Servomotor SG90

   Ee.ic.ac.uk. 2011. *Datasheet servomotor SG90*. [online] Available at: < http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf > [Accessed 5 January 2021].

5. Sensor infrarrojo TCRT5000

   Vishay.com. 2009. *Datasheet sensor infrarrojo TCRT5000*. [online] Available at: < https://www.vishay.com/docs/83760/tcrt5000.pdf > [Accessed 5 February 2021].

6. Sensor ultrasonidos HC-SR04

   Leantec.es. 2010. *Datasheet sensor ultrasonidos HC-SR04*. [online] Available at: < https://leantec.es/wp-content/uploads/2019/06/Leantec.ES-HC-SR04.pdf > [Accessed 5 April 2021].

7. Display led

   Ledtoplite.com. 2014. *Datasheet display led 8x8*. [online] Available at: < http://www.ledtoplite.com/uploadfile/2017/1088/TOP-CC-1088AS.pdf > [Accessed 17 March 2021].

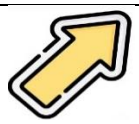
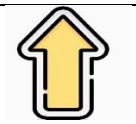
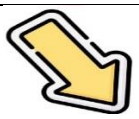
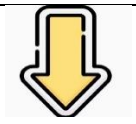
   Datasheet.octopart.com. 2013. *Datasheet MAX7219*. [online] Available at: < https://datasheet.octopart.com/MAX7219CWG-Maxim-datasheet-98628.pdf > [Accessed 28 March 2021].

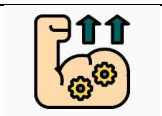
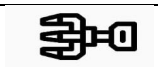
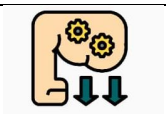
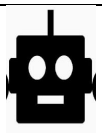
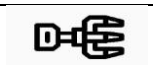
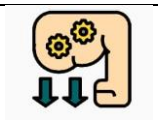
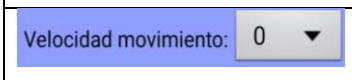
8. Módulo batería

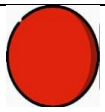
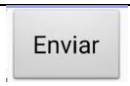
   DIYMORE. 2016. *16340 Battery Shield ESP8266 ESP32 Power Bank Supply Module For Arduino UNO R3*. [online] Available at: < https://www.diymore.cc/products/esp8266-esp32-dual-16340-lithium-battery-module-usb-mobile-power-bank-battery-holder-charger-board-module-for-arduino-uno-r3 > [Accessed 15 June 2021].

I.9.2. ANEXO B: LISTA DE COMANDOS POSIBLES

Tabla 7: Lista de comandos ligados a cada funcionalidad.

Modo de funcionamiento	**Botón en App**	**Comando enviado**	**Función invocada**
Sigue-lineas		o	LineFollow()
			Sale del bucle de LineFollow()
Control remoto		1	UPRight1()
		2	UP2()
		3	UPLeft3()
		4	TurnRight4()
		5	Stop5()
		6	TurnLeft6()
		7	DownRight7()
		8	Down8()
		9	DownLeft9()

		f	ArmLeftUP()
		g	ArmLeftCenter()
		h	ArmLeftDown()
		i	NeckLeft()
		j	NeckCenter()
		k	NeckRight()
		l	ArmRightUP()
		m	ArmRightCenter()
		n	ArmRightDown()
		w+(0≤número<9)	Cambia valor de velocidad de ruedas
Autónomo		x	AutomateMode()
			Sale del bucle de AutomateMode()

Control de leds		a	drawRows() Ojo enfadado
		b	drawRows() Ojo circular
		c	drawRows() Ojo cuadrado
		d	drawRows() Ojo en X
		e	drawRows() Ojo reptil
		r	drawRows() Ojos bizcos
		s	drawRows() Ojos locos
		T	drawRows() Ojos corazón
		v+(0<número≤3)	Cambia valor velocidad deslizamiento
		t	Guarda el resto de la cadena como texto para imprimir

I.9.3. ANEXO C: MANUAL USUARIO APLICACIÓN

En este anexo se va a mostrar la App para móvil desarrollada a partir de “App inventor 2” y se va a hacer una pequeña guía de funcionamiento para el usuario.

I.9.3.1. MENÚ INICIO

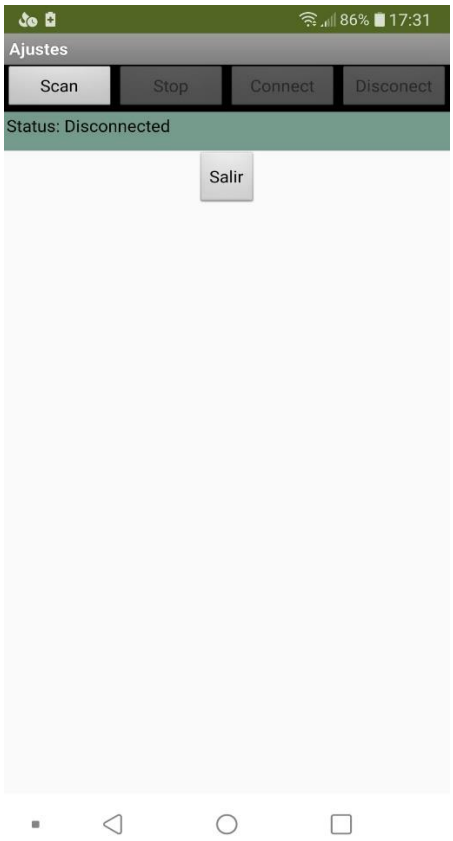


Ilustración 56: Punto de partida en App.

El primer paso para comenzar a usar la app es hacer un escaneo de los dispositivos Bluetooth que haya cercanos, encontrar el dispositivo deseado e intentar conectarse.

En la etiqueta de “Status” nos mostrará nuestro estado de conexión, cambiando dinámicamente según se encuentre el dispositivo.

Una vez pulsado el botón “Scan” y tras haber visualizado el dispositivo deseado en la lista, se deberá en primer lugar parar este modo con el botón “Stop” para posteriormente seleccionar el dispositivo y pulsar a “Connect”.

En caso de producirse algún error al intentar conectarse la app nos mostrará una ventana de aviso como que no se ha podido establecer la conexión, en tal caso se deberá reintentar la operación.

En caso de establecer la conexión con éxito nos aparecerá un nuevo botón para comenzar con el control del robot.

Ilustración 57:Listado de dispositivos disponibles en App.

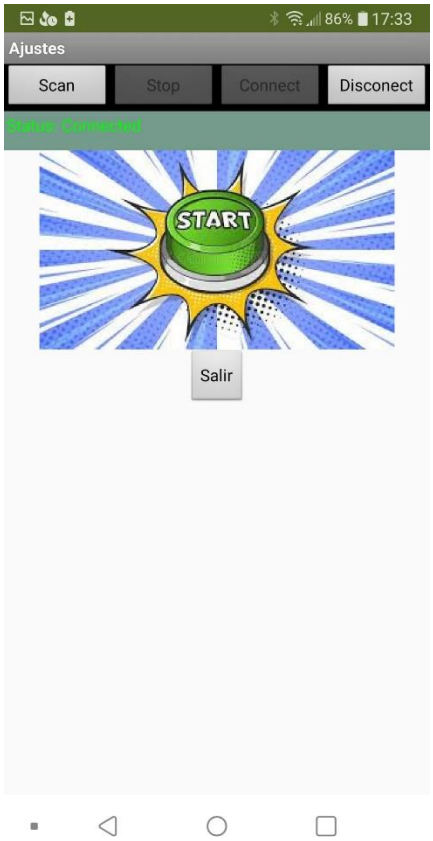


Ilustración 58: Ha fallado la conexión con dispositivo en App.

Ilustración 59: Dispositivo conectado con éxito en App.

I.9.3.2. MENÚ CONTROL

Una vez se ha realizado la conexión podremos elegir el modo de funcionamiento del robot a través de un selector.

Cada vez que se cambie de modo se envía un comando que para todos los motores y posteriormente enviará el comando correspondiente al modo.

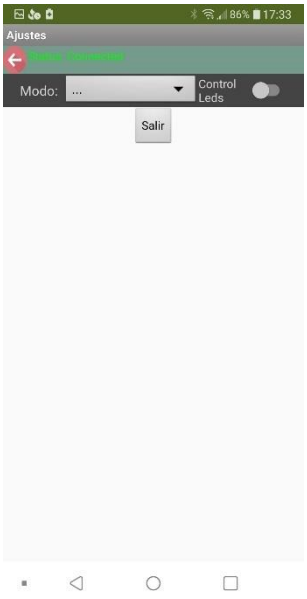


Ilustración 61:Modo control del robot en App.

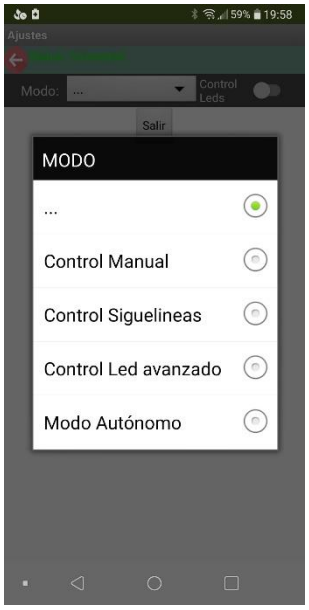


Ilustración 60: Selector de modo abierto en App.

A través de un swich podemos hacer visible el modo para controlar las expresiones del robot, esta botonera será activable en todos los modos de funcionamiento.

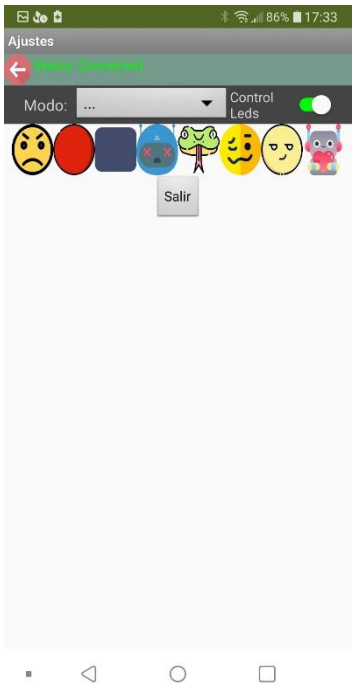


Ilustración 62:Swich para control de leds activado en App.

I.9.3.3. MODO DE CONTROL MANUAL

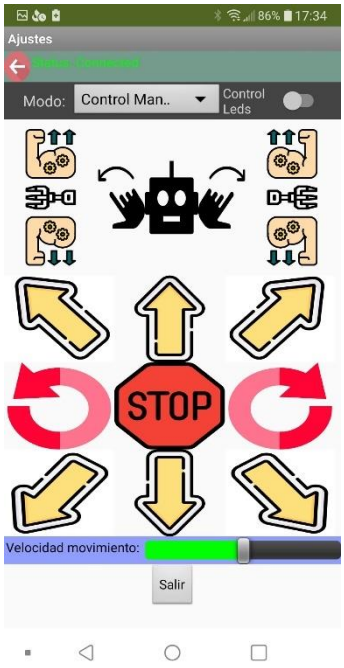


Ilustración 63:Modo de control manual en App.

En el modo de control manual se disponen de 18 botones, 9 correspondientes a mover al robot en diferentes direcciones, y otras 9 con el objetivo de mover las articulaciones del robot entre el punto máximo, mínimo y central de su rango.

Además, se dispone de una barra que modifica la velocidad de las ruedas entre 8.

I.9.3.4. MODO CONTROL DE LEDS AVANZADO

Dentro del modo de control de leds avanzado disponemos de un cuadro para escribir texto, el cual podremos enviar para que se muestre a lo largo de las matrices leds, deslizando de lado a lado y pasando de una a otra. El máximo de caracteres enviables son 22.

Además, se dispone de otra barra que varía la velocidad de desplazamiento del texto imprimido por el robot entre bajo, medio y alto.

El texto será limpiado del cuadro una vez haya acabado de imprimirse en la matriz, además durante este tiempo se impide el envio de otros comandos debido a que nuestro robot estaría en bucle imprimiendo el texto y no escuchará inmediatamente nuevos comandos, sino que se le quedarían acumulados en el puerto serie, si mandamos muchos podríamos saturar el canal y perder la conexión con éste. Ver Plano Nº4 donde se explica el diagrama de flujo de esta estructura.

Ilustración 64:Modo de control led avanzado en App.

I.9.3.5. MODO AUTÓNOMO Y SIGUE-LINEAS

La estructura de ambos modos es la siguiente, al entrar en el modo correspondiente se tendrá acceso a una lista de tiempos posibles para entrar en el modo.

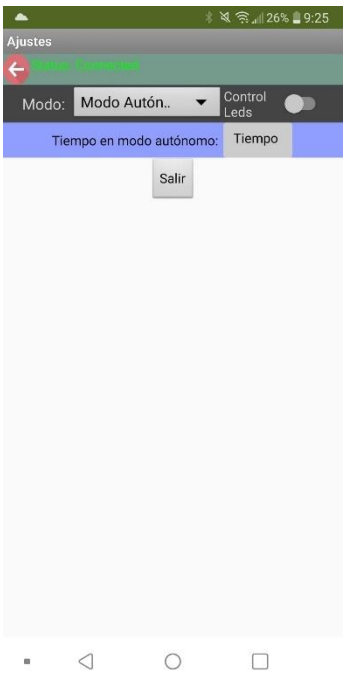


Ilustración 66: Modo de control autónomo en App.

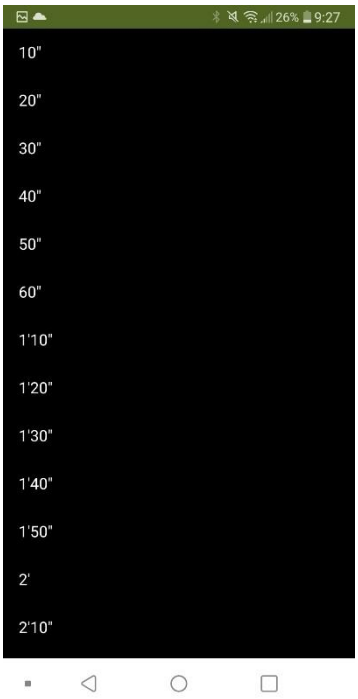


Ilustración 65: Lista de selección de tiempo en modo de control de App.

Una vez seleccionado el tiempo para que permanezca en este modo será imposible mandar cualquier otro comando, debido a que nuestro robot estará en bucle realizando el control correspondiente, y no escuchará inmediatamente nuevos comandos, sino que se le quedarían acumulados en el puerto serie, si mandamos muchos podríamos saturar el canal y perder la conexión con éste.

Por otro lado, será visible un botón de stop, el cual tiene un delay de unos 5 segundos, para darle tiempo al robot a salir del bucle, tras pasado este tiempo ya podremos a enviar más comandos. En el Plano Nº3 está representado el diagrama de flujo de esta estructura.

Ilustración 68:Botón stop en modo de control autónomo de App 1.

Ilustración 67: Botón stop en modo de control autónomo de App 1.

BLOQUE II:

PLANOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo final de carrera

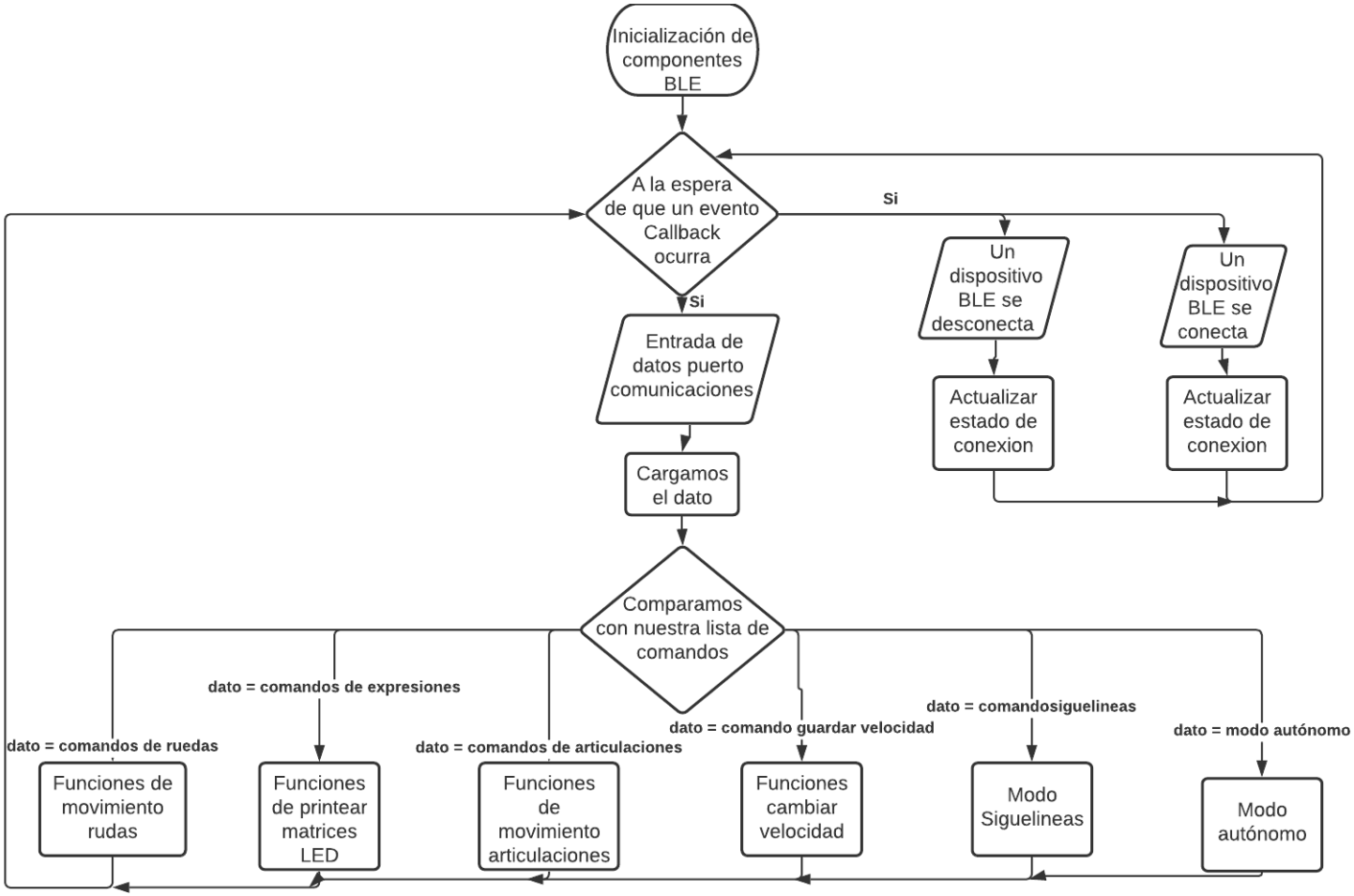
Realizado por Daniel Soto Hernández

Tutorizado por Leopoldo Armesto

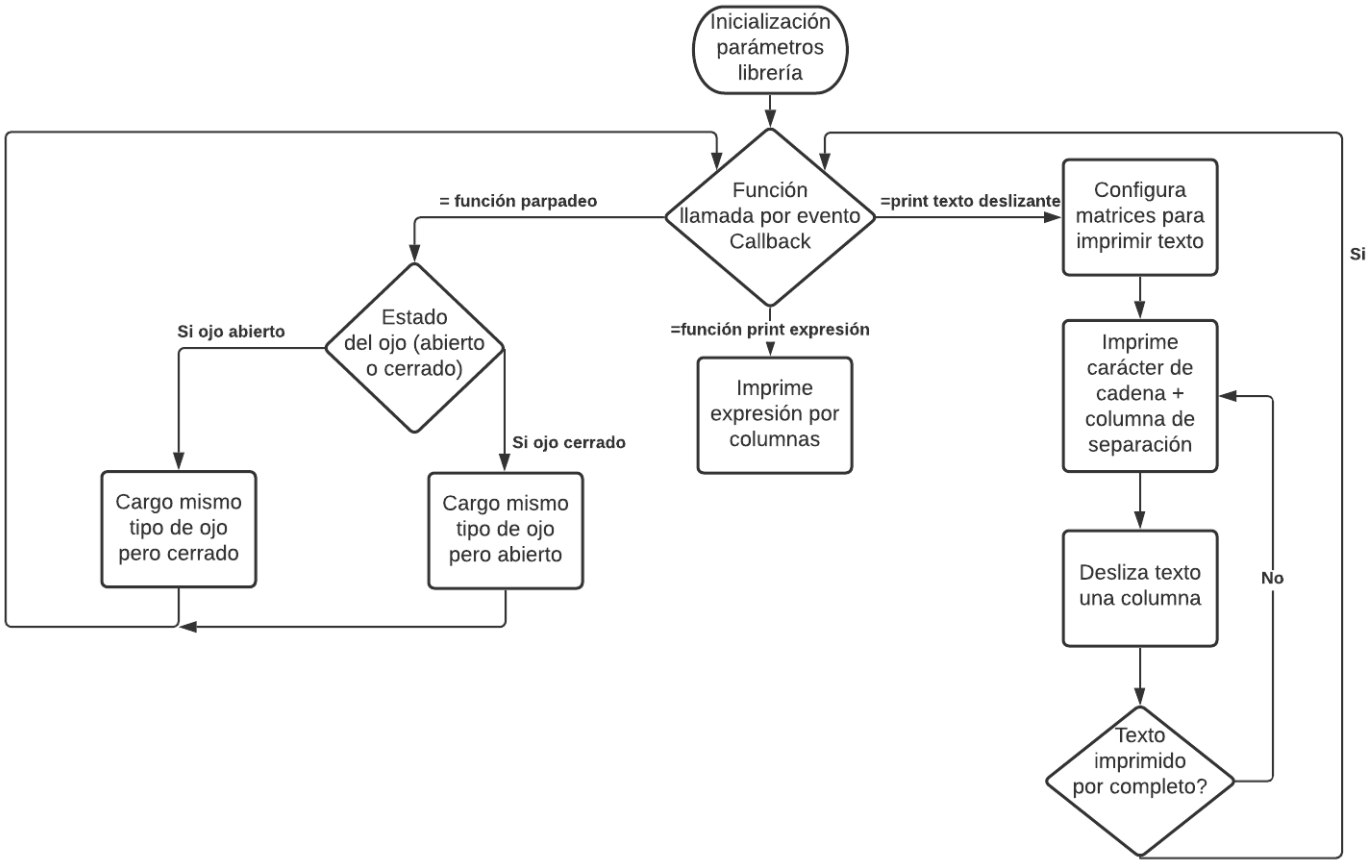
Curso académico 2020/2021

Escuela Técnica Superior de Ingeniería del Diseño

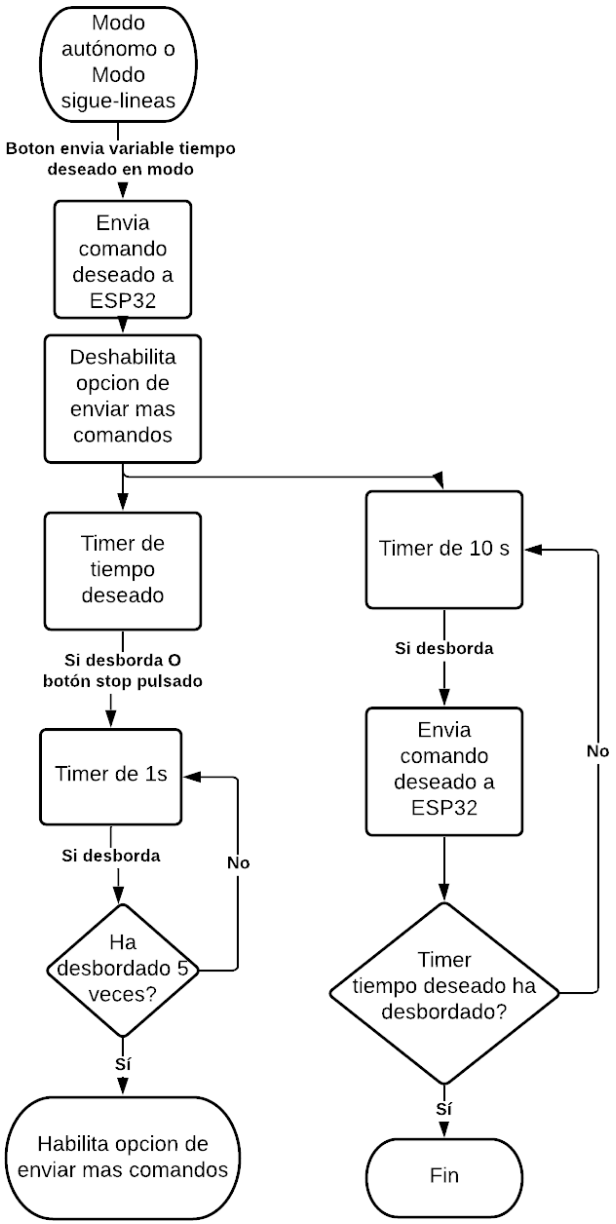
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



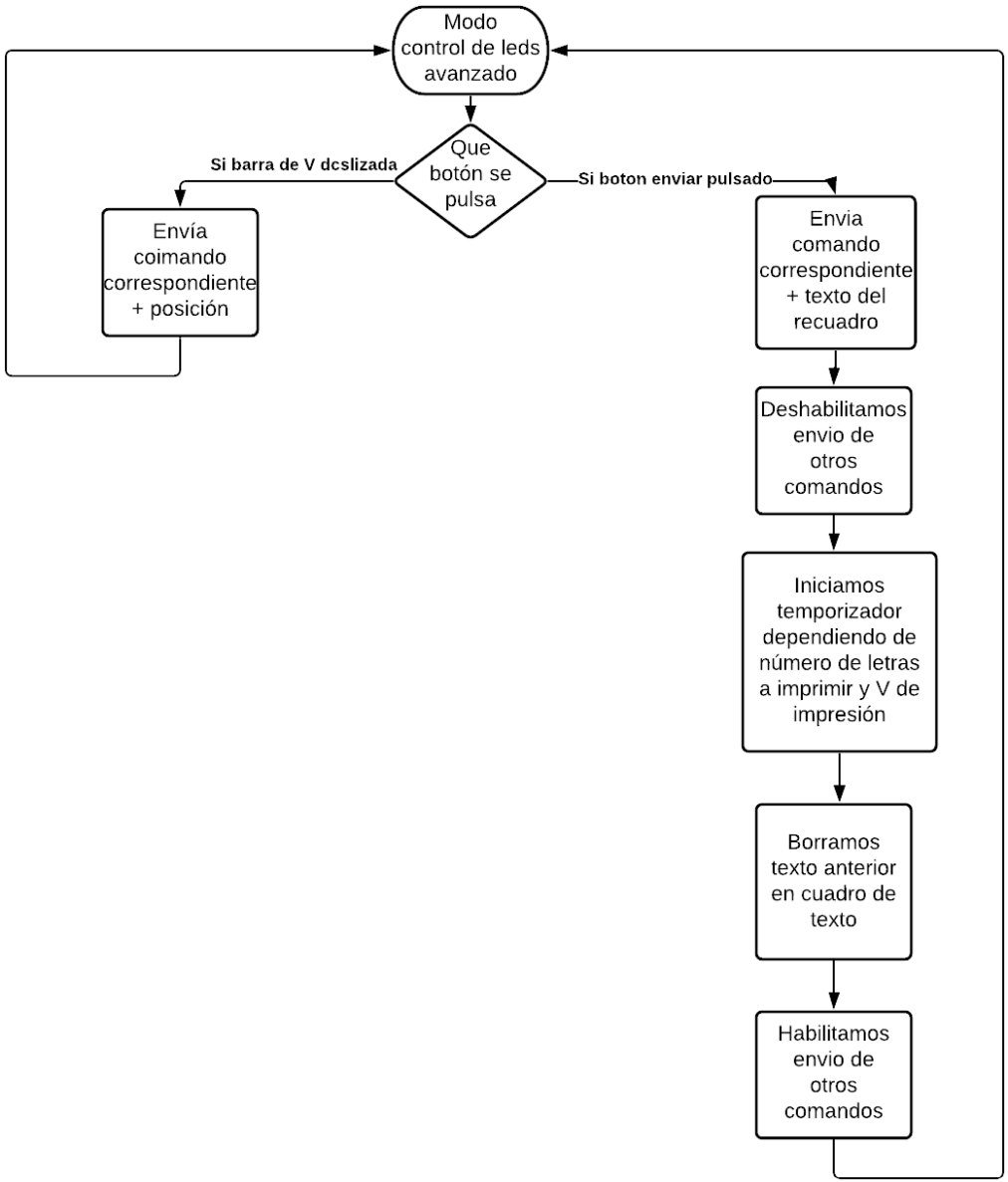
Autor:	Titulo:	
Soto Hernández, Daniel	Diagrama de flujo estructura bluetooth low energy	
Fecha:	Escala:	Nº de plano:
27/06/2021		1



Autor:	Titulo:	
Soto Hernández, Daniel	Diagrama de flujo funcionalidades matrices LED	
Fecha:	Escala:	Nº de plano:
27/06/2021		2

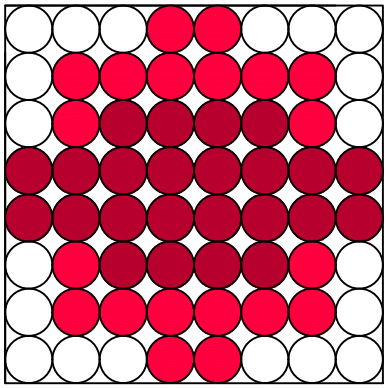


Autor:	Titulo:	
Soto Hernández, Daniel	Diagrama de flujo para envio de comandos de tiempo en modo Autonomo/Sigue-lineas en AppInventor2	
Fecha:	Escala:	Nº de plano:
27/06/2021		3

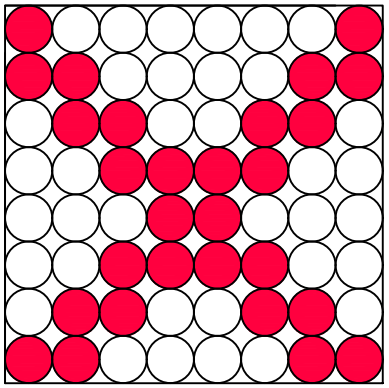


Autor:	Titulo:	
Soto Hernández, Daniel	Diagrama de flujo estructura envio de comandos en modo control de led avanzados en AppInventor2	
Fecha:	Escala:	Nº de plano:
03/07/2021		4

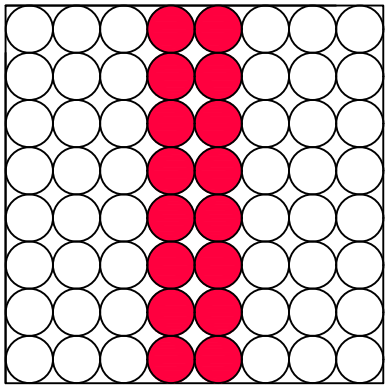
1. Circle



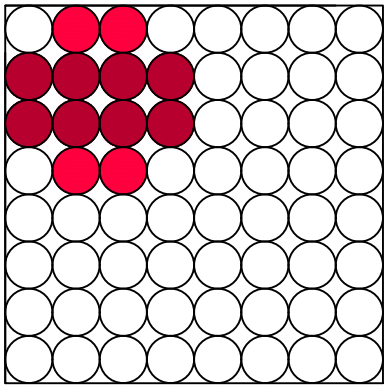
3. X



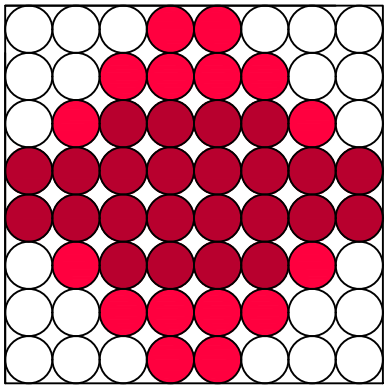
5. Reptil



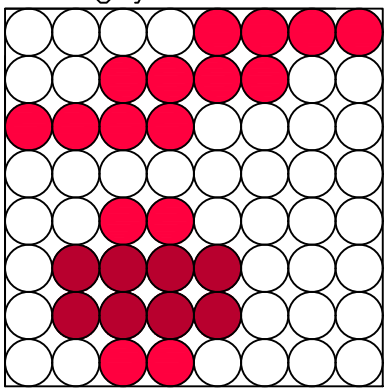
7. Amables



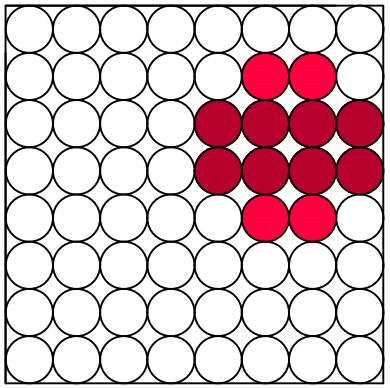
2. Rectangular



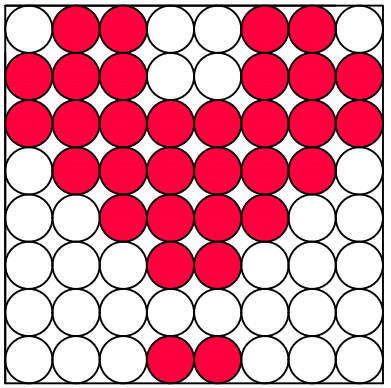
4. Angry



6. Cruzados



8. Corazón

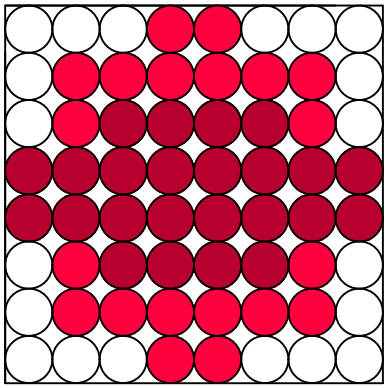


Leyenda:

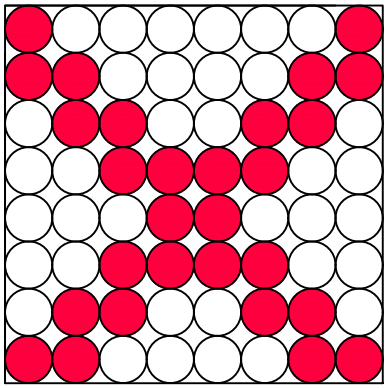
	Ojos abiertos		Ojos parpadeando
Nº	Matriz de datos	Nº	Matriz de datos
1	{0x18,0x7e,0x7e,0xff,0xff,0x7e,0x7e,0x18}	5	{0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18}
2	{0x18,0x3c,0x7e,0xff,0xff,0x7e,0x3c,0x18}	6	{0x00,0x00,0x00,0x00,0x60,0xf0,0xf0,0x60}
3	{0x81,0xc3,0x66,0x3c,0x18,0x3c,0x66,0xc3}	7	{0x60,0xf0,0xf0,0x60,0x00,0x00,0x00,0x00}
4	{0xf0,0x3c,0x0f,0x00,0x0c,0x1e,0x1e,0x0c}	8	{0x66,0xe7,0xff,0x7e,0x3c,0x18,0x00,0x18}

Autor:	Titulo:	
Soto Hernández, Daniel	Disposición de matriz izquierda en cada expresion	
Fecha:	Escala:	Nº de plano:
03/07/2021	3:2	5

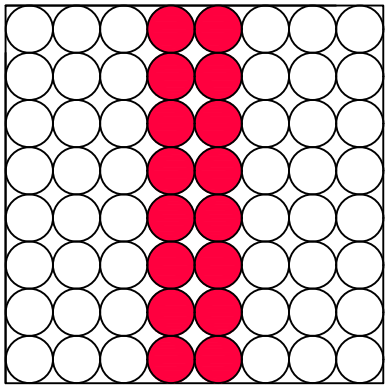
1. Circle



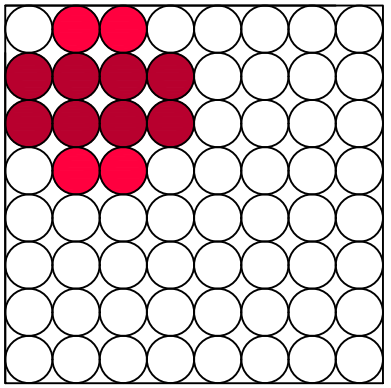
3. X



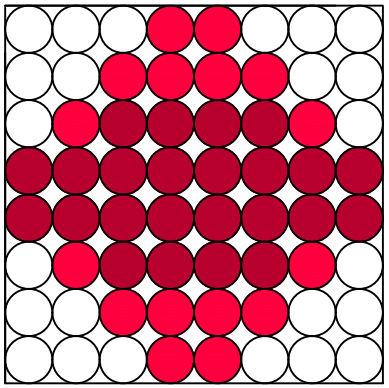
5. Reptil



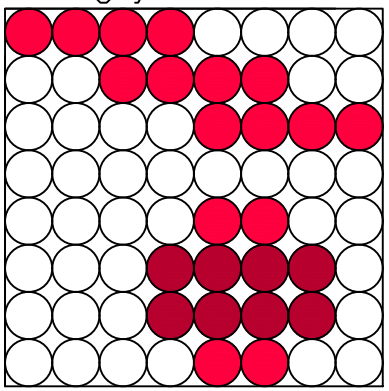
7. Amables



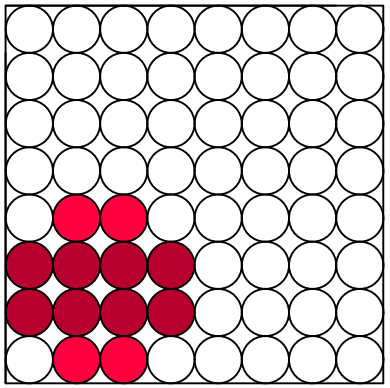
2. Rectangular



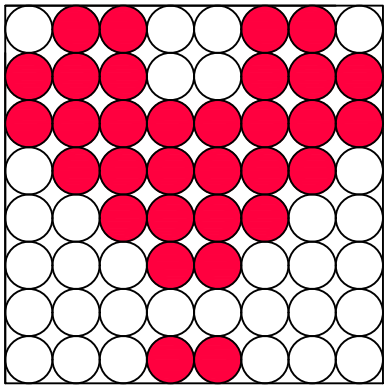
4. Angry



6. Cruzados



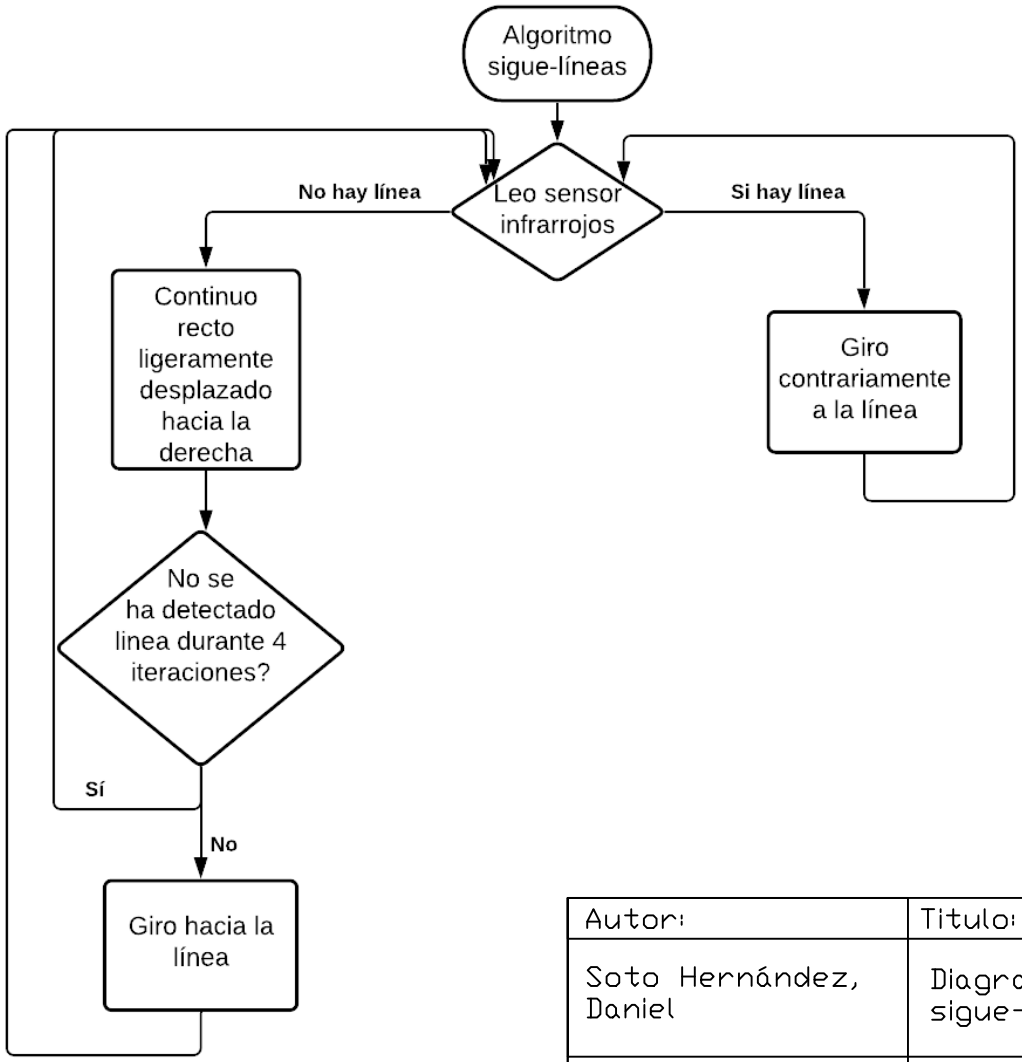
8. Corazón



Leyenda:

	Ojos abiertos		Ojos parpadeando
Nº	Matriz de datos	Nº	Matriz de datos
1	{0x18,0x7e,0x7e,0xff,0xff,0x7e, 0x7e,0x18}	5	{0x18,0x18,0x18,0x18,0x18,0x18, 0x18,0x18
2	{0x18,0x3c,0x7e,0xff,0xff,0x7e, 0x3c,0x18}	6	{0x00,0x06,0x0f,0x0f,0x06,0x00, 0x00,0x00}
3	{0x81,0xc3,0x66,0x3c,0x18,0x3c, 0x66,0xc3}	7	{0x60,0xf0,0xf0,0x60,0x00,0x00, 0x00,0x00}
4	{0x0f,0x3c,0xf0,0x00,0x0,0x78, 0x78,0x00}	8	{0x66,0xe7,0xff,0x7e,0x3c,0x18, 0x00,0x18}

Autor:	Titulo:	
Soto Hernández, Daniel	Disposición de matriz derecha en cada expresion	
Fecha:	Escala:	Nº de plano:
03/07/2021	3:2	6



Autor:	Titulo:	
Soto Hernández, Daniel	Diagrama de flujo del algoritmo modo sigue-líneas	
Fecha:	Escala:	Nº de plano:
27/06/2021		7

BLOQUE III:

PLIEGO DE CONDICIONES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo final de carrera

Realizado por Daniel Soto Hernández

Tutorizado por Leopoldo Armesto, Andrés Conejero y Miquel Cañada

Curso académico 2020/2021

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

III.1. DEFINICIÓN Y ALCANCE DEL PLIEGO

La presente especificación técnica se refiere al diseño, seleccionado, programación y puesta a punto de un robot móvil de propósito general.

Se incluye por tanto todo lo relacionado con la programación del microcontrolador, que la electrónica seleccionada sea adecuada para el propósito utilizado y que el ajuste de parámetros para el correcto funcionamiento del robot sea correcto. Quedando excluidos de esta manera todo lo relacionado con el diseño de la estructura del robot.

III.2. CONDICIONES Y NORMAS DE CARÁCTER GENERAL

- En el proyecto, se deben emplear los materiales y mano de obra que cumplan las condiciones exigidas en las "Condiciones técnicas" del Pliego de Condiciones y se realizarán todos y cada uno de los trabajos contratados, de acuerdo con lo especificado también en dicho documento.

- En el proyecto, se deben seguir los procedimientos indicados en “Condiciones de ejecución” en el proceso de montaje y programación del robot, si se comete algún fallo en algún punto presente en ese apartados será responsabilidad del ingeniero encargado de la verificación del montaje del robot.

- Será de fundamental importancia cumplir con las pruebas estipuladas en el aparatado “Pruebas y ajustes finales o de servicio”, en las cuales se verifica el correcto funcionamiento del robot, y que el robot cumple con las pautas de diseño y funcionalidades estipuladas en el contrato.

- En caso de sustituir algún material se deberá de hacer bajo propia responsabilidad del ingeniero responsable del proyecto, y siempre sustituyéndolo por otro material de especificaciones similares, o en caso contrario realizar los ajustes necesarios en el firmware del proyecto para solventar la posible variación de funcionamiento respecto al proyecto original.

- No es necesario considerar ninguna normativa, debido a que se trata de un proyecto de propósito general, con objetivo experimental y de aprendizaje, quedando así excluido de la aplicación del **Real Decreto 188/2016, de 6 de mayo,** tal y como se ha especificado en el subapartado de la Memoria “Limitaciones y condicionamientos”.

- Todos los trabajos se deberán ejecutar con estricta sujeción a la documentación presentada en el presente proyecto. Se exime de responsabilidad ante posibles negligencias en el montaje de éste.

III.3. CONDICIONES TÉCNICAS

III.3.1. CONDICIONES DE LOS MATERIALES

III.3.1.1. PLACA DE DESARROLLO

DESCRIPCIÓN

Una placa de desarrollo “WeMos D1 ESP32 R32 WROOM-32 WiFi y Bluetooth” versión 1.0.0 junto con una placa de expansión “Arduino Sensor Shield” versión 5.0.

CONTROL DE CALIDAD

Comprobar el funcionamiento de todos los pines con un sensor o actuador comprobado previamente.
Medir con un multímetro las alimentaciones de todos los pines de alimentación de la “Sensor Shield” y comprobar la diferencia de potencial.

En el programa Arduino Ide seguir toda la configuración del entorno de trabajo correspondiente al apartado “5.8 Arduino” de la Memoria.

III.3.1.2. SERVOMOTORES

DESCRIPCIÓN

Dos servomotores de rotación continua “FS90R” para las ruedas.
Tres servomotores de 9g “SG90” para las articulaciones.

CONTROL DE CALIDAD

Llevar a los servomotores a sus límites de movimiento, comprobar que tienen una movilidad de 180º aproximados.
Hacer funcionar los servomotores durante un par de minutos para comprobar si fallan antes de realizar el montaje.

III.3.1.3. BATERIA

DESCRIPCIÓN

Dos baterías de 3.7V y mínimo 3000mA en paralelo, dispuestas en el módulo “16340”, con la distribución indicada en el apartado “6.7 Batería” de la Memoria.
Una power bank común de móvil conectada a través del micro-usb a nuestro micro, siempre y cuando se cumplan con los controles de calidad marcados.
En ningún momento se deberá alimentar el robot por dos vías diferentes ya que podría dañar el dispositivo.

CONTROL DE CALIDAD

Con la batería recargada al máximo comprobar si ofrece la diferencia de potencial indicado con un multímetro entre sus bornes.
Conectar varios actuadores y comprobar que los hace funcionar a todos simultáneamente con suficiente potencia.
Observar que la batería no se sobrecaliente.

Comprobar mediante trayectorias bruscas que la batería no rompa con el correcto comportamiento dinámico del robot debido a su peso o distribución, haciéndolo volcar o lastrándolo en exceso.

Se comprobará antes de conectar el cable usb desde el ordenador a la placa que la batería este apagada, será posible apagarla accionando cualquiera de los dos swich que integra o desconectando el módulo “16340”.

III.3.1.4. MATRICES LED

DESCRIPCIÓN

Se necesitarán dos matrices de led de 8x8 con integrado “MAX721”.
Instalación y uso de las funciones de las librerías que se indican en la Tabla Nº3 de la Memoria.

CONTROL DE CALIDAD

Alimentar y conectar cada matriz led a sus pines correspondientes y ejecutar la función de testeo presente en sus librerías.
Comprobar que todos los leds se iluminan y no hay ninguno fundido.
Comprobar que los cables hacen buen contacto.

III.3.1.5. SENSOR ULTRASONIDOS

DESCRIPCIÓN

Se utilizará un sensor de ultrasonidos “HC-SR04”,
Instalación y uso de las funciones de las librerías que se indican en la Tabla Nº2 de la Memoria.

CONTROL DE CALIDAD

Hacer medición del sensor y comprobar que los datos sean concluyentes aproximadamente.

III.3.1.6. SENSOR INFRARROJOS

DESCRIPCIÓN

Sensor óptico reflectivo con transistor de salida “TCRT5000”.

CONTROL DE CALIDAD

Comprobación de valores tomados por el sensor concluyentes, tanto en su modo detección, conectándolo en un pin digital, como en modo medición, conectándolo en un pin analógico.

III.3.1.7. CABLE MICRO-USB

DESCRIPCIÓN

Cable de carga y transmisión de datos.

CONTROL DE CALIDAD

Comprobación de que nuestro ordenador reconoce el puerto COM donde este conectada nuestra placa en el Administrador de dispositivos.

III.3.2. CONDICIONES DE LA EJECUCIÓN

III.3.2.1. ROBUSTEZ

DESCRIPCIÓN

Se ensamblarán las piezas correctamente, asegurando que el diseño es compacto y no quedan holguras, tornillos sueltos ni ningún factor no considerado en el diseño.
Uso de cola termofusible para evitar vibraciones.

CONTROL DE CALIDAD

Revisar que toda la tornillería está bien apretada de forma visual, antes de cerrar la carcasa por completo.

III.3.2.2. SOLDADURA

DESCRIPCIÓN

En el diseño final se realizarán soldaduras con estaño en todas las conexiones realizadas para así garantizar que no se suelte ningún cable con el robot ya ensamblado y garantizar su correcto funcionamiento.
Comprobación minuciosa con de la Tabla Nº1 de la Memoria previamente a cada conexión.

CONTROL DE CALIDAD

Segunda revisión de la Tabla Nº1 de la Memoria, dando un pequeño tirón a cada cable para comprobar que está correctamente soldado.

III.3.2.3. PROXIMIDAD

DESCRIPCIÓN

Para realizar el manejo del robot o el envío de comandos a través de la App permanecer en un rango cercano para mantener la conexión bluetooth.

CONTROL DE CALIDAD

Permanecer en un rango inferior a 4-5 metros, evitar la presencia de paredes entre el robot y el móvil
En el caso de que el robot no haga caso inmediato de la orden mandada no sobrecargar el canal enviando más comandos de forma excesiva ya que se podría inducir a un error de conexión.

III.3.2.4. DEPURACIÓN

DESCRIPCIÓN

Se recomienda utilizar la visualización del puerto de comunicaciones dentro del programa de Arduino, donde se visualizarán los comandos recibidos a tiempo real. Esto combinado con el sistema de depuración de App inventor 2 permite al programador acotar el origen del error

fácilmente.

III.3.2.5. AJUSTE DE PARÁMETROS

DESCRIPCIÓN

Al tratarse de un hardware económico hace que se presenten diferencias entre dos unidades del mismo componente, lo que se puede traducir en desviaciones de la trayectoria deseada, offsets en medidas de ciertos componentes, desviación de grados de movimiento entre servomotores, movimientos no deseados por pequeñas derivas, etc.

CONTROL DE CALIDAD

Para solventar estos detalles se deberá de ajustar constantes ubicadas en las funciones de movimiento del código según el tipo de error que tengamos con el objetivo de cumplir con las correctas funcionalidades del robot.

III.4. CONDICIONES FACULTATIVAS

III.4.1. OBLIGACIONES Y DERECHOS DEL CONTRATISTA

- El contratista es el único responsable de la ejecución de los trabajos que ha contratado y de las faltas y defectos que, en éstos, puedan existir, por su mala ejecución o por la deficiente calidad de los materiales empleados o aparatos colocados, sin que pueda servirle de excusa el hecho de que hayan sido valoradas en las certificaciones parciales del montaje.

- El contratista tiene la obligación de conocer la normativa aplicable y conocer el proyecto en todas sus partes.

- Presencia o localización de los responsables o sus representantes durante la ejecución del proyecto.

- Obligación de disponer un documento donde se reflejen las indicaciones, aclaraciones o modificaciones del proyecto.

- Obligación de seguir en todo momento las indicaciones del proyecto.

- Obligación de realizar cuantas inspecciones fuesen necesarias para garantizar el correcto desarrollo del proyecto, haciéndose cargo de los gastos que conlleven.

- Obligación del contratista de reponer todos aquellos materiales o trabajos que no se ajusten a las calidades especificadas en el proyecto.

- Derecho a recibir compensación económica por los trabajos realizados no especificados en los documentos del proyecto y necesarios para la correcta ejecución.

III.5. CONDICIONES LEGALES

- El Contratista es responsable de la ejecución del proyecto en las condiciones establecidas en el Contrato y en los documentos que componen el Proyecto (la Memoria no tendrá consideración de documento del Proyecto).

- No se le admitirán reclamaciones de ninguna especie fundada en indicaciones que, sobre el proyecto, se hagan en la Memoria, por no ser este documento el que sirva de base a la contrata. Deberán de estar reflejados en las especificaciones técnicas.

III.6. PRUEBAS Y AJUSTES FINALES O DE SERVICIO

Se dispone en este apartado las pruebas que validan en correcto funcionamiento de las diferentes funcionalidades a ejecutar por nuestro robot, para cumplir con los requisitos se deberán de realizar ajustes en el firmware según lo indicado en el apartado de la Memoria “8. Pruebas y ajustes finales”.

Se deberá de verificar este apartado una vez tengamos todo nuestro robot ensamblado con la disposición de la electrónica final y con la batería cargada al máximo, ya que al hacer ligeras modificaciones podría cambiar los resultados en las pruebas.

Previamente a las pruebas se deberá de garantizar que ordenemos lo que ordenemos a través de la App en ningún momento se producirán picos de corriente que puedan resetear nuestro micro por falta de alimentación, para ello se seguirán las pautas indicadas en el apartado Nº8.1.1. de la Memoria.

III.6.1. CONTROL REMOTO

III.6.1.1. DESCRIPCIÓN

Ambas pruebas se deberán de realizar en todas las diferentes velocidades de movimiento disponibles y en todas las trayectorias y sentidos: arriba izquierda, derecha y abajo izquierda, derecha. Para los botones de rotación no será necesario ninguna verificación, únicamente que gire sobre sí mismo con la velocidad deseada.

- Se dispondrá en línea recta a aproximadamente a un metro de distancia dos puntos de paso, por los que deberá de pasar nuestro robot únicamente pulsando el botón indicado para avanzar en línea recta, verificando así que se ha implementado correctamente el apartado Nº8.1 de la Memoria, donde se ajustan las velocidades preestablecidas a las ruedas.

- Se pondrá a funcionar el robot en trayectorias curvas a todas las velocidades disponibles.

- Se comprobará que todos los comandos de la pantalla de control remoto ejecuten la acción correspondiente en el robot de la forma deseada, además se observará si los ángulos máximos y mínimos de las articulaciones están bien ajustadas, siguiendo con las pautas marcadas en el apartado Nº8.1 de la Memoria.

III.6.1.2. CONTROL DE CALIDAD

- El robot se mueve en línea recta pasando por los dos puntos marcados la mayoría de las veces y en al menos tres de todas las velocidades disponibles, tanto hacia adelante como hacia atrás, en caso de no cumplirse se deberá de realizar modificaciones en las velocidades donde no se cumpla.

- Por otro lado, se verificará de manera visual que el robot gira en las dos direcciones con el mismo radio y brusquedad aproximadamente, tanto hacia delante como hacia atrás, es decir que los coeficientes indicados en el mismo apartado Nº8.1 de la Memoria han sido ajustados correctamente en su correspondiente función.

- En el caso de los servos de las articulaciones se ejecuten los movimientos a la posición deseada sin colisionar con otros elementos del robot y en el caso de las matrices leds que se imprima el dibujo especificado en los Planos Nº5 y Nº6. Se recomienda implementar las funciones de movimiento del cuello para que gire progresivamente y evitar desconexiones de cables por un giro demasiado brusco.

III.6.2. SIGUE-LINEAS

III.6.2.1. DESCRIPCIÓN

Se seguirán las instrucciones para preparar la prueba que se indican en el apartado Nº8.2 de la Memoria.

- Se elegirá el interior o exterior del circuito según hayamos preparado nuestro firmware y colocaremos a nuestro robot en zona blanca apuntando hacia la línea, tras ello seleccionaremos un tiempo de ejecución del modo sigue-líneas y enviaremos el comando desde la App.

- Se priorizará las trayectorias suaves, aunque se desvíen de las líneas del circuito a las trayectorias bruscas que sigan con más precisión la línea.

III.6.2.2. CONTROL DE CALIDAD

- Nuestro robot deberá de completar todo el recorrido hasta llegar al mismo punto sin quedarse bloqueado ni perder la línea, para ello se modificarán los parámetros del

firmware tal y como se especifica en el apartado Nº8.2 de la Memoria. Además, el robot no hará giros bruscos de forma constante, se priorizará una trayectoria recorrida por el robot más suaves y con menos cambios de sentido incluso cuando se desvíe un poco de la trayectoria marcada por la línea.

III.6.3. MODO AUTÓNOMO

III.6.3.1. DESCRIPCIÓN

- En la verificación de que el modo autónomo sea correcto se utilizará una habitación despejada de sillas y mesas, las patas de las cuales probablemente no sean detectadas por el sensor de ultrasonidos y haría colisionar al robot. Se colocará al robot en cualquier posición de la habitación siempre y cuando no esté apuntando directamente a una esquina y se seleccionará el modo autónomo durante un tiempo mínimo de un minuto.

- Se intentará ajustar a través del firmware que el robot se mueva siempre hacia posiciones despejadas.

III.6.3.2. CONTROL DE CALIDAD

- Nuestro robot será capaz de permanecer durante el tiempo indicado sin quedarse atascado con ninguna pared u objeto, y es capaz de corregir su trayectoria en todo momento.

BLOQUE IV: PRESUPUESTO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo final de carrera

Realizado por Daniel Soto Hernández

Tutorizado por Leopoldo Armesto, Andrés Conejero y Miquel Cañada

Curso académico 2020/2021

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

IV.1. DEFINICIÓN Y ALCANCE DEL PRESUPUESTO

A lo largo del siguiente presupuesto se va a aludir a todo el material necesario para llevar a cabo el proyecto, así como el costo de la mano de obra referido al montaje y realización de los controles de calidad que garanticen que el producto se ha ensamblado correctamente y que el desarrollo de sus funcionalidades es el correcto, ajustando parámetros del firmware si fuese necesario.

En otras palabras, en el actual documento se va a presentar el costo real de fabricación del producto clasificado según su naturaleza, incluyendo el ensamblado y conexionado de los componentes por un operario o particular y su ajuste de parámetros dependiendo de las propiedades de su hardware en particular.

Por otro lado, se presentará el costo del diseño y el tiempo empleado en el seleccionado de electrónica y pruebas previas realizadas para tomar decisiones en el diseño final, así como el tiempo de diseño y programación de firmware y software, esta sección se verá reflejada en el apartado Mano de obra indirecta.

IV.2. MATERIA PRIMA DIRECTA

MATERIAL	CANTIDAD	COSTO UNITARIO (€)	COSTO TOTAL (€)
WeMos D1 ESP32 R32 WROOM-32 WiFi y Bluetooth	1	10	10
Arduino Sensor Shield V5.0	1	4,5	4,5
Matriz de LEDs 8×8 max7219	2	3,5	7
Micro servo SG90	3	2	6
Micro servo FS90R rotación continua	2	7,5	15
Rueda FS90R	2	2,5	5
Sensor infrarrojos TCRT500	1	1,5	1,5
Sensor Ultrasonidos HC-SR04	1	1,5	1,5
Ruedas chinas	2	1,2	2,4
Cables con pines hembra-hembra	15	0,25	3,75
Módulo fuente de alimentación	1	10	10
Baterías recargables de litio 3000mA	2	2	4
Plancha de plástico 2mm para corte laser 297x420	1	6,33	6,33
COSTE MATERIA PRIMA DIRECTA TOTAL			76,98

IV.3. MATERIA PRIMA INDIRECTA

MATERIAL	CANTIDAD	COSTO UNITARIO (€)	COSTO TOTAL (€)
Hardware duplicado de repuesto y pruebas	1	76,98	76,98
Cinta aislante negra	1	2	2
Oficina acondicionada con material	20 días	45	900
Cargador baterías	1	2	2
Estaño para soldar	5 g	0,13	0,63
Tubo de cola termofusible	1	0,16	0,16
Proceso de corte laser y estructura	1	20	20
Energía eléctrica	30 kW	0,13	3,91
COSTE MATERIA PRIMA INDIRECTA TOTAL			1005,67

IV.4. MANO DE OBRA DIRECTA

ELEMENTO	CANTIDAD (HORAS)	COSTO UNITARIO (€/h)	COSTO TOTAL (€)
Técnico de taller de electrónica (Montaje)	1	35	35
Técnico de taller de electrónica (Control de calidad)	1	35	35
COSTE MANO DE OBRA DIRECTA TOTAL			70

IV.5. MANO DE OBRA INDIRECTA

ELEMENTO	CANTIDAD (HORAS)	COSTO UNITARIO (€/h)	COSTO TOTAL (€)
Ingeniero electrónico Junior	150	25	3750
COSTE MANO DE OBRA INDIRECTA TOTAL			3750

COSTE TOTAL	4902,65

BLOQUE V: BIBLIOGRAFÍA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo final de carrera

Realizado por Daniel Soto Hernández

Tutorizado por Leopoldo Armesto, Andrés Conejero y Miquel Cañada

Curso académico 2020/2021

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

V.1 BIBLIOGRAFÍA

- JSON: ¿Qué es y para qué sirve?

  NextU LATAM. 2019. *JSON: ¿Qué es y para qué sirve?* [online] Available at: < https://www.nextu.com/blog/que-es-json/ > [Accessed 18 April 2021].

- Matriz LED con Arduino MAX7219

  Guerra, J., 2017. *Matriz de LED 8x8 con Arduino y driver MAX7219*. [online] Programar fácil con Arduino. Available at: < https://www.programarfacil.com/blog/arduino-blog/matriz-led-arduino-max7219/ > [Accessed 5 April 2021].

- Configurando entorno de Arduino

  Hackster.io. 2018. *Wemos R32 with Arduino - Startup Guide!*. [online] Available at: < https://www.hackster.io/NYH-workshop/wemos-r32-with-arduino-startup-guide-7bc841 > [Accessed 30 March 2021].

  Uriarte, I., 2017. *Instalando el ESP32 | Tienda y Tutoriales Arduino*. [online] Prometec.net. Available at: < https://www.prometec.net/instalando-esp32/ > [Accessed 16 February 2021].

- Programación ESP32

  Randomnerdtutorials.com. 2020. *140+ ESP32 Projects, Tutorials and Guides with Arduino IDE | Random Nerd Tutorials*. [online] Available at: < https://randomnerdtutorials.com/projects-esp32/ > [Accessed 7 May 2021].

  Guerra, J., 2019. *⚡ ESP32 Wifi + Bluetooth en un solo lugar*. [online] Programar fácil con Arduino. Available at: < https://programarfacil.com/esp8266/esp32/ > [Accessed 16 March 2021].

  Beunza, J., 2021. *ESP32 conexión BLE*. [online] Juanjobeunza.com. Available at: < https://juanjobeunza.com/esp32-ble/ > [Accessed 26 June 2021].

- Infrarrojos TCR5000 y sigue-líneas

  Juegos Robótica. 2017. *Robótica educativa #17 Programación de robots sigue líneas.*. [online] Available at: < https://juegosrobotica.es/podcast-017/ > [Accessed 8 July 2021].

- Ultrasonidos HC-SR04

  LLamas, L., 2019. *Medir distancia con Arduino y sensor de ultrasonidos HC-SR04*. [online] Luis Llamas. Available at: < https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/ > [Accessed 13 May 2021].

- Redacción de documentos técnicos

  Gómez-Senent Martínez E., González Cruz, M. C., Sánchez Romero M. A. “Cuadernos de Ingeniería de Proyectos II: Del Diseño de Detalle a la Realización”. Ed. S. P. UPV. 1997. [Accessed 23 March 2021]

  Martínez De Pisón Ascacíbar, F. J. y otros. “La oficina Técnica y los proyectos industriales. Volumen II”. Ed. S. P. UR. 2002. [Accessed 5 April 2021]

- Normas de diagramas de flujos

  Es.wikipedia.org. 2021. *Diagrama de flujo - Wikipedia, la enciclopedia libre*. [online] Available at: < https://es.wikipedia.org/wiki/Diagrama_de_flujo#:~:text=El%20diagrama%20de%20flujo%20o,procesos%20industriales%20y%20psicología%20cognitiva. > [Accessed 19 April 2021].

- Información ESP32

  Tech Explorations. 2018. *ESP32 vs Arduino*. [online] Available at: < https://techexplorations.com/guides/esp32/begin/esp32ard/ > [Accessed 17 June 2021].

- Desarrollo de Apps en App Inventor 2

  Appinventor.mit.edu. 2021. *Tutorials for MIT App Inventor*. [online] Available at: < https://appinventor.mit.edu/explore/ai2/tutorials > [Accessed 5 April 2021].

- Bluetooth Low Energy

  Embedded Centric. 2019. *Introduction to Bluetooth Low Energy / Bluetooth 5*. [online] Available at: < https://embeddedcentric.com/introduction-to-bluetooth-low-energy-bluetooth-5/ > [Accessed 25 March 2021].

  Randomnerdtutorials.com. 2021. *ESP32 Bluetooth Low Energy (BLE) on Arduino IDE | Random Nerd Tutorials*. [online] Available at: < https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/ > [Accessed 14 May 2021].

  AFANEH, M. “Intro to Bluetooth low energy”. Ebook 2017. [Accessed 14 March 2021]