UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO FIN DE GRADO

TFG. Nº: **770G01A127**

TÍTULO: **OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO**

AUTOR: JUAN RAMÓN COSTA RODRÍGUEZ

TUTOR: JUAN MANUEL RIVAS RODRÍGUEZ

FECHA: **SEPTIEMBRE 2017**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

TÍTULO: **OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO**

ÍNDICE GENERAL

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**
AVDA. 19 DE FEBRERO, S/N
15405 – FERROL

FECHA: **SEPTIEMBRE 2017**

AUTOR: **JUAN RAMÓN COSTA RODRÍGUEZ**

Fdo.: Juan Ramón Costa Rodríguez

1.1.- ÍNDICE DE FIGURAS

1.2.- INDICE DE TABLAS

TÍTULO: **OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**
AVDA. 19 DE FEBRERO, S/N
15405 – FERROL

FECHA: **SEPTIEMBRE 2017**

AUTOR: **JUAN RAMÓN COSTA RODRÍGUEZ**

Fdo.: Juan Ramón Costa Rodríguez

2.- MEMORIA

2.1.- TITULO DEL PROYECTO

Optimización del algoritmo de un robot tipo laberinto.

2.2.- OBJETO DEL PROYECTO

El presente documento tiene como objeto principal la elaboración de un código que permita a un robot salir de forma eficaz y autónoma de un laberinto, así como la construcción del mismo y el estudio, diseño y construcción del robot.

El robot deberá alcanzar la salida de un laberinto del que no conozca previamente la salida, pero del que si conocerá previamente las dimensiones. Por otro lado, la entrada deberá ser especificada por el usuario, tal y como se indica en la Guía de Usuario que se encuentra en el punto 3.3. del presente documento.

Además de alcanzar la meta, el robot deberá avanzar por las calles del laberinto sin ningún tipo problema.

Se detallarán minuciosamente los pertinentes procedimientos de prueba y sus resultados para demostrar el correcto funcionamiento del robot.

Este proyecto, aparte de lograr alcanzar el objetivo anterior, también pretende ser una plataforma de aprendizaje de la programación en Arduino y del uso de diferentes periféricos típicos de Arduino. Por ello se realiza una introducción a la programación en Arduino y una serie de explicaciones en cuanto a la naturaleza de los diferentes componentes, sus principios de funcionamiento, el control por programa y su función en el robot, que pretenden ser lo más claras y concisas posibles además de la correspondiente justificación de la selección de cada uno de los componentes. El uso del robot deberá ser lo más intuitivo posible para que cualquier usuario que se esté introduciendo al mundo de la programación en Arduino pueda entender su funcionamiento y utilizarlo, e incluso pueda realizar diferentes pruebas con él.

Además de todo lo anterior, este proyecto pretende ser una guía para la construcción de futuros dispositivos como el elaborado, por lo que además de detallar los pasos seguidos para alcanzarlo, se comentan diferentes errores cometidos en la elaboración del mismo y diferentes aspectos a evitar en la elaboración de un proyecto de esta índole para posibles modificaciones.

2.3.- ALCANCE

En este proyecto se analizarán, detallarán y justificarán todos los pasos seguidos hasta la consecución del modelo final del robot, así como las diferentes posibilidades estudiadas, tanto en la elaboración del código, como en la construcción física del robot y del laberinto. Se estudiarán todos los componentes que intervienen en el proyecto, tanto los que conforman las construcciones definitivas, como las alternativas estudiadas, justificando siempre su elección.

Además se realizarán pruebas finales para comprobar su correcto funcionamiento.

Aunque no esté dividido ni organizado de tal forma, el proyecto se podría dividir en cinco fases principales:

- Fase I. Introducción a la programación de Arduino.
- Fase II. Estudio de los componentes que conforman el robot y de sus alternativas.
- Fase III. Construcción del robot definitivo.
- Fase IV. Programación y estudio del algoritmo.
- Fase V. Pruebas finales.

2.4.- ANTECEDENTES

El prototipo a diseñar que se obtiene del análisis y estudio del presente documento estará formado por diferentes componentes, tanto actuadores, como sensores, fuentes de energía eléctrica o circuitos electrónicos. Todos ellos conforman un diseño final compacto que le permitirá al robot cumplir con su

cometido. Pero estos componentes no siempre fueron un requisito básico ni formaron parte todos de las primeras ideas o los primeros conceptos del robot, incluso de los primeros prototipos. A lo largo de la elaboración del proyecto se han estudiado diferentes posibilidades, incluso probado algunas de ellas hasta llegar a alcanzar el diseño final.

En este capítulo se enumerarán todos aquellos aspectos necesarios para la compresión, tanto de la solución final adoptada, como de las diferentes alternativas estudiadas, explicando cada uno de los componentes que han intervenido en la realización de este proyecto.

2.4.1.- PLATAFORMA DE PROGRAMACIÓN

Todo robot con un mínimo de complejidad cuenta, como los seres humanos, con un cerebro sobre el que gire todo su funcionamiento. El encargado de este tan importante cometido será un microcontrolador que lo controle todo.

Un microcontrolador es un circuito integrado programable que actuará en consecuencia de una serie de órdenes almacenadas en su memoria. Todo microcontrolador cuenta, básicamente, con una unidad central de procesamiento (CPU), una memoria y una serie de periféricos de entrada salida. En resumidas cuentas, un microcontrolador trabaja de la siguiente manera: los periféricos de entrada dan señales a la CPU para que, si el programa almacenado en memoria lo indica, actúe en consecuencia, bien sea por programa o dando señales en forma de pulsos eléctricos a su vez a los periféricos de salida. Los microcontroladores leen sobre los sensores y escriben sobre los actuadores.

Para que no haya dudas, un microcontrolador no es lo mismo que un mircoprocesador, si no que el primero es como una versión compacta del segundo, incluyendo todos los elementos en un solo circuito integrado, pero con inferiores características y un inferior precio también.

El robot objeto de este proyecto contará con un microcontrolador. Este podría ser de la familia PIC de microchip o Intel, pero el escogido, como se especifica en los requisitos de diseño, será Arduino.

Arduino no es en sí mismo un microcontrolador, sino una placa de circuito impreso con un microcontrolador, una plataforma electrónica de código abierto, un entorno de desarrollo que facilita la programación de un microcontrolador, con un hardware y un software sencillos.

Esta placa de desarrollo basa su funcionamiento en un entorno de desarrollo denominado Entorno de desarrollo Integrado (IDE), que proporciona un lenguaje propio llamado *Arduino programme language,* basado en el lenguaje de programación *Wired.* Para facilitar su codificación y el manejo por parte de cualquier usuario que, sin grandes conocimientos informáticos quiera empezar a programar, proporciona un entorno de trabajo basado en *Processing,* un lenguaje de programación en código abierto inicialmente concebido para el aprendizaje y que a su vez se basa en el lenguaje C/C++.

Una vez que se escribe un programa o *sketch* en ese lenguaje y con en el mencionado entorno de desarrollo, este se grabará en el micro y se ejecutará continuamente cuando se alimente el Arduino correctamente, sin necesidad de estar conectado al PC.

2.4.1.1.- Hardware Arduino

El hardware de Arduino, además de con un microcontrolador, cuenta con puertos de comunicación, puertos de entrada/salida digitales y analógicos, un pin de alimentación, pines de tierra, un regulador de voltaje, un puerto USB, un puerto Jack de alimentación o un botón de *reset* entre otros.

Generalmente el microcontrolador acostumbra ser del tipo Atmel AVR, concretamente de los tipos Atmega328, Atmega1280 y ATmega8 por su sencillez, pero se está ampliando a microcontroladores Atmel con arquitectura ARM Cortex y también Intel Quark.

Los Atmega pertenecen a la familia AVR, que son una serie de microcontroladores con Reduced Instruction Set Computer (RISC) del fabricante Atmel. Este tipo de arquitectura le permite al sistema contar con un costo reducido y con una gran sencillez, tanto para programar, como para depurar errores.

La composición de un microcontrolador está formada por varios bloques funcionales, basados generalmente en la arquitectura Harvard, que aporta mayor velocidad y menor longitud de programa que una arquitectura del tipo Von Neumann.

En la siguiente figura puede observarse un esquema de la arquitectura interna de un microcontrolador tipo Harvard.

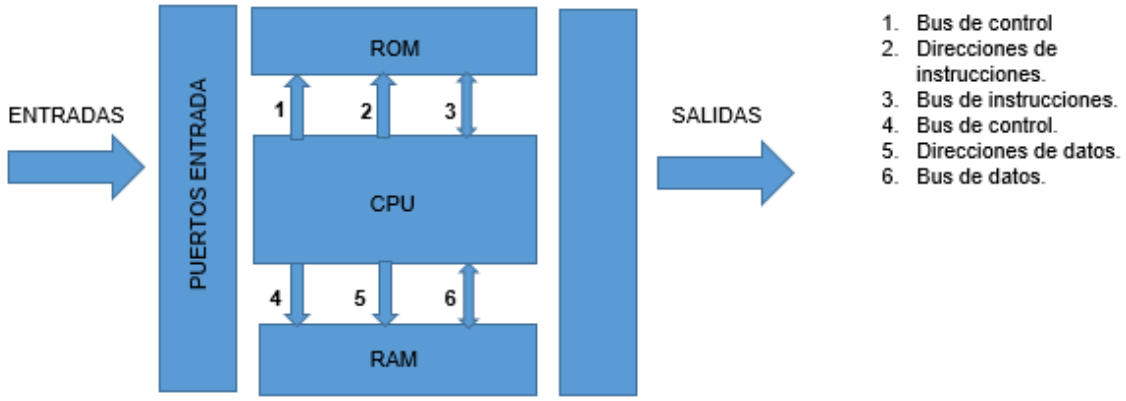


Figura 2.4.1.1.1 - Esquema de arquitectura interna de un microprocesador tipo Harvard.

La CPU será la responsable de ejecutar las instrucciones almacenadas en la memoria. Esta se divide en dos partes, la memoria volátil, conocida como RAM (Random Access Memory) y la no volátil, conocida como ROM (Read Only Memory).

La RAM es una memoria volátil, es decir, que almacena datos temporalmente y que cuando se desconecta de la alimentación los pierde, por lo que también reciben el nombre de memorias temporales. Como su nombre indica son de acceso aleatorio y acceder a cada celda requiere del mismo tiempo de espera.

Las memorias RAM funcionan como memorias de trabajo para el sistema operativo y los programas, en ellas puede leerse y escribirse y básicamente almacenan la información que se va a procesar, pero no como ha de hacerse. Estas instrucciones de ejecución están almacenadas en la memoria ROM.

Las memorias RAM y ROM son los principales tipos de memorias, pero también existen un gran número de alternativas. De todas las alternativas existentes, los diferentes tipos de Arduinos que se encuentran en el mercado, como ya se verán en capítulos posteriores, cuentan con diferentes memorias, que se expondrán a continuación:

- Electrically Erasable Programmable Read-Only Memory (EEPROM). Son memorias no volátiles, similares a las memorias ROM en cuando a volatilidad, pero diferentes en cuanto a su capacidad de modificación, ya que estas son programables y borrables eléctricamente, al igual que una RAM.

La memoria EEPROM en arduino sirve para grabar aquella información que el correspondiente programador quiera almacenar a largo plazo. Por ejemplo, para entenderse mejor la importancia de esta memoria, si como porgramador de una aplicación, se quisiera establecer una alarma, la contraseña estaría almacenada en esta memoria.

- Flash. Memorias basadas en las EEPROM, pero que trabajan con velocidades muchísimo mayores ya que permiten acceder a múltiples celdas de memoria a la vez, al contrario que las EEPROM, que acceden de una en una.

Arduino utiliza esta memoria para almacenar el programa o sketch escrito y cargado por el usuario.

- Static Random Access Memory (SRAM). Memoria similar a una RAM convencional, con la diferencia de trabajar con tecnología basada en semiconductores, lo que le aporta la capacidad de mantener los datos

guardados mientras el circuito esté alimentado, sin necesidad de utilizar circuito de refresco.

La memoria SRAM en Arduino es el espacio de memoria en donde se almacenan las diferentes variables que utilice el programa durante su ejecución.

- Registros. Memorias de reducido tamaño encargadas de almacenar datos de operaciones o resultados de la ejecución de instrucciones. Su tamaño es completamente opuesto a su importancia en el microprocesador, ya que aunque son pequeñas, su importancia es mayúscula para el desarrollo operativo de la actividad de la placa y definen la potencia del mismo en cuanto a velocidad de ejecución. A registros más grandes, mayores operaciones se podrán hacer.

Cuando se habla de que un microprocesador es de 4, 8, 16 o 32 bits, se habla de los tamaños de estos registros.

Como ya se ha comentado, la mayoría de Arduinos utilizan el microcontrolador Atmega. Existe una familia grande de modelos Atmega, por lo que para tomar la importante decisión sobre el modelo de Arduino por el que decantarse, es preciso primero conocer las diferentes características de cada modelo de microprocesador, que vendrán recogidas en la tabla que se encuentra a continuación:

Producto	Flash (KB)	EEPROM (Bytes)	RAM (Bytes)	E/S	SPI	USART	TWI	PWM
MegaAVR								
ATmega48	4	256	512	23	1	1	1	5
ATmega8	8	512	AK	23	1	1	1	3

ATmega88	8	512	1K	23	1	1	1	5
ATmega8515	8	512	512	35	1	1	-	3
ATmega8535	8	512	512	32	1	1	1	4
ATmega16	16	512	1K	32	1	1	1	4
ATmega162	16	512	1K	35	1	2	-	6
ATmega168	16	1K	1K	23	1	1	1	5
ATmega32	32	2K	2K	32	1	1	1	4
ATmega64	64	4K	4K	53	1	2	1	8
ATmega128	128	4K	4K	53	1	2	1	8
ATmega256	256	8K	8K	53	1	2	1	16
ATmega169	16	53	53	53	1	1	-	4
ATmega329	32	53	53	53	1	1	-	4

Tabla 2.4.1.1.1 - Características de los microcontroladores Atmega.

En cuanto al hardware del Arduino es importante también hablar del reloj del sistema. El reloj determina la rapidez con la que el sistema es capaz de ejecutar las instrucciones. Esta velocidad no es igual en todos los microcontroladores y por supuesto, tampoco para todos los tipos de Arduino.

La frecuencia de ejecución de un microcontrolador viene dada en hertzios (Hz) y está determinada por un oscilador de cuarzo o resonador cerámico, unos componentes electrónicos capaces de entregar una señal eléctrica estable que varía entre dos valores, alto y bajo, a elevadas velocidades. Para la mayoría de Arduinos la cadencia es de 16 MHz.

Como ya se explicó, un microcontrolador ejecuta las instrucciones contenidas en su memoria de forma cíclica e infinita. Por ello existe un botón físico para reiniciar el programa que lo que hace es cortar la alimentación.

Además de los bloques de memoria y de procesamiento, un microcontrolador también cuenta con entradas y salidas, que permiten al sistema comunicarse con el exterior.

2.4.1.2.- Comunicaciones Arduino

Una de las bases del funcionamiento del Arduino es su capacidad de interactuar con el exterior, permitiendo implementar aplicaciones tan variadas como leer el estado de un sensor, bien sea físico o analógico, encender un led, tomar una medición de tensión, controlar motores, actuadores… las posibilidades son infinitas. A continuación se explican las diferentes posibilidades de comunicación de dicha placa:

- **E/S digitales**. Arduino cuenta tanto con entradas como con salidas digitales. Estas se encuentran en un zócalo de la placa, numeradas, una detrás de la otra y perfectamente indicadas. Permiten conectar sensores y actuadores que utilicen señales digitales.

  Las E/S digitales distinguen dos estados, alto y bajo. Estos dos estados no son valores fijos, ya que debido a numerosos factores, como el ruido eléctrico, no son siempre fijos, por lo que para solucionar esto se utilizan resistencias Pull-Down y Pull-UP, que fuerzan todo el tiempo a un pin para permanecer en el estado deseado.

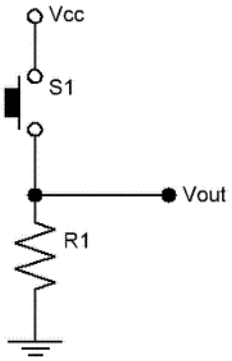


Figura 2.4.1.2.1 – Resistencia Pull-Down.

La resistencia Pull-Down se conecta por un lado a tierra (GND) y por el otro a un interruptor que la conectará o desconectará a los 5 V de alimentación de la placa. La tensión de salida (Vout en la figura 2.4.1.2.1) será de valor lógico HIGH cuando el interruptor esté cerrado y LOW cuando esté abierto.

Por otro lado, para la resistencia de Pull-Up, sucede lo contrario. Cuando el interruptor está abierto la salida está a HIGH y cuando está cerrado a LOW.

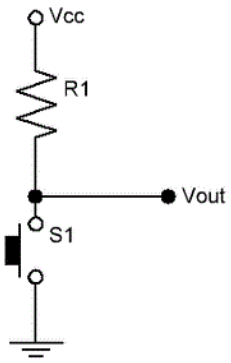


Figura 2.4.1.2.2 – Resistencia Pull-Up.

Utilizar estas configuraciones simplemente se utiliza para dar a una salida al valor digital que se desee, bien sea un valor alto o bajo. Si lo que se quiere es poner una salida a nivel alto, se utilizará la resistencia de Pull-Up y para hacerlo a nivel bajo la de Pull-Down.

Para configurar por programa los puertos digitales existe un juego de instrucciones. En primer lugar debe indicarse si el pin digital funcionará como entrada o como salida de la siguiente manera:

pinMode(12,INPUT)

En primer lugar se escribe el número de pin y en segundo si es entrada o salida (nº pin, INPUT/OUTPUT). Por defecto los pines de Arduino se inicializan como entrada.

Una vez configurado el primer parámetro se configurará el segundo, que define si se utilizará resisencia de Pull-Up o de Pull-Down:

digitalWrite(12,HIGH)

Como en el caso anterior, en primer lugar se indicará el número de pin y en segundo el tipo de resistencia que se desea.

Para leer el valor de un pin se utilizará el siguiente comando:

digitalRead(nº pin)

En cuanto a las señales analógicas, Arduino solamente cuenta con entradas para este tipo de señales. Una señal eléctrica analógica es aquella en la que los valores de la tensión o voltaje varían constantemente y pueden tomar cualquier valora lo largo del tiempo. Un microcontrolador no tiene la capacidad para trabajar con este tipo de señales, ya que solamente trabaja con lógica digital, unos y ceros, por lo que si quiere trabajar con señales analógicas precisa previamente una conversión.

- **E/S analógicas.** Se define como analógica a aquella señal que puede tomar cualquier valor dentro de un rango determinado a lo largo del tiempo. Sensores de temperatura, distancia, pH o intensidad de corriente, entre muchos otros, entregan señales de este tipo.

En el mundo real todas las señales son analógicas, por lo que para trabajar con ellas es preciso convertirlas siempre a un valor digital. Esto se lleva a cabo gracias a los convertidores Analógico-digital (A/D).

Estos convertidores lo que hacen es “clasificar” los diferentes valores que puede dar una señal analógica dentro de 2^N niveles, siendo N el número de bits con los que trabaje el sistema. De esta forma habrá (2^N)-1 intervalos, es decir, si el convertidor AD es de 10 bits, como en la mayoría de placas arduino, la señal que entre por una entrada analógica podrá tomar 1024 valores (2^10), habiendo 1023 intervalos. Cualquier convertidor AD también viene definido por la resolución que presente, que viene definida por la siguiente fórmula:

$$Resolución = (rango\ voltaje\ entrada)/((2\text{^}\mathrm{N}) - 1\ )$$
(2.4.1.1.1)

Cuanto mayor sea el número de bits del convertidor, mayor será el número de intervalos y, en consecuencia, menor será el ancho de este, por lo que se logrará reproducir la señal analógica a convertir con mayor precisión, que se refiere al ancho en mV de cada intervalo. Cualquier cambio en la señal analógica será detectado antes, es decir, la resolución es menor.

Cuanto menor sea el rango de voltaje de entrada, para un número determinado de bits, menor será la resolución.

En la elección de un AD siempre se busca que cumpla con los requerimientos deseados, ni que se quede corta la resolución ni la precisión, pero que tampoco sobre, ya que se estarían infrautilizando recursos, lo que se conoce como precisión relativa.

El Arduino trabaja con valores de referencia de 5V, por lo que si trabajara también con 10 bits, la precisión sería de 4,88 mV, lo que supondría una precisión relativa del 0,1% (1/1024). Si por ejemplo se intentara reconstruir una señal analógica con rango de 0 a 1 V, se estarían infrautilizando los recursos del AD, ya

que la precisión relativa que se obtendría seria del 0,5%, 5 veces mayor que aprovechando todo el rango de la señal, es decir, el AD respondería como si tuviera menos bits de los que en realidad tiene.

El Arduino aporta una solución para solventar este problema, permitiendo cambiar la tensión de referencia con la siguiente función:

AnalogRef(valor de referencia)

A continuación se expondrán los diferentes valores de referencia que se pueden introducir:

Valor de referencia	Valor establecido
DEFAULT	5 V o 3,3 V según modelos
INTERNAL	1,1 V
EXTERNAL	Voltaje aplicado a patilla Vref [0-5V]
INTERNAL1V1	1,1 V (solo en Mega)
INTERNAL2V56	2,56 V (solo en Mega)

Tabla 2.4.1.2.1- Valores para la tensión de referencia del AD.

Más adelante, en el capítulo de resultados finales se comentará y justificará la decisión tomada con respecto a la tensión de referencia.

Volviendo al tema de la lectura de señales analógicas, esta se lleva a cabo mediante una simple instrucción, a la que como en el caso de las señales digitales, hay que indicarle el pin por el que se va a tomar la medida:

analogRead(Nº pin)

Esta función devolverá el resultado de la conversión del AD.

Una vez vistas las entradas y salidas digitales, así como las entradas analógicas, solo queda por estudiar las salidas analógicas. Para generar una señal analógica se utilizan los pines asignados para llevar a cabo la modulación por amplitud de pulso (PWM), que permiten conectar periféricos, como sensores o motores que no trabajen únicamente con 0V o 5V.

Para utilizar las salidas PWM para simular salidas analógicas, en primer lugar es necesario declarar el pin que se va a usar como salida y después utilizar la instrucción correspondiente:

```
pinMode(pinPWM, OUTPUT)
```

```
analogWrite(pinPWM, valor)
```

A la hora de escribir el valor de salida para el PWM hay que tener cuidado con el número de bits que utilice la placa para las salidas.

- **Conectores de comunicación.** Las tarjetas Arduino cuentan con tres puertos de comunicación: el puerto serie, el puerto Inter Integrated Circuit (I^2C) y el puerto SPI. Ninguno de ellos, excepto el puerto serie, cuenta con conectores dedicados para su funcionamiento, si no que comparten su funcionalidad con otros conectores, es decir, trabajan con conectores compartidos, que no son los mismos para los diferentes tipos de Arduino. A continuación se estudiarán los tres puertos de comunicación:

Puerto serie. Punto de conexión de la placa con el ordenador. Es básico para enviar el programa escrito en el PC al Arduino, para que este lo ejecute, además de necesario para un sinfín de posibilidades, como analizar los datos obtenidos por un sensor para comprobar su funcionamiento, observar e tiempo real de ejecución el estado de ciertas variables…

El puerto serie basa su funcionamiento en el envío de una secuencia de bits, uno detrás de otro, entre dos ordenadores o dispositivos. Para ello son necesarios, como en cualquier comunicación, como mínimo, un transmisor (canal

Tx) y un receptor (canal Rx), además de posibles complementos como canales de sincronismo, de referencia de tensión…

En la actualidad existen varios modelos de puertos serie, que suelen recibir el nombre de puertos Universally Asynchronous Receiver/Transmitter (UART), que es una unidad en cargada llevar a cabo las conversiones necesarias tanto en la transmisión como en la recepción de datos serie. Los más conocidos son los puertos USB Universal Serial Port (USB), de los cuales todos los ordenadores cuentan con varios. Otros puertos serie conocidos son los I2C, SPI o Ethernet entre muchos otros.

Todas las placas de Arduino cuentan con al menos una unidad UART y trabajan con la tecnología Transistor transistor logic (TTL). Estos puertos pueden presentarse bien en forma de puerto USB o micro USB, o bien a través de pines de la placa.

I2C. Puertos de comunicación basados en el protocolo que recibe el mismo nombre. Este protocolo surge en los años 80 de la mano de Philips como estándar de comunicación industrial entre microcontroladores y los periféricos y se asienta con éxito en este ámbito gracias a su sencillez y por tratarse de un protocolo abierto.

El funcionamiento de este protocolo, como ya se mencionó, es muy sencillo, basándose únicamente en transmisiones serie y síncronas por medio de dos hilos de control, uno para transmitir los datos, System Data (SDA), y otro que indica cuando deben leerse, el System Clock (SCL).

Esta comunicación sigue el modelo de comunicación maestro-esclavo. Uno de los componentes conectados al bus debe actuar como maestro, que será el que controle el reloj y el otro componente actuará como esclavo. Aunque pueden conectarse a la vez varios maestros, pero solo puede estar activo uno al mismo tiempo. La comunicación es bidireccional, es decir, que sólo puede ir en un sentido o en otro. Por otro lado, los datos y direcciones se transmiten en palabras de 7 bits (A7-A1) más uno (A0), que indica si es lectura o escritura. Esto último e

slo que establece el número de dispositivos que se pueden conectar al bus, ya que si hay 7 bits para direccionar los dispositivos, habrá 127 posibles direcciones y, por tanto, 127 posibilidades de conectar dispositivos.

Cada placa de Arduino cuenta con diferentes pines para utilizar este protocolo.

SPI. Puertos de comunicación basados en el protocolo Serial Peripheral Interface (SPI), que surge en el año 1982, desarrollado por Motorola y utilizado para la transferencia de información entre circuitos integrados en equipos electrónicos.

El protocolo SPI permite una comunicación full dúplex, es decir, permite mantener una comunicación bidireccional mediante el envío y la recepción de mensajes de forma simultánea. Además utiliza una solución síncrona, con una línea de datos para el reloj que regule la comunicación y otra para los datos. El reloj del sistema (CLK O SCK) es generado por el maestro e indica al dispositivo receptor cuando debe leer los datos enviados por el maestro a través de la línea de datos dispuesta para esa función, llamada Master Out Slave In (MOSI) y que conecta la salida del maestro con la entrada del esclavo. Si este último responde deberá hacerlo por la línea dispuesta para ello, la Master In Slave Out (MISO). Además de las líneas de comunicación anteriormente mencionadas existe una cuarta, la línea Slave Select (SS), para seleccionar al esclavo.

2.4.1.3.- Alimentación del Arduino

La placa Arduino precisa estar alimentada para que pueda operar en el desarrollo de sus tareas programadas. Dicha placa está preparada para ser alimentada de varias formas diferentes, bien sea alimentándola con 5V directamente o utilizando el regulador de tensión con el que cuenta.

Los 5V pueden conectarse o bien directamente al pin convenientemente indicado y dispuesto para ello, pero teniendo en cuenta que estos 5V deberán ser exactos, o bien a través de la conexión USB.

Utilizando el regulador de tensión, esta puede entregarse a través de un puerto de tipo Jack para las placas que cuenten con el, o bien a través del pin Vin, sí presente en todos los modelos de Arduino. Ambos pines presentarán la misma tensión sea cual sea el pin alimentado y admiten un rango de tensiones de entre 6 y 20 V, aunque el fabricante recomienda moverse entre los 7 y 12 V para no forzar al regulador.

Dicho regulador permite que fluctuaciones en los valores de la alimentación no afecten al sistema, ya que siempre aporta los 5V con los que trabaja Arduino y con los que se alimenta el microcontrolador del mismo. Estos 5 V regulados también se ofrecen a través de ciertos pines, al igual que otra tensión regulada de 3,3V.

2.4.1.4.- Tipos de Arduino y características principales

Hoy en día existen en el mercado numerosas posibilidades a la hora de adquirir una placa Arduino. Cada uno de los modelos de Arduino presenta diferentes especificaciones y están diseñadas para trabajar en circunstancias específicas.

La elección de la tarjeta se hará a partir del estudio de sus características principales y de las especificaciones requeridas por el comprador.

Velocidad de reloj u oscilador
Tipo de procesador
Tipo de memoria: SRAM, Flash, ROM, EEPRO
Capacidad almacenamiento memorias
Número de entradas y salidas digitales y analógicas
Convertidor Analógico Digital

Características eléctricas (Alimentación, voltaje en pines, corrientes admitidas…)
Comunicaciones
Dimensiones

Tabla 2.4.1.2.2- Principales características de un microcontrolador.

En la tabla anterior pueden verse las principales características que definen a un microcontrolador como Arduino. Dependiendo de estas, el Arduino cumplirá mejor o peor las especificaciones exigidas por el usuario, que suelen centrarse principalmente en los siguientes puntos:

- Número de pines necesarios, bien sean analógicos, digitales o pines PWM..

- Resolución del AD. En ciertas aplicaciones puede ser que la resolución de algunas placas para la conversión analógico-digital se quede corta.

- Convertidor DA. No todas las placas cuentan con un convertidor analógico-digital y a veces es necesario.

- Dependiendo del tamaño de código que se quiera implementar se necesitará mayor o menor memoria flash.

- La velocidad de procesamiento dependerá de la RAM y del microprocesador elegido.

- Voltaje al que trabajen los periféricos que se quieran controlar desde el Arduino.

A continuación se detallarán las características fundamentales de las principales placas presentes en el mercado:

Arduino Uno. Plataforma de programación más extendida y que más tiempo lleva en el mercado, siendo la base para el desarrollo del resto de placas. Es perfecta para iniciarse en la programación en Arduino, ya que cuenta únicamente con los elementos básicos para construir objetos de una complejidad relativamente baja.

Microcontrolador	Atmega328 de 8 bits
Velocidad del procesador	16 Mhz
Memoria Flash	32 KB
Memoria SRAM	2 KB
Memoria EEPROM	1 KB
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	Jack, USB, pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	14
Pines PWM	4
Conectores entrada analógica	6
Voltaje operación pines	5 V
Corriente máxima pines (E/S)	40 Ma
Resolución entradas analógicas (AD)	10 bits
Resolución salidas analógicas (por PWM)	8 bits
Comunicaciones	- Comunicación serie por medio de un puerto USB o dos pines de la placa. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines).
Dimensiones tarjeta	53 x 69 mm

Precio	30 €

Tabla 2.4.1.2.3- Especificaciones Arduino Uno

Arduino Zero. Este Arduino permite acceso a la memoria para depuración del código, tiene capacidad de procesamiento muy alta a altas velocidades, permite conectar ciertos periféricos, posee mayor resolución que otras placas gracias a sus 12 bits y cuenta con una salida analógica con un convertidor DA. Por el contrario trabaja con 3,3 V y solo da 7 mA en los pines.

Microcontrolador	Atsamd21G18 de 32 bits con core ARM Cortex M0+
Velocidad del procesador	48 Mhz
Memoria Flash	256 KB
Memoria SRAM	32 KB
Memoria EEPROM	No disponible, se puede emular con 16 KB disponibles para ello.
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	Jack, USB, pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	14
Pines PWM	Todos los digitales menos 2 y 7 (8 bits como salida)
Conectores entrada analógica	6
Conectores salida analógica	1
Voltaje operación pines	3,3 V
Corriente máxima pines (E/S)	7 mA
Resolución entradas analógicas (AD)	12 bits

Resolución salida analógica (DA)	10 bits (A0)
Resolución salida analógica (por PWM)	8 bits
Comunicaciones	- Comunicación serie por medio de dos puertos USB o dos pines de la placa. Además pueden conectarse móviles, cámaras o periféricos como ratones. - Protocolo I2C(2 pines). - Protocolo SPI (4 pines).
Dimensiones tarjeta	68 x 30 mm
Precio	45 €

Tabla 2.4.1.2.4- Especificaciones Arduino Zero.

Arduino Leonardo

Microcontrolador	ATmega 34u4
Velocidad del procesador	16 Mhz
Memoria Flash	32 KB
Memoria SRAM	2,5 KB
Memoria EEPROM	1 KB
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	Jack, USB, pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	20
Pines PWM	7
Conectores entrada analógica	12 (ampliación a 12 pines digitales a mayores)
Conectores salida analógica	1

Voltaje operación pines	5 V
Corriente máxima pines (E/S)	40 mA
Resolución entradas analógicas (AD)	10 bits
Resolución salida analógica(por PWM)	8 bits
Comunicaciones	- Comunicación serie por medio de un puerto USB o dos pines de la placa. Posibilidad de usar USB2.0. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines).
Dimensiones tarjeta	68 x 30 mm
Precio	20 €

Tabla 2.4.1.2.5- Especificaciones Arduino Leonardo.

Arduino Yun

Microcontrolador	ATmega 34u4
Velocidad del procesador	16 Mhz
Memoria Flash	32 KB
Memoria SRAM	2,5 KB
Memoria EEPROM	1 KB
Rango alimentación recomendado	5 V (no cuenta con regulador)
Puertos de alimentación	Vin(+) y GND(-)
Conectores E/S digitales	20
Pines PWM	7
Conectores entrada analógica	12 (ampliación a 12 pines digitales a mayores)
Conectores salida analógica	1

Voltaje operación pines	3,3 V
Corriente máxima pines (E/S)	40 mA
Resolución entradas analógicas (AD)	10 bits
Resolución salida analógica(por PWM)	8 bits
Comunicaciones	- Comunicación serie por medio de un puerto USB o dos pines de la placa. Cuenta con host USB, que le permite comunicarse con periféricos y utilizar Linux. - Wifi. - Ethernet. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines).
Dimensiones tarjeta	53 x 68.5 mm
Precio	75 €

Tabla 2.4.1.2.6- Especificaciones Arduino Yun.

Además el Arduino Yun cuenta con un microprocesador Linux Atheros AR9331, con las siguientes características:

Procesador	Atheros AR9331
Arquitectura	MIPS @400MHz
Voltaje operativo	3,3V
Ehernet	IEEE 802.3 10/100 Mbits
Wifi	IEEE 802.11b/h/n
Lector MicroSD	Micro-SD only
RAM	64 MB DDR2
Memoria flash	16 MB

Tabla 2.4.1.2.7- Características microprocesador Atheros AR9331.

Arduino DUE

Microcontrolador	AT91SAM3X8E
Velocidad del procesador	84 Mhz
Memoria Flash	512 KB
Memoria SRAM	96 KB
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	Jack, USB, pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	54
Pines PWM	12
Conectores entrada analógica	12
Conectores salida analógica (DA)	2
Voltaje operación pines	3,3 V
Corriente pines (E/S)	130 mA
Resolución entradas analógicas (AD)	12 bits
Resolución salidas analógicas (DA)	12 bits
Resolución salida analógica(por PWM)	8 bits
Comunicaciones	- 4 UART. Comunicación serie por medio de dos puertos USB. Uno para comunicación y programación y otro para ser cliente o host para utilizar periféricos como el ratón o el teclado. También puede comunicarse por medio de ocho pines de la placa. Posibilidad de usar USB2.0. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines), 2 puertos.

Dimensiones tarjeta	53.3 x 101.52 mm
Precio	45 €

Tabla 2.4.1.2.8- Especificaciones Arduino DUE.

Además cuenta con un controlador DMA que permite aliviar a la CPU en tareas que requieran mucha memoria.

Arduino MEGA

Microcontrolador	ATmega2560
Velocidad del procesador	16 Mhz
Memoria Flash	256 KB
Memoria SRAM	8 KB
Memoria EEPROM	4 KB
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	Jack, USB, pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	54
Pines PWM	15
Conectores entrada analógica	16
Voltaje operación pines	5 V
Corriente pines (E/S)	20 mA (40 mA maximo)
Resolución entradas analógicas (AD)	10 bits
Resolución salida analógica(por PWM)	8 bits

Comunicaciones	- 4 UART. Comunicación serie por medio de un puerto USB. También puede comunicarse por medio de ocho pines de la placa. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines), 2 puertos.
Dimensiones tarjeta	101x 53 mm
Precio	50 €

Tabla 2.4.1.2.9- Especificaciones Arduino MEGA.

Arduino Ethernet

Similar al arduino UNO pero con conexión Ethernet y posibilidad de conectar memoria MicroSD.

Microcontrolador	Atmega328 de 8 bits
Velocidad del procesador	16 Mhz
Memoria Flash	32 KB
Memoria SRAM	2 KB
Memoria EEPROM	1 KB
Ranura micro SD	-
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	Jack, USB, pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	14
Pines PWM	4
Conectores entrada analógica	6
Voltaje operación pines	5 V

Corriente máxima pines (E/S)	40 mA
Resolución entradas analógicas (AD)	10 bits
Resolución salidas analógicas (por PWM)	8 bits
Comunicaciones	- Comunicación serie por medio de un puerto USB o dos pines de la placa. - Ethernet. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines).
Dimensiones tarjeta	53.34 x 68.58 mm
Precio	51 €

Tabla 2.4.1.2.10- Especificaciones Arduino Ethernet.

Arduino Fio

Similar al Arduino Ethernet pero con menor frecuencia de reloj y sin conector USB.

Microcontrolador	Atmega328 de 8 bits
Velocidad del procesador	8 Mhz
Memoria Flash	32 KB
Memoria SRAM	2 KB
Memoria EEPROM	1 KB
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	14
Pines PWM	4

Conectores entrada analógica	6
Voltaje operación pines	5 V
Corriente maxima pines (E/S)	40 mA
Resolución entradas analógicas (AD)	10 bits
Resolución salidas analógicas (por PWM)	8 bits
Comunicaciones	- Comunicación serie por medio de un cable FTDI o dos pines de la placa. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines).
Dimensiones tarjeta	43,2 x 8,5 mm
Precio	30 €

Tabla 2.4.1.2.11- Especificaciones Arduino Fio.

Arduino nano

Microcontrolador	Atmega328 de 8 bits
Velocidad del procesador	16 Mhz
Memoria Flash	32 KB
Memoria SRAM	2 KB
Memoria EEPROM	1 KB
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	14
Pines PWM	6

Conectores entrada analógica	10 A
Voltaje operación pines	5 V
Corriente maxima pines (E/S)	40 mA
Resolución entradas analógicas (AD)	10 bits
Resolución salidas analógicas (por PWM)	8 bits
Comunicaciones	-Comunicación serie por medio micro USB o dos pines de la placa. -Protocolo I2C(2 pines) -Protocolo SPI(4 pines)
Dimensiones tarjeta	43,2 x 8,5 mm
Precio	25 €

Tabla 2.4.1.2.12- Especificaciones Arduino Nano.

Arduino micro	
Microcontrolador	Atmega32u4
Velocidad del procesador	16 Mhz
Memoria Flash	32 KB
Memoria SRAM	2,5 KB
Memoria EEPROM	1 KB
Rango alimentación recomendado	7-12 V
Rango alimentación admitido	6 – 20 V
Puertos de alimentación	pin de alimentación (Vin(+) y GND(-))
Conectores E/S digitales	20
Pines PWM	7

Conectores entrada analógica	12
Voltaje operación pines	5 V
Corriente maxima pines (E/S)	20 mA
Resolución entradas analógicas (AD)	10 bits
Resolución salidas analógicas (por PWM)	8 bits
Comunicaciones	- Comunicación serie por medio de micro USB o dos pines de la placa. - Protocolo I2C (2 pines). - Protocolo SPI (4 pines).
Dimensiones tarjeta	18 x 48 mm
Precio	25 €

Tabla 2.4.1.2.13- Especificaciones Arduino Micro.

Otras posibilidades. En el mercado aún existen otras posibilidades como el Arduino BT con bluetooth, Arduino Tian, Arduino industrial 101 o diferentes versiones de los ya explicados como el Arduino Uno Wifi o el Arduino Yun mini.

2.4.1.5.- Instalación software de Arduino

A continuación se mostrarán los pasos a seguir para la instalación del software de Arduino:

- Descargar la última versión del IDE de Arduino desde: **http://arduino.cc/en/Main/Software.**

- Elegir la opción de Windows Installer si el ordenador con el que se trabaja utiliza Windows.

- Aceptar acuerdo de licencia.

- Elegir directorio de instalación.

- Permitir instalar los drivers que solicita.

- El programa ya estaría instalado.

Para más información acerca de la instalación en Windows consultar la guía de instalación **https://www.arduino.cc/en/Guide/Windows**. Si el sistema operativo con el que se trabaja es Mac consultar **http://arduino.cc/en/Guide/MacOSX** y si es Linux **https://www.arduino.cc/en/Guide/Linux** .

2.4.1.6.- Programación con Arduino

En este apartado de los antecedentes se hará una breve y clara introducción del lenguaje de programación en Arduino que permita al lector introducirse en el mundo de la programación de este microcontrolador si es que no lo ha hecho antes. Se explicará lo necesario para entender la estructura básica del programa que se presentará en este proyecto, así como poder modificar o mejorar ciertos aspectos el mismo o para comenzar la elaboración de su propio proyecto. No se pretende entran en detalle en todos los aspectos que involucran a la programación del código del presente proyecto ya que en puntos posteriores se explicará el código exhaustivamente, simplemente acercar al lector a la programación básica en Arduino.

La estructura general de cualquier programa en Arduino se compone de 3 secciones:

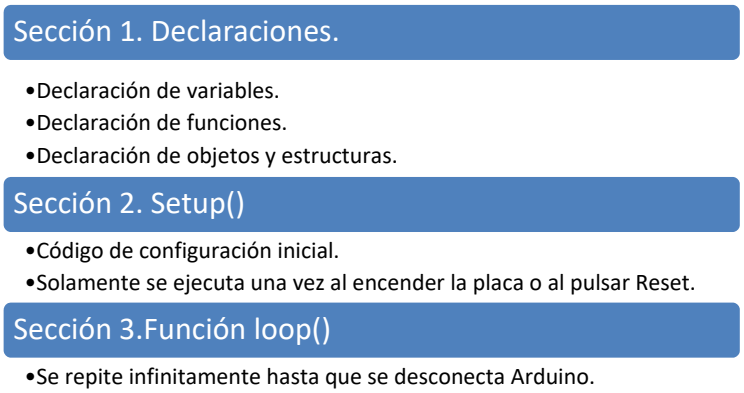


Figura 2.4.1.6.1- Estructura programa Arduino

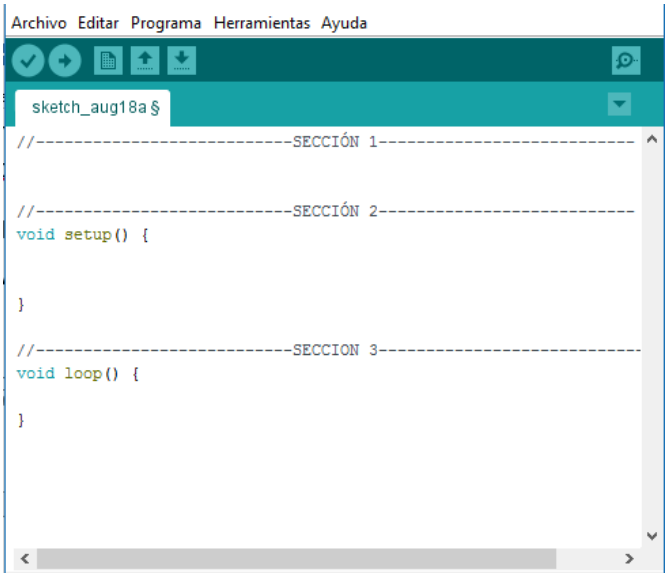


Figura 2.4.1.6.2- Estructura sketch Arduino

Los elementos básicos de sintaxis para la estructura del programa son:

- **;** Utilizado para la separación de instrucciones.
- **{}** Definen el principio y el final de un bloque de instrucciones.
- *//* Comentario en línea única.

- ***/* */*** Comentario multilínea. El comentario va entre cada par de símbolos.

Las variables son estructuras de datos que pueden variar su contenido durante la ejecución del programa.

Desde el punto de vista del ámbito del programa se diferencian dos tipos de variables:

- Variables locales. Declaradas dentro de una función y solo utilizadas dentro de esa misma función.

- Variables globales. Son declaradas en la sección 1 de programa y pueden ser utilizadas en cualquier parte del mismo.

En cuanto a los diferentes tipos de datos que pueden almacenar cada variable, se clasifican en:

- “boolean”: las variables de este tipo ocupan un byte de memoria y solamente pueden tener dos valores: verdadero o falso, 1 o 0, HIGH o LOW.

- “char”: el valor que puede tener es un sólo caracter (letra, dígito, signo de puntuación...). Cada variable de este tipo ocupa un byte de memoria y almacena el número correspondiente al caracter en código ASCII.

- “byte”: el valor que puede tener es siempre un un número entero entre 0 y 255 ya que ocupa un byte de memoria.

- “int”: el valor que puede tomar es un número entero de entre -32768 y 32767 y ocupa 2 bytes de memoria para su almacenamiento.

- “word”: las variables de este tipo ocupan el mismo espacio que las de tipo “int”, por lo que tienen el mismo número de combinaciones numéricas posibles, a diferencia que las de tipo “word” no pueden tomar valores negativos.

- “short”: este tipo de variables utilizan 2 bytes de memoria y pueden tomar valores entre -32768 y 32767.

- “long”: el valor que puede tener una variable de este tipo es un nu´ mero entero entre -2147483648 y 2147483647, por lo que precisa 4 bytes (32 bits) de memoria para almacenarse.

- “unsigned long”: una variable de este tipo puede tener un valor entre 0 y 4294967295, utilizando 4 bytes de memoria para almacenarse.

- “float”: almacenan variables en punto flotante de 32 bits con decimales en un rango entre 3.4028235E+38 a -3.4028235E+38.

- “double”: son exactamente iguales que las variables de tipo “float” pero con doble precisión. Valor máximo de 1.7976931348623157x10^308.

- “string”: almacena una cadena de caracteres.

- “array”: almacena un conjunto de datos del mismo tipo.

Antes de utilizar una variable en el programa siempre hay que declararla. Para ello ha de seguirse la siguiente secuencia:

Figura 2.4.1.6.2- Secuencia para declarar variables.

Un ejemplo sería el siguiente:

Int velodidad_motores = 100;

Arduino también permite declarar variables con un valor constante. Esto se realizará de manera análoga a a declaración de variables, pero el tipo de variable será const.

Operadores aritméticos:

- x++ equivalente a: x=x+1.
- x-- equivalente a: x=x-1.
- x+=y equivalente a: x=x+y.
- x-=y equivalente a: x=x-y.
- x*=y equivalente a: x=x*y.
- x/=y equivalente a: x=x/1.

Operadores de comparación:

- x== comparación de igualdad.
- x!= comparación de diferencia.
- x<y comparación menor que.
- x>y comparación mayor que.

- x<=y comparación menor o igual que.

- x>=y compapración mayor o igual que.

Operadores lógicos:

- &&: Comprueba que las dos condiciones sean ciertas (Operador AND).

- ||: Comprueba que, al menos, una de dos condiciones sea cierta (Operador OR).

- !: Comprueba que no se cumpla la condicio´ n a la que procede (Operador NOT).

Estructuras de control:

- Sentencia if. Estructura de decisión que se utiliza para probar si una determinada condición se cumple. De ser así, el programa ejecutará las ordenes que se le escriban dentro de las llaves de las estructura. De no cumplirse no.

- Sentencia if/else if/else. Evolución de la estructura if que se basa en condiciones encadenadas. Si no se cumple la primera condición, definida con un if, el programa mira si se cumple la siguiente condición, definida con else if. Si no se cumple la siguiente más, se pasa a la siguiente, si no la siguiente… y así como tantos else if se escriban. Por último, si no se cumple ninguna condición el programa ejecutará el contenido del else. Si se cumple alguna condición anterior al else se ejecutan las órdenes pertenecientes a esa condición y se sale de la estructura.

- Sentencia switch. Se trata de una forma más elegante que el bloque “if” para realizar comprobaciones de condiciones múltiples. La sintaxis de esta

estructura se realiza de forma que en primer lugar se escribe la orden switch() y dentro de los paréntesis la variable que se va a comparar para las condiciones. A continuación dentro de los corchetes de la estructura switch se escribirán las diferentes condiciones, de forma que se escribirá la orden case seguida del valor con el que se quiera comparar la variable del switch y de dos puntos (:). Se podrán poner tantos case como se quiera.

Debajo del case se escribirán las instrucciones a ejecutar en caso de que se cumpla esa condición y debajo de estas la sentencia break para cerrar la condición y poner otra o terminar la estructura.

Default será el equivalente a else en la estructura switch.

- Sentencia while. Repite la ejecución de las instrucciones que están dentro de sus llaves mientras la condición especificada sea cierta.

- Sentencia for. Repite la ejecución de las instrucciones que están dentro de sus llaves tantas veces como le sea indicado.

- Sentencia do. Funciona exactamente igual que la sentencia while.

Funciones temporales:

- Milis().Devuelve el número de milisegundos transcurridos desde que la placa comienza a ejecutar el sketch. El dato que devuelce es un unsigned long y se desboradaría en 50 días.

- Delay().Pausa el sketch durante los milisegundos especificados.

Interrupciones Arduino: Arduino dispone de interrupciones para detectar un evento y, en consecuencia, realizar una serie de instrucciones programadas cuando esto ocurra. Esta función es muy importante, ya que en un programa relativamente largo, aunque el Arduino es muy rápido, cada pin será revisado cada cierto periodo de tiempo, por lo que una señal a evaluar en un pin puede

cambiar antes de que el Arduino revise ese mismo pin. Además, revisar el estado de un pin periódicamente, se produce un derroche de consumo de energía y de capacidad de procesamiento.

Existen 2 tipos de eventos que pueden disparar una interrupción:

- Evento hardware.

- Evento programado o timer.

Aunque la mayoría de microprocesadores, como los PIC, soportan interrupciones por software, Arduino no lo hace, siendo esto una de las principales cosas a mejorar en el futuro por los desarrolladores de estas placas.

En cuanto al funcionamiento de una interrupción, cuando un evento dispara una interrupción, la ejecución normal del micro se suspende salvando el estado del microprocesador, tanto los valores de los registros como los flags y salta a ejecutar una función especial que llamamos Interrupt Service Handler o Servicio de gestión de interrupción (ISH).

Una vez que se salta a la interrupción se ejecuta la rutina de servicio de interrupción(ISR). La ISR es una función que no recibe ni devuelve nada y que no admite simultaneidad con otras interrupciones. Cuando la ISR finaliza, el procesador vuelve tranquilamente al punto donde lo dejó y sigue con lo que estaba como si no hubiese pasado nada.

Interrupciones hardware. Se denomina interrupción por hardware a aquella que se dispara cuando se produce un determinado evento en un pin de Arduino, anteriormente declarado para ello, que funcionará como una especie de timbre, de forma que cuando se produzca la señal de disparo anteriormente establecida se detenga la ejecución del programa y se salte a la interrupción. Cada placa de Arduino presenta una serie de pines que además de su funcionamiento habitual pueden funcionar como interrupciones si así se les indica por programa. A continuación podrán verse los pines susceptibles de crear interrupciones para

cada modelo de placa de Arduino:

Placa	Pines
Uno, Nano, Mini	2, 3
Mega, Mega2560, Mega ADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, Yun	0, 1, 2, 3, 7
Zero	Todos los pines excepto el 4
Due	Todos los pines digitales

Tabla 2.4.1.6.4-: Pines con función de interrupción

Aunque no se vean, en cualquier programa de Arduino se utilizan interrupciones, ya que tanto las funciones delay() como milis(), la comunicación I2C o la comunicación serie, entre otras, utilizan interrupciones, aunque visualmente no se vean.

Para declarar una interrupción se usa la siguiente función:

attachInterrupt(interrupt, ISR, mode);

Donde interrupt es el número de interrupción que se quiere utilizar, ISR la función de interrupción y mode se refiere al modo de captura del evento de interrupción. Existen 4 modos de captura:

- Falling: flanco de bajada del pin de interrupción. Cuando s epasa de un valor alto a bajo se llama a la interrupción.

- Rising: flanco de subida.

- Change: flanco de subida o de bajada.

- LOW: se dispara cuando el pin está a cero.

El pin utilizado para la interrupción debe ser declarado como entrada.

Cuando se usan interrupciones hay que tener en cuenta que cualquier variable que se modifique dentro de la ISR y se use también en el programa principal debe estar declarada como volatile, lo que le indica al compilador que esta variable tiene que ser siempre consultada antes de ser usada. Declarar como volatile una variable es más ineficiente que hacerlo de forma convencional, por lo que solamente deben ser declaradas como volátiles aquellas variables que cumplan la condición anterior.

Otros aspectos a tener en cuenta son que durante la ejecución de las interrupciones no se actualizan los valores de las funciones millis() ni micros(), por lo que si el tiempo de ejecución de programa es determinante en la aplicación no se recomiendan interrupciones muy largas. También respectivo a estas funciones, dentro de la ejecución de la ISR se anulan todas las interrupciones, por lo tanto también las funciones delay() y millis(), no así la función micros().

Además de la función de declaración de una interrupción existen más funciones para el tratamiento de interrupciones:

- Interrupts(): habilita las interrupciones.

- noInterrupts(): deshabilita las interrupciones.

- detachInterrupt(): deshabilita una interrupción concreta.

- digitalPinToInterrupt(pin): permite declarar una interrupción poniendo directamente el pin utilizado.

Timers o interrupciones programadas. Estas interrupciones son similares a las explicadas anteriormente, con la diferencia de que las hardware saltaban

cuando se daba una condición y las programadas cuando transcurre un determinado tiempo.

Los timers dependen directamente del tipo de placa que se use, ya que como ya se vio en el apartado de características técnicas de los diferentes tipos de Arduinos estudiados, cada uno trabaja con un procesador de velocidad de procesamiento diferente. La mayoría cuentan con un cristal de cuarzo de 16Mhz, aunque los hay de 32 Mhz, 48 Mhz, 84Mhz…

Teóricamente podría fijarse una interrupción en cada ciclo de reloj de Arduino, pero eso no es viable ya que cada instrucción ocupa varios ciclos de reloj. Por ello se establece un registro para cada timer que indica un entero sin signo que establece cuando debe dispararse la interrupción de dicho timer. El tamaño de este registro también caracteriza los timers de cada Arduino, los hay de 8 bits, 16 bits… y reciben el nombre de Compare Match Registers o CTRs.

Para aportar más flexibilidad a los tiempos Arduino cuenta con preescalers, es decir, divisores, que aumentan la frecuencia de conteo del timer. Los valores típicos de los preescalers son 1, 8, 64, 256 y 1024. Al asignar un timer internamente lo que hace Arduino es multiplicar el valor de ese preescaler por el número de cuentas asignado por el tamaño de registro, alargando de esta forma la cuenta. Cuando la cuenta llega a su fin el registro se desborda, algo que Arduino detecta y, en consecuencia, se salta a la interrupción.

En la siguiente tabla pueden observarse los tiempos de desbordamiento de cada timer para los valores más típicos de velocidad de procesamiento, preescaler y tamaño de bits del registro:

Velocidad procesador	Tamaño registro	Preescaler	Tiempo timer
16 MHz	8 bits	1	16 us
16 MHz	8 bits	1024	16,384 ms
16 MHz	16 bits	1	4,096 ms

16 MHz	16 bits	1024	4,19 s
32 MHz	8 bits	1	8 us
32 MHz	8 bits	1024	8,192 ms
32 MHz	16 bits	1	2,048 us
32 MHz	16 bits	1024	2,08 s

Tabla 2.4.1.6.5.- Capacidad timers según características Arduino.

El procesador marca el tiempo de ciclo del programa:

$$tiempo\ de\ ciclo = \frac{1}{Velocidad\ procesador} \qquad (2.4.1.6.1)$$

El número de bits del registro del timer marca el número de cuentas que tiene que hacer el timer para saltar a la interrupción.

$$nºcuentas = 2^{nºbits} \qquad (2.4.1.6.2)$$

A cada desbordamiento del timer se hace una cuenta. El timer tarda el tiempo de ciclo en hacer una cuenta. Cuando llegue al número de cuentas totales se desborda. Este valor hay que multiplicarlo por el preescaler.

$$tiempo\ desbordamiento\ timer = nºcuentas * t.ciclo * preescaler \qquad (2.4.1.6.3)$$

Por último, en cuanto a este apartado de programación, se comentará una función que aunque más adelante no se usará, puede ser de gran ayuda a la hora de trabajar con el AD del Arduino y con periféricos.

Función map(). Función muy recurrente cuando se desea convertir un valor que se encuentra dentro de un rango determinado a su equivalente para otro rango. Es muy útil cuando se utiliza el AD del Arduino, por ejemplo, cuando se lee un valor analógico. Esto se hará en un rango de 0 a 1024, por lo que si se quiere

escribir a continuación e una salida analógica con un rango de 0 a 255, es idónea la función map(). A continuación se muestra un ejemplo para estos valores:

Valor convertido= Map(valor a convertir, 0,1024,0,255);

2.4.2.- COMPONENTES FINALES

A continuación se tratarán por separado cada uno de los componentes que conformarán conjuntamente el robot definitivo. Se comenzará definiéndolos, tanto internamente como funcionalmente, justificando su elección y su cometido en el robot, así como la implementación de su control para cumplir su función y su colocación física en el mismo y posterior conexión hardware a la placa.

2.4.2.1.- Sensores

Un sensor básicamente es un dispositivo capaz de detectar magnitudes físicas y transformarlas en variables eléctricas.

2.4.2.1.1.- Sensores de ultrasonidos

2.4.2.1.1.1.- Función en el robot

Para la detección de los obstáculos propios del laberinto, en este caso concreto, de las paredes, con el fin de poder avanzar a través de sus calles, se ha elegido una de las opciones típicas a la hora de detectar presencia de obstáculos, los sensores de ultrasonidos. Los sensores de distancia ultrasónicos basan su funcionamiento en la emisión de un haz de ultrasonidos y la posterior recepción del mismo, tras rebotar en el correspondiente objeto que queremos detectar, en nuestro caso la pared.

El robot contará con dos sensores de ultrasonidos. Uno se situará encima del servo y el otro se encontrará fijo en la parte delantera del robot, apuntando al frente. El primer sensor girará acorde al giro del servo, que como ya se verá en su correspondiente apartado, tiene un rango de 180º. Este sensor será el encargado

de controlar las distancias laterales a cada lado según se precise, tanto en el avance longitudinal por un pasillo, como en un giro. El segundo sensor será el encargado de detectar los obstáculos frontales.

2.4.2.1.1.2.- Constitución física y principios de funcionamiento

Los sensores ultrasónicos trabajan basándose en la denominada propiedad magnetoestrictiva de ciertos materiales. Denominamos magnetostricción a la propiedad inherente de los materiales ferromagnéticos (hierro, niquel, cobalto y sus aleaciones) que provoca que estos se deformen mecánicamente ante la presencia de campos magnéticos. Debido a estas deformaciones, se producen una serie de vibraciones y en consecuencia, sonido. En el sensor de ultrasonidos este material estará dispuesto en forma laminar.
Internamente, un material magnetostrictivo está compuesto por regiones polarizadas magnéticamente, que, al ser interferidas por un campo magnético, rotarán, provocando la vibración mencionada anteriormente.

El principio de funcionamiento es sencillo. Básicamente consiste en el envío y posterior recepción de un pulso sonoro. De esta forma, solamente habrá que contar el tiempo que transcurre desde que se envía el haz ultrasónico hasta que se produce la recepción, y por último, mediante una sencilla conversión, se obtienen los centímetros a los que se encuentra el obstáculo a detectar. Esto se tratará más en profundidad en el siguiente apartado.

Hasta ahora se trató solamente la generación del haz sonoro, pero no la recepción del mismo. Esta se basa en el efecto inmediatamente contrario al anterior. Si como ya se ha visto, una corriente provoca una vibración mecánica y esta a su vez un haz sónico, al revés, un haz sonoro, también provocará una vibración y esta, a su vez, una corriente eléctrica.

2.4.2.1.1.3.- Técnica de medición

La técnica de medición consiste en enviar un tren de pulsos de alta frecuencia (imperceptibles para el oído humando) y comenzar una temporización

que indique el tiempo que tarda el eco proveniente del rebote del haz en volver. El tiempo transcurrido entre estos dos sucesos es proporcional al doble de la distancia con el obstáculo, ya que se mide el tiempo que tarda el sonido en ir y en volver. El funcionamiento de estos sensores puede observarse en la figura 2.4.2.1.1.3.1.

Para llevar a cabo la conversión tiempo-distancia, es preciso conocer previamente la velocidad del sonido, que para unas condiciones normales de 20ºC de temperatura, 50% de humedad y presión atmosférica a nivel del mar es de 340 m/s. En unidades más acordes a nuestro rango de trabajo, hablamos de unos 0,034 cm/microsegundos.

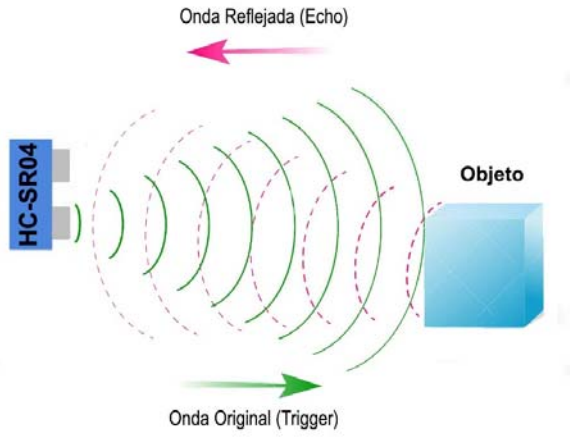


Figura 2.4.2.1.1.3.1- Funcionamiento sensor ultrasonidos

Una vez que se tiene claro que se necesitará la velocidad del sonido para la realización de la conversión y que el tiempo obtenido es el doble del preciso, surge el concepto de ancho de pulso, que es la variable que falta para completar la ecuación:

$$Distancia = \frac{Ancho\ de\ pulso * velocidad\ del\ sonido}{2} \qquad (2.4.2.1.1.3.1)$$

El pulso a enviar, debe ser de por lo menos 10 µs. Una vez que se envíe el pulso a través de la salida Trig del sensor (figura 2.4.2.1.1.3.2) y se esperen 10 µs para cumplir la especificación, el sensor enviará 8 pulsos ultrasónicos de 40 Hz, para después poner a nivel alto la entrada Echo. De esta forma debe medirse el

ancho de esta señal, que durará a nivel alto hasta que se reciba el pulso enviado por el trig anteriormente, momento en el que se pone a nivel bajo. Esta temporización puede hacerse de diversas maneras, arduino aporta un abanico enorme de posibilidades, pero la función escogida fué Pulsein, que mide el tiempo que una señal está en nivel alto (o bajo si es lo que se le indica). El pulso de Echo presenta un rango que engloba valores entre 100 µs y 18 ms, teóricamente, claro está, ya se comentó antes que en la práctica no se cumplían las expectativas en cuanto a rango se refiere. Por otro lado entre medida y medida el fabricante recomienda esperar unos 50 ms después de terminar la cuenta, cosa que cumpliremos con creces en nuestro programa, como ya se comprobará en el correspondiente apartado de programación.

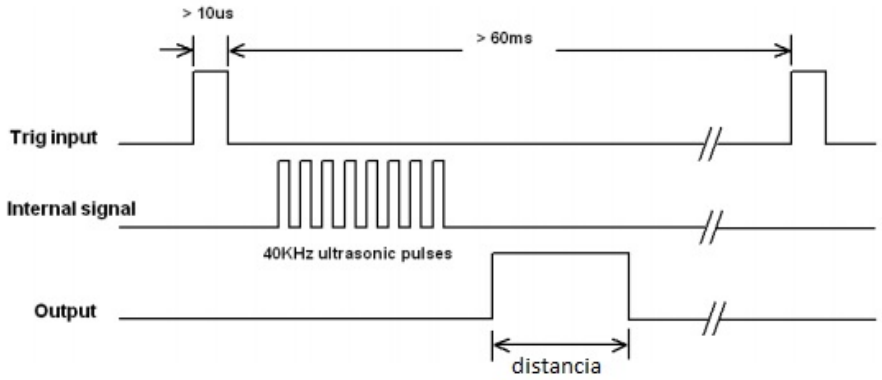


Figura 2.4.2.1.1.3.2- Funcionamiento interno sensor ultrasonidos.

2.4.2.1.1.4.- Modelo seleccionado, funcionamiento y características

Para este proyecto se ha seleccionado el sensor de ultrasonidos HC-SR04. Se trata de un sensor con sistema de emisión/recepción independiente, por lo que cuenta con dos membranas o transductores, el altavoz y el micrófono.

Figura 2.4.2.1.1.4.1-Sensor HC-SR04

La lámina magnetostrictiva emisora será alimentada por una corriente eléctrica, con una serie de pulsos eléctricos, que generará un tren de pulsos ultrasónicos. Por otro lado, la receptora, recibirá los pulsos y generará otros impulsos eléctricos, que el sensor leerá y trabajará en consecuencia a los valores obtenidos.

Esta dualidad emisor-receptor permite medir distancias muy cortas, ya que podemos emitir y recibir al mismo tiempo. El rango de operación teórico de este sensor va de 2 cm a 400 cm, con una resolución de 0,3 cm. En la práctica, sin embargo, es fácil darse cuenta de que la medición real es mucho más limitada, no pasando de los 200 cm. En cuanto a las frecuencias de trabajo, los sensores de ultrasonidos funcionan sobre los 40 kHz, aunque frecuencias a partir de 20 kHz ya se denominan ultrasonidos.

Este sensor no destaca especialmente por su precisión, la cual es bastante baja, pero si lo hace, en cambio, por su bajo coste y su mínimo consumo, de alrededor de 15 mA, así como por su sencillez de uso. A pesar de la comentada falta de precisión, estos sensores sí son efectivos por lo que si se requiere una aplicación en el que la precisión no es determinante, son óptimos.

Como todos los sensores del mercado, ciertos parámetros afectan a su funcionamiento ideal. En nuestro caso, la temperatura ambiente, la humedad y los materiales en los que reflejan la orientación de la superficie a medir, que puede provocar que la onda se refleje, falseando la medición. Además, no resultan adecuados cuando en el campo de medición se encuentran varios objetos, ya que el haz rebota en las superficies generando ecos y falsas mediciones. Tampoco

son apropiados para el funcionamiento en el exterior y al aire libre.

Por último, en cuanto a lo que se refiere a la descripción de este sensor, se mostrará a continuación una tabla que resume sus principales características de funcionamiento, tabla 2.4.2.1.1.4.1:

Alimentación	5V DC
Frecuencia de trabajo	40KHz
Consumo(suspendido)	<2Ma
Consumo(trabajando)	15mA
Ángulo efectivo	<15º
Rango medición	2cm-400cm(200 en la práctica)
Resolución	0.3m
Señal trig	>10us
Dimensiones	45x20x15 mm

Tabla 2.4.2.1.1.4.1- Características del sensor HCSR04.

2.4.2.1.2.- Optointerruptor

2.4.2.1.2.1.- Función en el robot

El robot contará con dos optointerruptores que se siturán en la parte inferior del mismo. Su función es la de contar los pulsos que genera el encoder dispuesto en el eje de la rueda del motor para conocer cuánto avanza el robot y controlar el recorrido por el laberinto.

2.4.2.1.2.2.- Principios de funcionamiento

Un optointerruptor es un sensor capaz de detectar presencia mediante

tecnología infrarroja (IR). El IR es un tipo de luz que los humanos no podemos ver con nuestros propios ojos, no podemos percibirla.

Estas ondas electromagnéticas se caracterizan, principalmente, por su frecuencia, o lo que es lo mismo, por el inverso de esta, la longitud de onda. A continuación se muestra una clasificación según el espectro electromagnético.

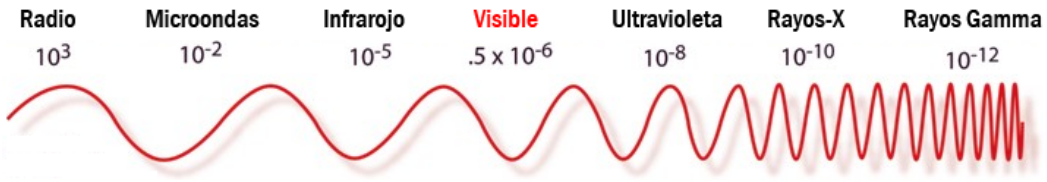


Figura 2.4.2.1.2.1.- Espectro electromagnético.

La luz IR es principalmente radiación térmica. Todo aquel material que esté por encima de 0ºC, irradiará energía en la banda infrarroja.

La estructura de estos sensores presenta forma de “U”, detectando cualquier objeto que pase por su interior. Uno de los extremos contiene un diodo emisor de IR, que como cualquier diodo, cuenta con un ánodo conectado a tensión y un cátodo conectado a masa, de forma que a través de ellos circulará una corriente eléctrica que provocará la emisión de una luz, en este caso. Al otro lado se encontrará un receptor, un fototransistor que recibirá la señal proveniente del emisor.

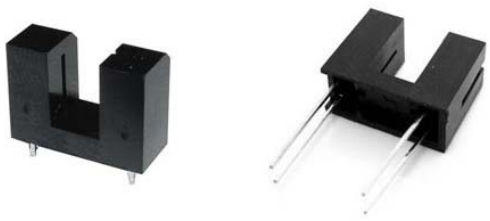


Figura 2.4.2.1.2.2.- Optointerruptor.

El receptor será biestado, es decir, dará un 0 o un 1, dependiendo de si detecta señal IR o no lo hace. Si no la detecta quiere decir que detecta presencia

de un objeto y, si no lo hace, que no hay ningún obstáculo.

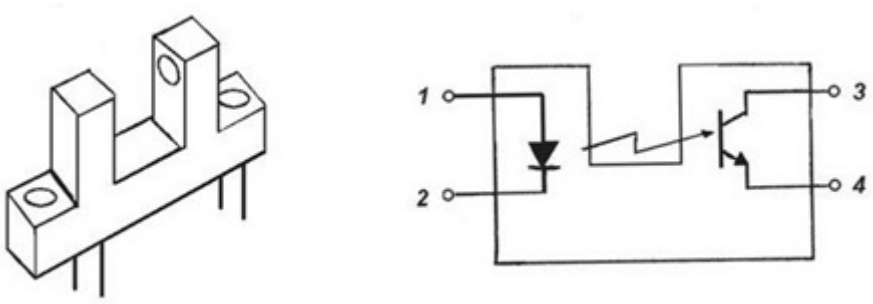


Figura 2.4.2.1.2.3.- Esquema optointerruptor.

Gracias a la característica anteriormente comentada, estos sensores son ideales para medir velocidad de giro de una rueda mediante el uso de una rueda perforada, lo que comúnmente se conoce como encoder, un dispositivo electromecánico que convierte la posición angular de un eje, directamente a un código digital. Existen dos tipos de encoders, los absolutos o los relativos, también llamados incrementales. Los incrementales sólo indican que la posición ha cambiado, mientras que los encoders absolutos indican la posición específica, ya que utilizan una codificación para cada posición, lo que significa que proporciona tanto la indicación de que la posición ha cambiacambio de posición, como la indicación de la posición absoluta del encoder.

En el caso de esta aplicación se usará un encoder incremental, que consiste en un disco ranurado que girará solidario al eje del motor. Este se colocará de tal forma que las ranuras ocupen la posición central del sensor (el centro de la “U”).

Cuando la rueda gire, lo hará el disco ranurado. El sensor IR detectará las ranuras, por lo que contando las ranuras detectadas en un espacio determinado de tiempo, se podrá conocer, tanto la velocidad de giro, como el espacio recorrido por el robot. Para conocer el sentido de giro sería preciso colocar dos optointerruptores, lo que se conoce encoder incremental en cuadratura, de forma que un sensor ocupe una ranura y el siguiente el espacio contiguo a esta para saber en que orden detectan la ranura y así conocer el sentido de giro. Para este proyecto no es importante conocer el sentido de giro, por lo que llegará con un

solo optointerruptor.

2.4.2.1.2.3.- Modelo seleccionado y características

El modelo seleccionado es el Modulo Sensor FZ0888 con un sensor IR S0038. Cuenta con una patilla de alimentación, otra de masa y una salida digital y otra analógica, que está deshabilitada. La salida digital es la que indica si se detecta presencia, poniéndose a nivel alto o no, caso en el que estaría a nivel bajo.

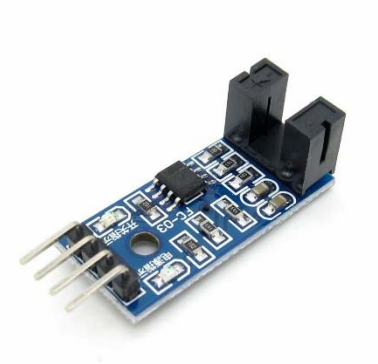


Figura 2.4.2.1.2.3.1- Modulo Sensor FZ0888

A continuación se muestran las principales características de este módulo:

Alimentación	3,3V - 5V DC
Consumo máximo	20 Ma
Dimensiones	30x14 mm

Tabla 2.4.1.2.3.1.- Características del optointerruptor.

En cuanto al encoder, el modelo seleccionado es de 20 ciclos por revolución (CPR).

Figura 2.4.1.2.3.2.- Encoders.

2.4.2.1.3.- Receptor de infrarrojos

2.4.2.1.3.1.- Función en el robot

La función del receptor de infrarrojos es la de recibir las señales provenientes de un mando para controlar de forma remota ciertas funciones del Arduino. El receptor recibirá las señales infrarrojas del mando, las tratará debidamente y se las enviará al Arduino, que responderá en consecuencia.

2.4.2.1.3.2.-Principios de funcionamiento

El funcionamiento de un receptor de infrarrojos se basa, como su propio nombre indica, en la tecnología IR, al igual que el optointerruptor ya visto en el punto anterior, donde ya se comentaron ciertos aspectos de la luz IR. Ambos sensores se basan en el mismo principio, pero difieren en la forma de tratarlo. Mientras que los optointerruptores basaban su funcionamiento en detectar interrupciones de un haz de luz IR, los receptores de infrarrojos utilizan los haces de luz IR como sistema de transmisión de información.

Para transmitir el mensaje se utiliza el mismo principio que se utiliza en las ondas de radio o en cualquier señal radioeléctrica que se transmita por el aire. Se envía un tren de ondas estable, lo que se denomina portadora, que se mezclará con la información que se quiera enviar, es decir, la información se envía modulada sobre una portadora. Esto se hace para mejorar el rechazo al ruido y a la luz ambiental.

El encargado de transmitir la señal infrarroja será un transmisor, que contendrá un transistor que controlará un led infrarrojo, de forma que se controle el envío o no de los haces IR. El transmisor de luz IR más común es un mando

convencional como el de las televisiones. Cualquier mando cuenta con un procesador que gobierna la generación de la señal y la mezcla con la portadora, lo que se conoce técnicamente como modular. La acción opuesta, demodular, será la que lleve a cabo el receptor.

Como en cualquier transmisión el mensaje tiene que seguir unas determinadas normas (forma de los pulsos, duración, contenido…) que deben ser conocidas tanto por el emisor como por el receptor para que la comunicación sea correcta. Sin embargo, no existe un único protocolo adoptado como estándar. En su lugar cada fabricante ha desarrollado los suyos propios. Así, tenemos el RC-5 y RC-6 de Philips, el SIRC de Sony, el protocolo NEC desarrollado por la compañía japonesa Nippon Electronic Company, entre muchos otros. La frecuencia de la onda portadora depende de protocolo empleado pero, en general, varía entre 36-50 kHz, siendo el más habitual en torno a los 38 kHz.

Como cada empresa que utilizaba tecnología IR para utilizar sus productos tenía su propio protocolo, al final, con el paso de los años, se empezaron a diseñar receptores capaces de trabajar con múltiples protocolos.

2.4.2.1.3.3.-Modelo seleccionado y características

El modelo seleccionado es el receptor AX-1838HS. Este receptor se comercializa en forma de módulo, con conexión para un sensor infrarrojo de tres terminales, un demodulador en la banda de 36-38kHz, un filtro PCM (Pulse Code Modulation) y preamplificación para rechazo de luz ambiental. Está diseñado para soportar diversos protocolos de comunicación.

Figura 2.4.2.1.3.3.1- Receptor AX-1838HS.

La mayoría de los protocolos IR utilizan una señal portadora del rango de microsegundos, por ejemplo e NEC, de los más utilizados, 26 us. El método de trabajo es similar en todos los protocolos, aunque cada uno utilice valores diferentes. Siguiendo con el ejemplo del NEC, se diferencia un 0 de un 1 por la secuencia de pulsos que se utiliza. A continuación puede verse en la siguiente figura como se distingue un 0 de un 1:

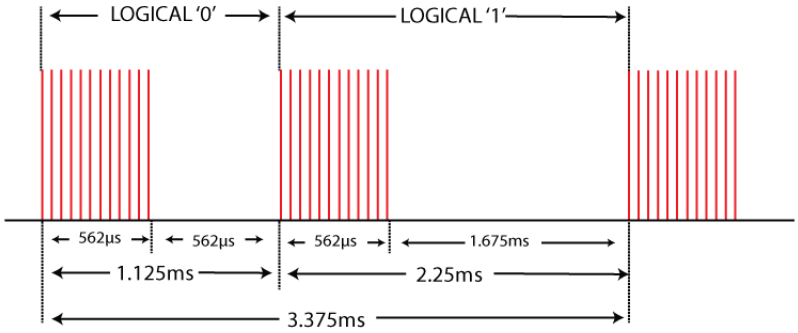


Figura 2.4.2.1.3.3.2- Generación binario protocolo NEC

A partir de esta codificación de los unos y de los ceros se envían los mensajes. Para posibilitar la comunicación emisor-receptor se utilizan paquetes de 8 bits y una secuencia concreta, con inicialización del mensaje, dirección del receptor y comando enviado. A continuación podrá verse un ejemplo de comunicación IR del protocolo NEC:

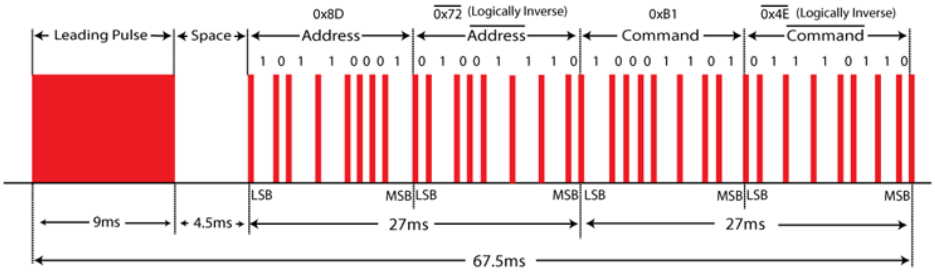


Figura 2.4.2.1.3.3.3- Ejemplo comunicación protocolo NEC.

El receptor recibirá este mensaje y lo demodulará, es decir, traducirá el mensaje modulado anteriormente por el mando a unos y ceros, señal que recibirá el Arduino.

A continuación pueden observarse las principales características de este módulo:

Alimentación	2.1 – 5.5V DC
Consumo máximo	1.5 Ma
Dimensiones	30x14 mm

Tabla 2.4.2.1.3.3.1.- Características receptor AX-1838HS.

2.4.2.1.4.- Sensores de infrarrojos

2.4.2.1.4.1.- Función en el robot

La función de los sensores de infrarrojos será la de permitir al robot funcionar como un seguidor de líneas.

2.4.2.1.4.2.- Principios de funcionamiento

Al igual que los dos últimos sensores vistos, estos sensores basan su funcionamiento en la luz infrarroja. Su funcionamiento es más parecido al del opto

interruptor, diferenciándose únicamente en la disposición del led emisor del luz infrarroja y del fototransistor receptor.

2.4.2.1.4.3.- Modelo seleccionado y característico

El modelo seleccionado es el módulo sensor TCRT5000. Este módulo cuenta, además de con el emisor y el receptor IR, con un comparador LM393 que permite conectar directamente el sensor al Arduino y enviarle señales de presencia o ausencia de luz IR a través de la salida D0, que estará a nivel alto si el receptor no recibe luz. El módulo tiene un potenciómetro para el ajuste de la sensibilidad y cuenta con LEDs que indican el estado de alimentación y del sensor.

El emisor de luz y el receptor se colocan en la misma dirección para detectar la presencia de un objeto utilizando la reflexión del infrarrojo sobre el objeto.

Alimentación	3,3-5 V DC
Consumo máximo	20 mA
Dimensiones	32x14 mm

Tabla 2.4.2.1.4.3.1- Características módulo sensor TCRT5000.

2.4.2.2.- Actuadores

Según indica la Asociación de la Industria Eléctrica (AIE), “Un actuador es un dispositivo inherentemente mecánico cuya función es proporcionar fuerza para mover o “actuar” otro dispositivo mecánico”. Esta fuerza puede provenir de diferentes fuentes, como la presión neumática, la presión hidráulica o la fuerza electromotriz eléctrica, siendo esta última la que proporcionan los actuadores que se utilizarán en este proyecto.

2.4.2.2.1.- Servomotor

2.4.2.2.1.1.- Función en el robot

El robot contará con un servo en la parte delantera, sobre el que irá colocado un sensor de ultrasonidos.

2.4.2.1.3.2.-Principios de funcionamiento

Al igual que los últimos dos sensores explicados, este sensor también basa su funcionamiento en la luz IR. Su funcionamiento es más parecido al del optointerruptor.

2.4.2.2.1.2.- Constitución física y principios de funcionamiento

Básicamente, un servo, a partir de una orden recibida, provoca una fuerza electromotriz. Internamente un servo está compuesto por diferentes componentes electromecánicos y electrónicos, como son un motor de corriente continua, una caja reductora, un circuito de control y un potenciómetro que le permite saber la posición en la que se encuentra y así poder controlarla. Esto último es lo que le otorga al servo su característica y principal ventaja sobre otros motores de corriente continua (CC), y es que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación y mantenerse estable en dicha posición, mientras que los motores CC convencionales giran sin parar hasta que se corta el suministro de corriente. A su vez también podremos controlar la velocidad de giro. Además los servos nos aportan mayor torsión o torque que los motores CC.

2.4.2.2.1.3.- Modelo seleccionado y característico

Existen diferentes modelos de servo para Arduino, pero el seleccionado fue el modelo Micro Servo 9g SG90 de Tower Pro (figuras 2.4.2.2.1.3.1 y 2.4.2.2.1.3.2), ya que se trata de un servo sencillo y barato cuyas características cumplían de sobras los requerimientos pretendidos.

Figura 2.4.2.2.1.3.1- Servo 9g SG90

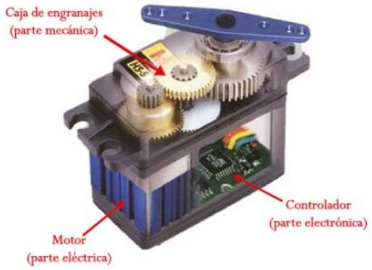


Figura 2.4.2.2.1.3.2.- Estructura Servo.

A continuación, se muestra una tabla resumen con las principales características:

Alimentación	3.5V ~ 6V DC
Consumo	250-650 mA depediendo de la carga
Torque	1.8 Kg-cm a 4.8V
Velocidad (sin carga)	0.10 sec/60° a 4.8V
Ancho de pulso	500-2400 us
Rango rotación	0-180°
Resolución	1º
Dimensiones	23x12.2x29 mm

Tabla 2.4.2.2.1.3.1.- Características del servo 9g SG90

2.4.2.2.1.4.- Técnica de control del servo y programación

Para trabajar con este servo se utilizan los pines Pulse-Width Modulation (PWM), modulación por ancho de pulsos, de nuestro Arduino Mega. Como ya se ha especificado anteriormente en su correspondiente apartado (2.4.1.2.), los pines digitales de Arduino son biestado, alto (5V) o bajo (0V). Sobre esta idea se desarrolla el concepto de este tipo de modulación, que pretende controlar los tiempos de estado alto y bajo para obtener una onda cuadrada, donde el tiempo que la señal esté a nivel alto, será el ancho de pulso. A mayor ancho de pulso

mayor será la cantidad de tiempo que el motor recibe tensión. Si el motor recibe tensión de forma constante, este gira a su máxima velocidad y potencia. Con PWM lo que se hace es aplicar la máxima tensión, pero no todo el tiempo, si no a pulsos, con lo que se consigue regular la velocidad manteniendo la potencia del motor. Otro sistema de control podría ser una regulación de la tensión que se aplica al motor de forma que cuanto menos tensión, menos velocidad este sistema es cuenta con un gran inconveniente, y es la pérdida de potencia que provoca.

Como ya se ha dicho, el ángulo de giro del servo se controlará con una señal PWM, que recibirá el circuito electrónico que controlará el movimiento y posición del servo, moviendo el eje del motor, que a su vez está acoplado a una caja de engranajes. La presencia de este sistema de engranajes hará que cuando se mueva el eje motor se produzca una inercia muy superior a la de un motor común y corriente. El eje motor también está acoplado a un potenciómetro, formando un divisor de tensión típico de cualquier potenciómetro. Este divisor dará una tensión proporcional al ángulo de giro del servo. En nuestro caso el servo tendrá un rango de 0° a 180° y una resolución de 1°.

A continuación se muestran una serie de diagramas de control del servo (figuras 2.4.2.2.1.3.3, 2.4.2.2.1.3.4 y 2.4.2.2.1.3.5):

Figura 2.4.2.2.1.3.3.- Diagrama control servo.

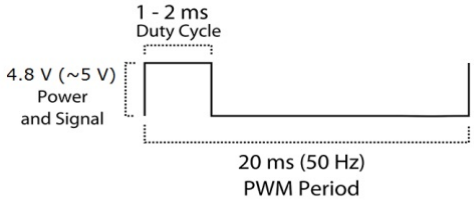


Figura 2.4.2.2.1.3.4.- Entrada PWM del diagrama de control del Servo.

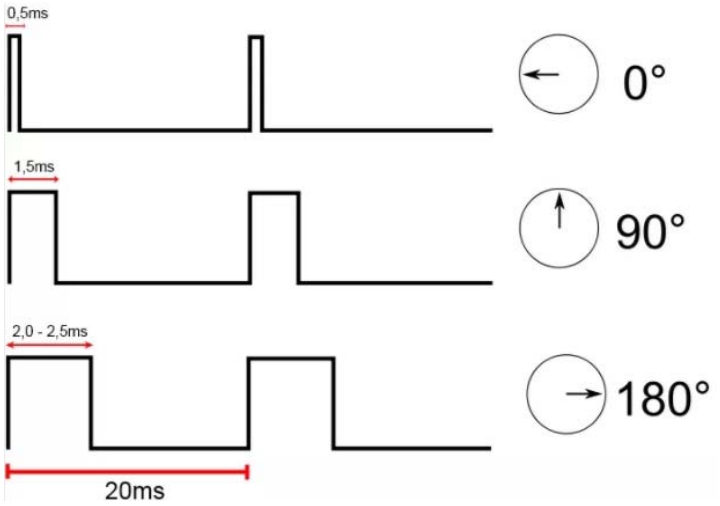


Figura 2.4.2.2.1.3.5.-. Control del Servo a través de PWM.

Para indicarle al servo que se coloque en el ángulo 0º, será preciso aportarle una señal con un ciclo de trabajo de 1,5 ms, 2 ms para 90° y 1ms para -90°. También pueden entenderse como ángulos de 90º, 0º y 180°.

2.4.2.2.2.- Motores Corriente Continua (CC)

2.4.2.2.2.1.- Función en el robot

El robot contará con dos motores de corriente continua, cuya función es la de hacer mover al robot, tanto en sentido de avance, como de retroceso o giro en ambos sentidos. Esto se consigue ya que ambos motores cuentan con una rueda que es la que facilita el desplazamiento.

2.4.2.2.2.2.- Principios de funcionamiento y constitución

Los motores de corriente continua, básicamente, convierten energía eléctrica en mecánica, por lo que reciben la denominación de componentes electromecánicos. La energía eléctrica proviene, evidentemente, de corriente eléctrica, que fluye por el motor ininterrumpidamente en un solo sentido a través de un circuito cerrado.

El funcionamiento de estos motores se basa en la ley de inducción electromagnética de Faraday, que enuncia que el voltaje inducido en un circuito cerrado es directamente proporcional a la rapidez con que cambia en el tiempo el flujo magnético que atraviesa una superficie cualquiera con el circuito como borde. Cuando un voltaje es generado por una batería, o por la fuerza magnética de acuerdo con la ley de Faraday, este voltaje generado, se llama tradicionalmente “fuerza electromotriz” (fem). La fem representa energía por unidad de carga.

Cualquier cambio del entorno magnético en que se encuentra una bobina de cable, originará un "voltaje" (una fem inducida en la bobina). No importa cómo se produzca el cambio, el voltaje será generado en la bobina. El cambio se puede producir por un cambio en la intensidad del campo magnético, el movimiento de un imán entrando y saliendo del interior de la bobina, moviendo la bobina hacia dentro o hacia fuera de un campo magnético, girando la bobina dentro de un campo magnético, etc. Desde un punto de vista práctico, si hacemos circular una corriente por un hilo que está entre los polos de un imán, aparecerá una fuerza mecánica que se opondrá a los cambios de esta corriente e intentara hacer girar el conductor para compensarlos. La fuerza que aparezca será proporcional a la intensidad del campo magnético, y a el número de conductores que sean recorridos por la corriente, (por eso lo que hacemos es arrollar espiras de cobre o devanados alrededor de un núcleo, para que la fuerza aumente). En otras palabras, el funcionamiento de estos motores se basa en la fuerza que se ejerce sobre un conductor eléctrico por el que circule una corriente y que se encuentre en el seno de un campo magnético. Esta fuerza provendrá por acción del

electromagnetismo.

En cuanto a su composición, cualquier motor DC está compuesto por dos partes principales, el estator y el rotor. El estator constituye la parte inmóvil del motor, siempre fija, ya que contiene los polos inductores del motor y es necesario que permanezcan inmóviles para el correcto funcionamiento de este. Estos polos son opuestos, es decir, presentan polaridad opuesta (N-S) y siempre se presentan en pares, constituyendo de esta manera, el campo magnético sobre el que se basa el funcionamiento del motor. Estos polos se sitúan en el interior de una carcasa de material ferromagnético.

Entre ambos polos se situará la otra parte principal del motor, el rotor. El rotor, construido por láminas de acero, se dispone sobre el eje central del motor y cuenta con pares de bobinado de cobre, situados alrededor de núcleo. Se disponen de tal forma que si solo tenemos un par de bobinas, se encuentran en posición directamente opuestas. El eje del motor le permite a este electroimán girar libremente entre los polos magnéticos norte y sur.

Cuando una corriente eléctrica circula por las bobinas del rotor, se produce un campo electromagnético, que interactuarán con el campo electromagnético generado en el estator, es decir, el campo magnético inductor.

En los motores DC siempre se busca la repulsión entre los campos magnéticos inductores e inducidos, es decir, siempre se busca que el polo sur del imán permanente interactúe con el polo sur del electroimán del rotor. De esta forma se repelerán, se producirá una fuerza magnética que provocará un torque y hará girar el eje sobre el que se dispone el rotor. Siguiendo este razonamiento siempre se llegará a una posición de equilibrio magnético, por lo que el giro se detendría y así no es como funciona un motor. Para continuar con el giro los motores DC cuentan con un conmutador o colector que, básicamente, lo que realiza es conmutar los campos magnéticos del rotor, es decir, que cada polo no realizará un giro de 360º, si no que a mitad de giro se invertirá la polaridad de cada bobina, produciéndose una nueva repulsión magnética, un par de fuerza de nuevo y, por tanto, se continuará el movimiento, el giro. Este giro permanecerá

activo mientras se mantenga la alimentación y esta sea suficiente para ello claro.

El conmutador no es más que un dispositivo cilíndrico de cobre que se encuentra dispuesto sobre el eje del rotor. El conmutador o colector, al estar constituido por cobre, es un elemento altamente conductor, ya que tienen que hacer circular la corriente eléctrica que circulará por las bobinas del rotor. Para invertir el sentido de giro del eje comentado anteriormente, el conmutador cuenta con zonas muertas, es decir, zonas no conductoras, habitualmente constituidas de mica. Las zonas conductoras del colector se denominan delgas y como las bobinas del rotor se encuentran en pares. Habrá tantos pares como pares de bobinas sobre el eje principal del motor.

La conexión eléctrica entre el motor y la fuente de alimentación se realiza a través de las escobillas, constituidas por grafito que, con una presión constante, aseguran el contacto permanente con las delgas del colector para inducir corriente a las bobinas. La corriente se inducirá a través de unos bornes, a las que se conectará la fuente de alimentación.

Cabe mencionar también que el espacio existente entre el rotor y el estator se denomina entrehierro, y debe de ser lo menor posible, ya que contiene aire y el aire presenta una elevada reluctancia magnética, por lo que se reduciría el efecto de los campos magnéticos y por tanto, la fuerza de repulsión magnética que genera el par del motor.

En la siguiente ilustración podemos observar gráficamente el funcionamiento de un motor DC. Como podemos observar el motor cuenta con los imanes norte y sur que constituyen el estator. Entre ellos se encuentra el rotor, que apoyado en el eje central girará en ambos sentidos dependiendo de cómo lo alimentemos. Alrededor del rotor observamos que se encuentra el conmutador, marcado en la imagen con una A y una B cada delga del mismo. Numerados se encuentran las escobillas, conectadas a la corriente eléctrica.

En la primera imagen puede observarse como se produce una fuerza electromotriz que hace girar el eje en sentido anti horario debido a la repulsión

electromagnética.

El motor gira hasta que haga 180º, momento que describe la segunda imagen de la figura 2.4.2.2.2.2.1. En este momento las escobillas del motor se encuentran con la zona no conductora del colector, por lo que cesará el paso de corriente y se pierde la polaridad electromagnética.

Por inercia el eje continúa girando y se produce la situación ilustrada en la tercera imagen de dicha figura. La delga que anteriormente estaba polarizada de una manera, ahora lo hace de manera inversa. El giro continúa y el motor trabaja correctamente.

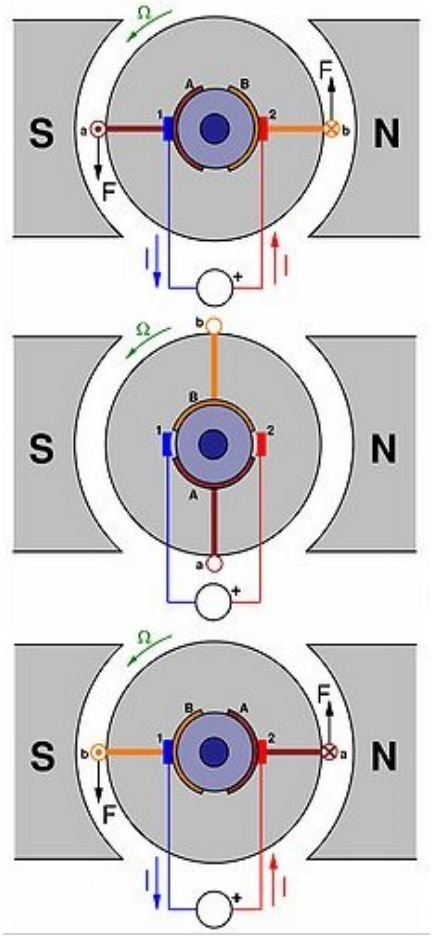


Figura 2.4.2.2.2.2.1.- Funcionamiento motores DC.

2.4.2.2.2.3.- Modelo seleccionado

El modelo de motores seleccionado ha sido uno de 5V genérico. Este motor presenta unas características muy ajustadas pero totalmente válidas para el cometido del robot. Con un voltaje de alimentación de entre 1.5 y 6V, trabaja a una corriente de 150 mA, con un torque máximo de 15g/m y una velocidad máxima de 12000 rpm. Su peso es de 16.5 g.

El modelo elegido cuenta además con una reductora 1:48. La caja reductora es un mecanismo que forma un grupo de engranajes, con el que se consigue mantener la velocidad de salida en un régimen cercano al ideal para el funcionamiento del generador.

Implementando el motor a este sistema, el conjunto presenta una velocidad sin carga de 100 RPM a 6V, así como una corriente de 180 mA.

En la siguiente imagen se podrá ver una imagen de ambos motores, así como de la rueda con la que cuentan.

Figura 2.4.2.2.2.3.1.- Motor DC con rueda.

2.4.2.3.- Fuente de alimentación

Cualquier circuito electrónico que interactúa con diferentes elementos como pueden ser sensores o actuadores, precisa de una alimentación de tensión para

poder generar la diferencia de potencial necesaria para que se origine el movimiento de los electrones, la corriente eléctrica.

2.4.2.3.1.- Función en el robot

La placa Arduino, como cualquier circuito electrónico, debe de ser alimentado, así como el resto de dispositivos controlados por el Arduino. El Arduino, como ya se ha visto en su correspondiente apartado, funciona con un voltaje operativo de 5V que se podrían obtener mediante la conexión USB con cualquier PC, pero se desea que el robot sea autónomo, sin necesidad de estar conectado al pc, por lo que se precisará una fuente de alimentación.

Arduino puede alimentarse tanto con 5V directamente o utilizando el regulador de tensión con el que cuenta. La solución adoptada es la de utilizar el regulador de tensión y una batería de polímero de litio (LIPO).

2.4.2.3.2.- Baterías de polímero de litio y modelo seleccionado

Una batería LIPO es un dispositivo diseñado para almacenar energía eléctrica. Las baterías LIPO trabajan bajo el mismo principio que rige el funcionamiento que las baterías de iones de litio, el intercambio de electrones entre el material del electrodo negativo y el material del electrodo positivo mediante un medio conductor. Para evitar que los electrodos se toquen directamente, se coloca entre ellos un material con poros microscópicos que permite tan sólo los iones (y no las partículas de los electrodos) migren de un electrodo a otro.

Las baterías de tipo LIPO son recargables, compuestas por células secundarias idénticas, colocadas en serie con el fin de proporcionar una mayor corriente de descarga. Cada celda suele tener entre 3,7 y 4.2V.

Las baterías LIPO se clasifican principalmente en función de 4 características generales, como son en primer lugar el número de celdas, que suele ser de 1 a 6 y que se indica con el número de celdas que contenga seguido

por una S, que se refiere a serie, ya que para aumentar el voltaje es preciso conectar celdas en serie. En segundo lugar se clasifican por la capacidad, que indica la cantidad de energía que puede contener la batería y que vendrá indicada como en cualquier otra batería en mAh. En tercer lugar se clasifican por el número de celdas en paralelo que presenten, para así aumentar la capacidad y que se indicará con el número seguido de una P. Por último se clasificarán dependiendo de la capacidad de descarga, la tasa de descarga, la rapidez con la que los iones fluyen del ánodo al cátodo, que es simplemente la rapidez con la que, de forma segura, una batería puede ser descargada.

La duración de una batería de este tipo, está definida por una fórmula:

$$Tiempo(\min) = \frac{Capac.(mAh)}{Velocidad\ descarga(mA)} \qquad (2..4.2.3.2.1)$$

Una batería LIPO de por ejemplo 1000 mAh y 20-30C de capacidad tarda una hora en descargarse si de ella se alimenta un dispositivo que consuma 1000 mA y puede dar una carga máxima sostenida de entre 20.000 y 30.000 mA.

El cálculo anterior evidentemente es teórico y en la práctica no será así, ya que a la batería en cada momento se le estará demandando una corriente diferente, el consumo no es constante. Además, la batería nunca debe descargarse por completo, ya que se dañaría. Además, hay que tener mucho cuidado también con su uso debido a los grandes niveles de energía que almacena, por lo que siempre hay que manipularlas con extrema precaución. La carga debe hacerse con un cargador adecuado y siempre supervisada.

El modelo elegido ha sido el 1.0 de la marca Turnigy, con unas especificaciones de capacidad de 1000 mAh, 20-30C de capacidad de descarga, que se traducen en 20.000 mA de descarga constante y 30.000 mA de pico y en cuanto a voltaje, 3 células de 3.7 V cada una, para sumar un total de 11.1V. En cuanto a peso y dimensiones, presenta un peso de 87 gramos y unas dimensiones de 77x33x20 mm.

Capacidad de descarga	20C-30C
Capacidad	1000 mAh
Número de celdas serie (Voltaje)	11.1V 3S (3.7 V/celda)
Número de celdas paralelo	1P

Tabla 2.4.2.3.2.1.- Características de la batería escogida.

Figura 2.4.2.3.2.1.-Batería seleccionada.

En cuanto a la recarga de las baterías, el fabricante recomienda hacerlo cuando se descarguen hasta obtener un voltaje de 10.8 V, nunca más. Para ello se dispondrá de un voltímetro de reducidas dimensiones. La recarga se realizará con un cargador debidamente diseñado para cargar este tipo de baterías, concretamente el modelo B3AC 2S 3S 7.4V 11.1V Power Lipo RC de la marca Imax.

Figura 2.4.2.3.2.2.- Cargador de baterías.

Este modelo de cargador cuenta con un equilibrado de las celdas de batería, por lo que no es algo con lo que haya que preocuparse. El equilibrado se refiere a que todas las celdas se carguen a niveles similares. Cuando se carga una batería LIPO siempre se va a cargar antes una celda que otra, ya que en el uso siempre se desequilibran. Cuando esto ocurre y una celda ya está cargada, la cantidad extra de energía la desprende en forma de calor o se descarga alguna celda si se superan ciertos niveles. Esto es lo que se denomina balanceo.

2.4.2.4.- Circuitos electrónicos

2.4.2.4.1.- Controlador de motores

2.4.2.4.1.1.- Función en el robot

El controlador de motores tiene la función de controlar el movimiento de los motores que permitirán desplazarse al robot, controlando tanto el avance como el retroceso del robot, el giro en sentido horario o anti horario y el frenado de cada uno de los motores.

2.4.2.4.1.2.- Modelo seleccionado

El controlador seleccionado ha sido el LN298N, un módulo integrado que permitirá controlar dos motores de corriente continua.

2.4.2.4.1.3.- Principios de funcionamiento

El modelo seleccionado se basa en el concepto de puente en H. El puente en H es un circuito electrónico cuya finalidad es la de convertidor en puente continua-alterna. El funcionamiento de este tipo de circuitos es sencillo, a partir de una entrada de continua se obtendrá una salida de alterna cerrando y abriendo interruptores en una determinada secuencia.

En la figura que se expone a continuación puede observarse un esquema

básico de un puente en H. Su nombre, como podemos ver en la imagen, proviene de su característica forma.

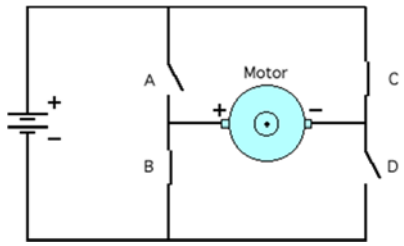


Figura 2.4.2.4.1.2.1.- Puente en H

Como podemos observar en la figura, la corriente está excitando al motor en el sentido que va desde el interruptor C al B. En caso contrario, estarían cerrados los interruptores A y D y el motor estaría excitado en el otro sentido, siempre de dos en dos. Como la carga se trata de un motor CC, con una secuencia u otra el motor girará en un sentido u otro. No existe otra combinación posible, ya que se produciría un cortocircuito.

El chip LN298N cuenta con 2 puentes en H, permitiendo controlar 2 motores CC (un puente para cada motor) por medio de 4 salidas digitales de arduino (2 para cada motor). Para hacer girar cada motor en un determinado sentido, se ha de establecer una patilla a HIGH y la otra a LOW. Para hacerlo girar en sentido contrario han de establecerse al revés.

2.4.2.4.1.4.- Características

Las principales características técnicas de este integrado son las siguientes:

Alimentación	6V - 12V DC / 12v-35v DC (jumper 5V)
Rango alimentación	4.8V - 46V
Corriente máxima	2ª
Nivel alto	Alimentacion-2.3V

Nivel bajo	1.5V-0.3V
Dimensiones	43x23.9x43 mm
Peso	29 g
Máxima disipación de potencia	25w

Tabla 2.4.2.4.1.2.1.- Características LN298N

Al igual que el Arduino, este integrado trabaja con tecnología TTL.

A continuación puede observarse un esquema de este circuito:

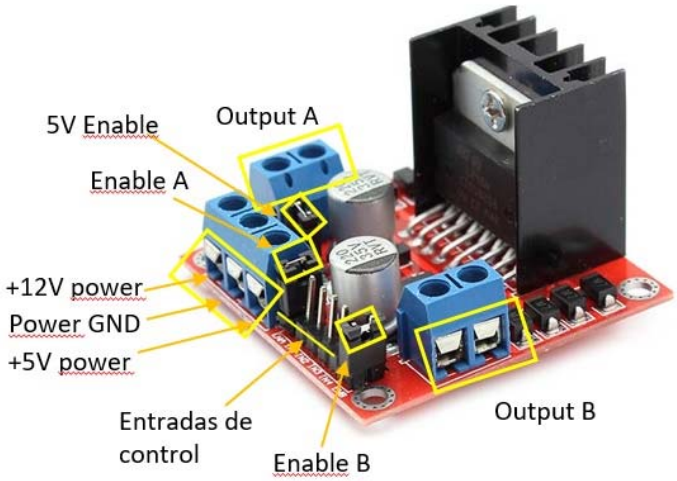


Figura 2.4.2.4.1.2.2.- Esquema LN298N.

Este módulo cuenta con diferentes elementos adicionales necesarios para funcionar correctamente. Además de un integrado L298N, encargado de ejercer el control, cuenta con un regulador lineal LM7805, similar al que tiene el Arduino, que permitirá alimentar a la parte lógica con 5V indiferentemente de la tensión con la que alimentemos los motores por la patilla de +12V y con diodos de protección, que protegerán el sistema de inversiones de polaridad o sobreintensidades.

La alimentación del integrado se puede realizar de dos maneras diferentes dependiendo de los niveles de tensión a los que se quieran alimentar los motores.

Para un rango de 6V a 12V, que será el utilizado en este proyecto, el jumper de selección de 5V deberá estar activo, activándose el regulador de voltaje. Para un rango de voltajes superior, de 12 a 35V, el jumper anterior debería estar deshabilitado, de forma que el regulador no trabajaría y debería de ser alimentado con 5V externamente, tensión que aporta el propio Arduino.

Como ya se comentó, este integrado cuenta con dos salidas (Output A y Ouput B) para controlar cada una de ellas un motor, de forma que con un nivel alto en una patilla de la salida A y un nivel bajo en la otra, se hará que el motor gire en un sentido, y al revés se hará que gire en el otro.

El LN298N permite habilitar o deshabilitar las salidas por software, mediante las entradas Enable A y Enable B, así como por hardware, ya que dispone de dos jumpers de selección. Al colocar el jumper las salidas estarían habilitadas.

Para el control de los motores, es decir, el control del puente en H que controlará el sentido de giro de los motores, se utilizarán las entradas de control. Estas entradas se conectarán al Arduino y mediante programa se controlarán los motores. En la siguiente tabla se recogen las diferentes posibilidades existentes para las entradas de control de ambos motores y el resultado, tanto el sentido de giro de cada motor y su respectiva rueda, como el resultado obtenido en el robot.

IN 1	IN 2	IN 3	IN 4	Salida motor 1	Salida motor 2	Acción robot
1	0	1	0	Horario	Horario	Avance
0	1	0	1	Antihorario	Antihorario	Retroceso
1	0	0	1	Horario	Antihorario	Giro derecha
0	1	1	0	Antihorario	Horario	Giro izquierda
0	0	0	0	Frena	Frena	Frenado por inercia
1	0	1	0	Frena	Frena	Frenado rápido

0	1	0	1	Frena	Frena	Frenado rápido

Tabla 2.4.2.4.1.2.2.-Control del L298N

El control de la velocidad de los motores se llevará a cabo a través de las patillas PWM del Arduino, que conectadas a los enables de cada motor, controlarán la velocidad de giro (rango 0-255).

2.4.2.4.2.- Convertidor CC-CC reductor de tensión

2.4.2.4.2.1.- Función en el robot

La función del convertidor DC-DC reductor de tensión en el robot es la de adaptar el voltaje proveniente de la fuente de alimentación a valores aceptables para alimentar el controlador de motores.

2.4.2.4.2.2.- Principios de funcionamiento

Este circuito electrónico tiene la función de proporcionar un voltaje regulado a partir de una fuente de tensión, de forma que el voltaje de salida será menor que el de entrada, todo ello con un rendimiento cercano al 90%.

Para lograr su cometido utilizan la denominada técnica de conmutación, de donde reciben el nombre de convertidor conmutado. Esta técnica consiste en trocear la señal de entrada del regulador mediante el uso de un interruptor o conmutador, que suele ser un transistor, que se cerrará y abrirá miles de veces por minuto. La onda resultante será cuadrada, por lo que es preciso que pase por un filtro LC, con condensador e inductancia, que se situará a continuación de un diodo que cerrará el circuito cuando el conmutador esté abierto.

Dicho filtro, junto al diodo, consiguen conformar un circuito capaz de proporcionar la tensión deseada a la salida. Cuando el transistor está en corte, le circuito se alimenta de la tensión exterior, cargándose el campo magnético de la

inductancia y el condensador. Una vez está cargado el condensador, la corriente circula por la carga.

Cuando el transistor corta la tensión, la inducción comienza a liberar la energía almacenada, al igual que el condensador. Ahora la corriente circulará por el diodo, que cerrará el circuito. De esta forma se mantiene el valor de la salida a un valor constante, muy estable y se obtiene un rendimiento notable, ya que cuando el transistor corta la corriente de entrada no se produce consumo de energía.

La tensión de salida se regula en función de la anchura de los pulsos que conmutan el transistor.

Este tipo de circuitos también reciben el nombre de reguladores Buck o Step Down.

2.4.2.4.2.3.- Modelo seleccionado y características

El modelo seleccionado para dicho cometido será el regulador LM2596S, un circuito monolítico que dispone de una referencia interna de 150Khz para la base de conmutación del transistor interno y que contiene la base para construir una fuente conmutada. Esto no será necesario ya que el mercado ofrece por un precio muy reducido módulos ya montados con todo lo necesario, como el módulo LM2596S DC-DC Step Down, que será el elegido en este proyecto.

Para el uso de este módulo, simplemente hay que conectar un voltaje de entrada y regular el potenciómetro con el que cuenta hasta que se obtenga a la salida el voltaje deseado, siempre teniendo presente las limitaciones y características que presenta el circuito, que vendrán recogidas a continuación en la tabla de características que se expone:

Rango voltaje entrada	4.5V-40V

Rango voltaje de salida	3.3V-37V
Corriente salida maxima	3 A
Corriente de pico admitida	4.5V (máximo de 6,9 V, no recomendable)
Frecuencia conmutación maxima	173 kHz
Frecuencia conmutación mínima	110 kHz
Frecuencia conmutación típica	150 kHz
Temperatura de trabajo	-40-125ºC
Ciclo de trabajo	0 -100 %

Tabla 2.4.2.4.2.3.1.- Características LM2596S DC-DC Step Down.

Este circuito además cuenta con un condensador a la entrada que permite hacer un filtrado previo a la conmutación.

El datasheet de este integrado cuenta con una serie de ejemplos de ensayos en los que se estudiaron diferentes casos. En ellos se recogen los parámetros necesarios para, a unas condiciones atmosféricas, obtener ciertos valores de tensión de salida, concretamente valores de salida de 3.3V, 5V Y 12V. A continuación podrán verse los mencionados casos:

Rango tensiones de entrada	4.75 V - 40 V
Rango corriente de carga	0,2 A – 3 A

Tabla 2.4.2.4.2.3.2.- Alimentación y carga 3,3 V.

Rango tensiones de entrada	7 V - 40 V
Rango corriente de carga	0,2 A – 3 A

Tabla 2.4.2.4.2.3.3.- Alimentación y carga 5 V.

Rango tensiones de entrada	15 V - 40 V
Rango corriente de carga	0,2 A – 3 A

Tabla 2.4.2.4.2.3.4.- Alimentación y carga 12 V.

	MIN	TYP	MAX
Tª = 25 °C	3.168 V	3.3 V	3.432 V
-40 °C < Tª < 125 °C	3.135 V		3.465 V

Tabla 2.4.2.4.2.3.5.- Tensiones de salida 3,3 V.

	MIN	TYP	MAX
Tª = 25 °C	4,8 V	5 V	5,2 V
-40 °C < Tª < 125 °C	4,75 V		5,25 V

Tabla 2.4.2.4.2.3.6.- Tensiones de salida 5 V.

	MIN	TYP	MAX
Tª = 25 °C	11,52 V	12V	12,48 V
-40 °C < Tª < 125 °C	11,4 V		12,6 V

Tabla 2.4.2.4.2.3.7.- Tensiones de salida 12 V.

En la siguiente tabla podrán observarse los datos referidos a la eficiencia para cada uno de los casos de tensión de salidas anteriores, para una entrada de 25 V y una corriente de carga de 3 A:

3.3 V	73%
5 V	80%
12 V	90%

Tabla 2.4.2.4.2.3.- Eficiencia según salida.

2.4.2.4.3.- Voltímetro digital

2.4.2.4.3.1.- Función en el robot

El robot contará con un voltímetro digital que le permita al usuario conocer tanto el voltaje de la batería, como el voltaje de salida del convertidor DC-DC para ajustarlo si es preciso.

2.4.2.4.3.2.- Principio de funcionamiento

Un voltímetro digital es un dispositivo que permite realizar la medición de la diferencia de potencial o tensión que existe entre dos puntos pertenecientes a un circuito eléctrico y representar ese valor en una pantalla.

2.4.2.4.3.3.- Modelo seleccionado y características

El modelo seleccionado no está catalogado bajo ningún nombre de modelo ni ninguna marca. En la siguiente figura puede verse una imagen del modelo elegido:

Figura 2.4.2.4.3.3.1.- Voltímetro digital.

Las características de este voltímetro pueden observarse en la siguiente tabla:

Rango medida	3,5V – 30 V
Corriente funcionamiento	30 mA
Tiempo refresco medida	500 ms
Representación voltaje	3 dígitos
Dimensiones	54x24x15 mm
Temperatura funcionamiento	-10º a 60º C

Tabla 2.4.2.4.3.3.1.- Características voltímetro digital.

2.4.2.4.4.- Pantalla LCD

El Liquid Crystal Display (LCD) es una pantalla de cristal líquido que se utiliza para mostrar información visual. La pantalla es delgada y plana y está formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora.

2.4.2.4.4.1.- Función en el robot

El LCD permitirá al usuario ver el camino recorrido por el robot y el camino más corto que podría realizar.

2.4.2.4.4.2.- Principio de funcionamiento

Los monitores LCD están hechos con dos láminas de vidrio herméticamente selladas, dentro de las cuales se introduce cristal líquido, una sustancia líquida, pero con propiedades cristalinas, ya que sus moléculas se agrupan en formaciones paralelas, formándose una capa de moléculas alineadas, que se sitúan entre dos electrodos, es decir, las láminas de vidrio y entre dos filtros de polarización.

Dado que las propiedades ópticas de la capa de moléculas varían en función de la orientación, y ésta depende del campo eléctrico o magnético que se genere entre dichas capas conductoras, entonces es posible producir todo tipo de efectos en la pantalla, controlándolos en cada punto. Los números o letras que aparecerán en la pantalla cuando se utilice se forman al aplicar cargas de bajo voltaje al líquido en puntos específicos. Este tipo de monitores también son conocidos como de matriz pasiva.

Cada pixel en una pantalla de matriz pasiva está controlado por transistores colocados alrededor de la pantalla, en el borde. Estos transistores controlan todos los pixeles en una columna o fila dada. En otras palabras, muchos pixeles obtienen sus instrucciones de uno o dos transistores. Los transistores son a su vez controlados por un procesador.

2.4.2.4.4.2.- Modelo seleccionado

El modelo seleccionado es un Display LCD 16x2 HD44780. Este display cuenta con una pantalla con fondo retroiluminado de 16 caracteres y dos filas (16x2). Viene equipado con un microcontrolador HD44780 de la compañía Hitachi que se encarga de generar los caracteres, polarizar la pantalla, desplazar el cursor….Además también viene equipado con una memoria ROM donde se encuentran almacenados los caracteres a través de una matriz de puntos, y de una memoria RAM donde se pueden almacenar caracteres creados por el usuario.

Para controlar el display este cuenta con una serie de pines. Normalmente cuentan con 16 pines, al igual que en el caso de este modelo. A continuación podrá observarse el nombre y la función de cada pin:

Pin	Símbolo	Función
1	VSS	GND

2	VDD	5V
3	V0	Voltaje operación LCD
4	RS	High: Dato Low: Instrucción
5	R/W	R: Leer W: Escribir
6	E	Señal validación (Enable)
7	DB0	Dato 0
8	DB1	Dato 1
9	DB2	Dato 2
10	DB3	Dato 3
11	DB4	Dato 4
12	DB5	Dato 5
13	DB6	Dato 6
14	DB7	Dato 7
15	BLA	5V Alimentación retroiluminación
16	BLK	0V Desactiva retroiluminación

Tabla 2.4.2.4.4.2.1.- Pines LCD.

Además de ofrecer los pines anteriores para controlar el LCD, la placa también ofrece la posibilidad de controlarlo mediante comunicación I2C.

2.4.3.- ALTERNATIVAS ESTUDIADAS

2.4.3.1.- Unidad de medición inercial

2.4.3.1.1.- Función en el robot

La función que tendría la IMU en el robot sería la de estabilizar el robot para que el avance del mismo a lo largo de las calles del laberinto sea lo más recto y lo más centrado posible, así como controlar la posición del robot antes y después de un giro.

2.4.3.1.2.- Principios de funcionamiento

Una Unidad de medición inercial es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, es decir, permite conocer la posición exacta del dispositivo sobre cada uno de sus ejes o su variación, pudiendo determinar la posición exacta del dispositivo en cada momento teniendo en cuenta como el eje de coordenadas como el punto medio del sensor. Es un componente básico en los sistemas de navegación inercial que utilizan tanto aviones como barcos, entre otros muchos.

El funcionamiento de una IMU se basa en la presencia de unos sensores llamados acelerómetros y giroscopios.

Los acelerómetros miden la aceleración lineal en uno de los tres ejes espaciales. Esto lo consiguen detectando fuerzas inerciales, lo que antiguamente se realizaba de forma mecánica. Dentro de un armazón se colocaba una masa suspendida en un mecanismo elástico, que basándose en la ley de Hooke y en la segunda ley de Newton, permitía medir la aceleración del sistema, ya que esta sería proporcional al desplazamiento de la masa.

Hoy en día los acelerómetros no utilizan esta tecnología, si no que se basan en otros fenómenos físicos más acordes a la tecnología actual, en forma de dispositivos electrónicos integrados, como es el caso de los acelerómetros

capacitivos y los piezoeléctricos, que detectan la aceleración que una fuerza externa provoca en el sensor y mezclando la información de los tres ejes determina mediante calculo, la dirección de la fuerza que se le aplica, es decir, detectan las fuerzas de inercia a las que se les somete y las descomponen en los 3 ejes dimensionales.

Una de las 3 fuerzas de inercia siempre será la de la gravedad, que se sabe que siempre es perpendicular a la superficie terrestre. Gracias a esta referencia es posible conocer los ángulos de inclinación del sensor por trigonometría, excepto del eje Z.

Para calcular la inclinación el los ejes x e y se utiliza la fórmula del arcotangente:

$$\text{Angulo Y} = \text{atan}\left(\frac{X}{\sqrt{Y^2 + Z^2}}\right) \qquad (2.4.3.1.2.1)$$

$$\text{Angulo Z} = \text{atan}\left(\frac{Y}{\sqrt{X^2 + Z^2}}\right) \qquad (2.4.3.1.2.2)$$

Por otro lado, los giroscopios detectan las fuerzas centrífugas a las que se someten los tres ejes dimensionales principales del sensor, en otras palabras, la rotación del sensor en cada uno de sus ejes en forma de velocidad angular.

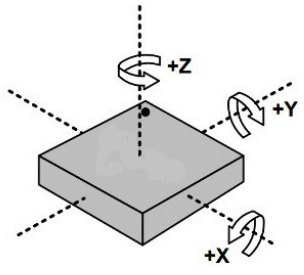


Figura 2.4.3.1.2.1.- Fuerzas centrífugas en los 3 ejes.

Con el giroscopio se obtiene velocidad angular, pero estas medidas

individualmente, al igual que las del acelerómetro, apenas sirven de nada. Si con el acelerómetro se obtiene el ángulo de inclinación, y se toma esta medida en un momento de tiempo llamado, por ejemplo, tiempo cero, para conocer lo que se desplaza el ángulo en un tiempo determinado, llamado tiempo de giro, es preciso echar mano de la medida del giroscopio. De esta manera podrá obtenerse el ángulo al que se encuentra cualquiera de los ejes con respecto a la referencia, tal y como se recoge en la siguiente ecuación:

$$Ángulo_{eje} = angulo_{eje_{tiempocero}} + medida_{giroscopio} + tiempo_{giro} \qquad (2.4.3.1.2.3)$$

Una IMU convencional, como la explicada hasta ahora, solo podría determinar la inclinación con respecto a los ejes X e Y, pero no al eje Z, para lo cual sería preciso un magnetómetro que funcione como brújula, un dispositivo capaz de medir la fuerza y/o dirección de los campos magnéticos que los afectan respecto el campo magnético terrestre.

Los sensores de que se componen las unidades de medida inercial se definen principalmente por su rango de trabajo, su sensibilidad y su frecuencia de funcionamiento. Para el caso de los acelerómetros, el rango de trabajo se mide con la gravedad estándar, aceleración de g = 9.80665 m/s2. Pueden encontrarse IMUs con un rango de 1.5g, 2g,4g, etc. Dependiendo de este rango se obtiene una sensibilidad u otra, ya que el nivel de voltaje máximo que se puede obtener viene definido por la alimentación del sensor, es decir, para mayor rango, mayor sensibilidad.

En los giroscopios el rango de medida se mide en º/s y la sensibilidad en mV/º/s. Los valores típicos del rango del giroscopio suelen ser ±200 º/s, ±300 º/s o ±500 º/s.

En caso de que la IMU tenga magnetómetro, el rango se mide en gauss.

La frecuencia de trabajo de la IMU vendrá marcada por la velocidad del Arduino y por la frecuencia de conversión del AD de la propia IMU, además de la

velocidad de transmisión de datos del protocolo que use para comunicarse.

2.4.3.1.3.- Modelo seleccionado

El modelo elegido para las pruebas es el MPU6050, una IMU de 6 grados de libertad, es decir, que cuenta con un acelerómetro y un giroscopio, ambos de 3 ejes. Además cuenta con conversores analógicos digitales de 16 bits y se comunica con el Arduino por I2C.

El voltaje de alimentación recomendado es de 3,3 V, aunque admite alimentaciones de 5V.

Rango voltaje entrada	2,3-3,3 V
Comunicación	I2C
Corriente de trabajo giroscopio	5mA
Corriente de trabajo acelerómetro	500uA
Rango escala giroscopio	±250, ±500, ±1000 y ±1000 º/s
Rango escala acelerómetro	±2g, ±4g, ±8g y ±16g
Algoritmos internos de calibración del dispositivo	
Procesador a bordo Digital Motion Processing™ (DMP™)	
Incluye un sensor de temperatura digital con una oscilación de +1%	
AD giroscopio	16 bits
AD acelerómetro	16 bits

Tabla 2.4.3.1.3.1.- Características MPU6050.

Figura 2.4.3.1.3.1.- MPU 6050

En la foto puede observarse una MPU 6050 con todas sus pines. Para el control de este módulo solamente es preciso utilizar los 4 pines de arriba. Existen numerosas librerías y ejemplos en internet.

2.4.3.2.- Magnetómetro

2.4.3.2.1.- Función en el robot

Un magnetómetro permitiría al robot conocer su desviación con respecto al norte magnético. Esto le permitiría, acompañado del correspondiente código, regular su desviación y poder avanzar así lo más recto posible, evitando las paredes laterales.

2.4.3.2.2.- Principios de funcionamiento

Un magnetómetro es un instrumento que mide los disturbios magnéticos provocados por el campo magnético de la tierra en términos de densidad de flujo magnético (Teslas). Si estos disturbios en un lugar determinado tienen un valor fijo, si ciertas sustancias o materiales los modifica (magnéticos), entonces un magnetómetro detectará la cantidad de distorsión y la fuerza de los campos magnéticos y utilizará esta información para discernir la dirección, rotación y el ángulo de los campos magnéticos, así como la ubicación de objetos específicos dentro de ellos.

Un magnetómetro mide las componentes del campo magnético presente. De esta manera permite calcular la orientación con respecto al norte magnético de

la tierra.

2.4.3.2.3.- Modelo seleccionado

El modelo escogido es el magnetómetro de 3 ejes HMC5883L.

Figura 2.4.3.2.3.1.- HMC5883L.

En la siguiente tabla se muestran las principales características de este módulo:

Rango voltaje entrada	2,3-3,3 V (admite 5V)
Comunicación	I2C
Bajo consumo de corriente	
Rango de medición	[±0,88Ga, ±8,1Ga]
Ganancias al rango de medición	[1370,230]
AD	12 bits

Tabla 2.4.3.2.3.1.- Características HMC5883L.

2.4.3.3.- Convertidor DC-DC elevador de tensión

2.4.3.3.1.- Principios de funcionamiento

Un convertidor DC-DC elevador de tensión también recibe el nombre de boost o de step-up y su función es la de proporcionar a su salida una tensión continua regulada superior a la de la entrada.

Este convertidor, al igual que el Buck, utiliza la técnica de conmutación para alcanzar su objetivo. La entrada de tensión cuenta con una bobina en serie, un interruptor, un diodo y un condensador, como puede apreciarse en la siguiente figura:

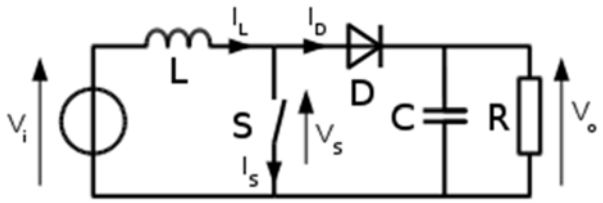


Figura 2.4.3.2.3.2.- Elevador de tensión.

El interruptor S abrirá o cerrara el circuito de carga de la bobina, alternando periodos de carga con periodos de descarga. Cuando se abra el interruptor, la corriente almacenada en la bobina se descargará a través del diodo y cargará a su vez al condensador a una tensión mayor a la tensión de entrada.

El interruptor, al igual que en el caso del convertidor buck, es un transistor, que se controlará normalmente por PWM, abriéndose y cerrándose miles de veces por segundo. El diodo se coloca para que no se cortocircuiten el condensador cargado y masa cuando el interruptor esté cerrado.

El ciclo de trabajo determina la capacidad de conversión de tensión del convertidor. A mayor ciclo de trabajo mayor será la tensión de salida con respecto a la tensión de entrada.

2.4.3.3.2.- Modelo seleccionado

El modelo seleccionado es el módulo XL6009 cuyo funcionamiento es similar al explicado para el módulo convertidor buck elegido para este proyecto. Por medio de un potenciómetro se regula el voltaje de salida sin necesidad de controlar el transistor por medio de PWM.

Figura 2.4.3.3.2.1.-Elevador XL6009

A continuación se exponen las principales características de este módulo:

Rango voltaje entrada	3V-32V
Rango voltaje de salida	5V-39V
Corriente salida maxima	3 A
Corriente entrada máxima	4 A (con disipador de calor)
Frecuencia conmutación de trabajo	400 kHz
Temperatura de trabajo	-40-85ºC
Ciclo de trabajo	0 -100 %
Eficiencia	94%

Tabla 2.4.3.3.2.1.- Características XL6009.

En el datasheet del módulo pueden encontrarse diferentes ejemplos de rendimiento:

Entrada	Salida	Corriente	Potencia
3 V	12 V	0,4 A	4,8 W
5 V	12 V	0,8 A	9,6 W
7,4 V	12 V	1,5 A	18 W

12 V	15 V	2 A	30 W

Tabla 2.4.3.3.2.2.- Ejemplos de rendimiento XL6009.

2.4.3.4.- Fuentes de alimentación

Cuando en un proyecto como este, se utilizan motores, surge la necesidad de utilizar una alimentación externa, no solo por la búsqueda de autonomía, como se recoge en los requisitos de diseño, sino que por lo general, los motores realizan un elevado consumo y con los 40 mA y 5V que ofrece Arduino es insuficiente. Como bien se comentó anteriormente, existen diversas formas de alimentar circuitos electrónicos. A continuación se presentarán las principales opciones que se plantearon a la hora de elegir un método de alimentación en concreto:

- Una pila de 9V. Voltaje adecuado para alimentar el circuito, pero cuenta con el hándicap de no ser recargable y de contar con una carga escasa para nuestra aplicación. Además su capacidad es pequeña y apenas puede llegar a aportar una intensidad de corriente máxima de 300mA.

- Cuatro pilas AA en serie. Proporcionan un total de 6V. Su capacidad está en niveles superiores a una pila de 9V (800-1500mAh), y más si hablamos de pilas alcalinas (1700-2800 mAh), capaces de aportar una intensidad de 1A. Aunque mejora notablemente la capacidad de la opción anterior, tampoco son recargables, por lo que tampoco son la opción óptima. Además 6V se hacen cortos para la alimentación de los dos motores CC con los que contará el robot, ya que aunque con 6V funcionen perfectamente, las pilas nunca aseguran un valor exacto, y con menos de 6V los motores trabajarían a unas rpm demasiado bajas, por no hablar de que las pilas convencionales pronto sufren caídas de tensión y ni por asomos aportan una tensión constante de 1.5V por unidad.

- Cinco pilas recargables de 1.2V. Las más conocidas y las que pueden encontrarse en cualquier tienda son las de NiMh, que cuentan con una aleación capaz de absorber hidrógeno como electrodo negativo. Estas baterías tienen una capacidad aceptable de entre 600 y 2500 mAh, proporcionando hasta 1 A. Esta opción cuenta con la ventaja respecto a las anteriores de que con el cargador adecuado podrían recargarse las baterías.

- Dos baterías de litio 18650 de 3.7V. Proporcionan una alta capacidad de carga, de hasta 4800 mAh con una capacidad de descarga 1C – 2C, lo que supone una intensidad máxima de 10 A. Este tipo de baterías son también recargables y el precio es bastante superior a las opciones anteriores.

- Banco de alimentación USB. Típicas baterías portátiles de los teléfonos móviles basadas en una batería de litio 18650. Como ventaja cuenta con que son recargables, que existe un gran abanico de diferentes capacidades, hasta niveles bastante altos y que proporciona unos valores de intensidades más que suficientes para un proyecto como este.

- Baterías de niquel (NIMH). Similar a las pilas recargables 1,2V de NiMh comentadas anteriormente, pero en formato de batería y con conectores incorporados. Generalmente se presentan en formato de 5 o 8 celdas con valores de tensión de 6V y 9.6V respectivamente. Cuentan con una capacidad de carga elevada (300 -5000mAh) .además son muy densas, es decir, presentan una alta densidad de carga y son capaces de proporcionar altos niveles de corriente (entre 3 y 4C).

2.5.- NORMAS Y REFERENCIAS

2.5.1.- Disposiciones legales y Normas aplicadas

En la redacción de este Proyecto se han tenido en cuenta todas y cada una de las especificaciones y normas contenidas en las Reglamentaciones y Normas de las Escuela Universitaria Politécnica de Serantes para la realización del trabajo de fin de grado.

2.5.2.- Bibliografía

[1] Goilav, N y Loi, G. (2016). *Arduino. Aprender a desarrollar para crear objetos inteligentes*. Barcelona: Eni ediciones.

[2] Lopez Aldea, E. (2016). *Arduino. Guía práctica de fundamentos y simulación*. Madrid: Ra-ma.

[3] Lozano Equisoain, D. (2017). *Arduino práctico*. Madrid: Anaya.

[4] Reyes Cortés, F. (2011). *Robótica Control de robots manipuladores*. Alfaomega

[5] Reyes Cortés, F. y Cid Monjaraz, J. (2015). *Arduino. Aplicaciones en robótica, mecatrónica e ingenierías*, Alfaomega.

[6] Torrente Artero, O. (2013). *Arduino. Curso práctico de formación*. Madrid: RC Libros.

2.5.3.- Otra referencias

En este apartado se incluirán aquellas referencias que se consideren de interés para la materialización del TFG.

[1] https://metacrilato.makinolo.com/web5.html

[2] http://www.matematicasdigitales.com/como-escapo-de-este-laberinto/

[3] http://es.wikihow.com/hacer-soldaduras-de-esta%C3%B1o-correctamente

2.5.4.- Programas Informáticos utilizados para elaborar el proyecto

En la redacción de este Proyecto se han utilizado las herramientas informáticas y programas de cálculo que se indican a continuación:

- MICROSOFT WORD 2013 como tratamiento de textos.
- MICROSOFT EXCEL 2013 para confección de tablas y cálculos.
- AUTOCAD 2014 para el desarrollo gráfico.
- ARUDUINO 1.8.4 para la programación.

2.6.- DEFINICIONES Y ABREVIATURAS

En este apartado se adjuntarán todas las definiciones necesarias para el correcto entendimiento del proyecto, así como todas aquellas abreviaturas utilizadas con sus respectivas definiciones, lo que se realiza para simplificar la lectura y homogeneizar el lenguaje técnico del proyecto. La primera vez que se utilice una abreviatura, se hará entre paréntesis siguiendo a la palabra que, en lo sucesivo, va a sustituir.

- **Arquitectura Hardvare.** Arquitectura Hardware de los microcontroladores basada en la utilización de dos memorias diferentes, a través de dos buses independientes, por parte de la unidad central de procesamiento (CPU). Una memoria almacena las instrucciones de programa mientras que la otra para almacenar datos.

- **Arquitectura Von Neumann.** Arquitectura Hardware de los microcontroladores basada en la conexión directa mediante un único bus entre la CPU y una memoria única, que contendrá tanto instrucciones de programa como datos.

- **Asociación de la Industria Eléctrica (AIE).**

- **Baterías de niquel (NIMH).**

- **CC o DC (Corriente continua o Direct Current).** Corriente de intensidad constante en la que el movimiento de las cargas siempre es en el mismo sentido.

- **Ciclo de trabajo.** En electrónica, el ciclo de trabajo, ciclo útil o régimen de trabajo es la relación que existe entre el tiempo en que la señal se encuentra en estado activo y el periodo de la misma.

- **Ciclos por revolución (CPR).** Capacidad de un encoder para dividir una rotación (360º) entre n.

- **Conversor Analógico-digital (AD).**

- **Convertidor CC-CC.** Los convertidores CC-CC son circuitos electrónicos de potencia que convierten una tensión continua en otro nivel de tensión continua y, normalmente, proporcionan una salida regulada.

- **Digital Motion Processing™ (DMP™).**

- **Electrically Erasable Programmable Read-Only Memory (EEPROM).**

- **Enable.** Patilla de habilitación.

- **Encoder.** Un encoder, también conocido como codificador o decodificador, es un dispositivo electromecánico de detección que proporciona una respuesta. Los Encoders convierten el movimiento en una señal eléctrica que puede ser leída por algún tipo de dispositivo de control en un sistema de control de movimiento. El encoder envía una señal de respuesta que puede ser utilizado para determinar la posición, contar, velocidad o dirección, convirtiendo la posición angular de un eje, directamente a un código digital. Un dispositivo de

control puede usar esta información para enviar un comando para una función particular.

- **Entorno de desarrollo Integrado (IDE).**

- **Espectro electromagnético.** Se denomina espectro electromagnético al rango de todas las radiaciones electromagnéricas posibles.

- **Exactitud.** Diferencia entre la medición realizada con respecto a la medición real.

- **Fuerza electromotriz (fem).**

- **HIGH.** En la electrónica digital se define como el nivel lógico alto. Un 1.

- **Interrupt Service Handler o Servicio de gestión de interrupción (ISH).**

- **LED.** Diodo emisor de luz.

- **LIPO (Polímero de Litio):** tipo de batería recargable compuestas por polímero de litio.

- **Liquid Crystal Display (LCD).** Los circuitos integrados que trabajan con esta tecnología destacan por presentar una gran inmunidad al ruido, baja impedancia de salida y una gran flexibilidad lógica.

- **LOW.** Nivel lógico bajo. Un 0.

- **Luz infrarroja (IR).**

- **Master In Slave Out (MISO)-**

- **Master Out Slave In (MOSI)**

- **Microcontrolador**. Circuito integrado programable capaz de ejecutar órdenes grabadas en su memoria.

- **PCM (Pulse Code Modulation).**

- **Precisión.** Número de bits para representar una medición real o ideal.

- **PWM (Pulse Width Modulation):** modulación por ancho de pulso. Técnica que consiste en la modificación del ciclo de trabajo, es decir, el tanto por ciento por periodo que una señal está a nivel alto, de una señal periódica cuadrada, bien sea para el envío de información o para realizar el control de la cantidad de energía enviada a una carga.

- **Random Access Memory (RAM).** Memoria de acceso aleatorio.

- **Read Only Memory (ROM).** Memoria de solo lectura.

- **Reduced Instruction Set Computer (RISC).** Computadoras con un conjunto de instrucciones reducido.

- **Regulador lineal.** Regulador de tensión que o bien basa su funcionamiento en un elemento activo (transistor bipolar) operando en su "zona lineal" o uno pasivo (diodo zener) operando en su zona de ruptura. El dispositivo regulador está diseñado para actuar como una resistencia variable, de forma que se modifique el valor resistivo en función de las variaciones de la tensión de entrada para mantener constante la tensión de salida.

- **Reloj del sistema (CLK O SCK).**

- **Revoluciones por minuto (rmp).**

- **Robot**: proviene de la palabra checa “robota” y significa trabajo. Un robot es un manipulador multifuncional reprogramable diseñado para mover materiales,

partes, herramientas o dispositivos especializados a través de movimientos programadospara la ejecución de variedad de tareas.

- **rutina de servicio de interrupción(ISR).**

- **Serial Peripheral Interface (SPI).**

- **Slave Select (SS).**

- **Static Random Access Memory (SRAM).**

- **System Clock (SCL).**

- **System Data (SDA).**

- **tecnología infrarroja (IR).**

- **Tecnología TTL (Transistor transistor logic).** Lógica transistor a transistor. Tecnología de construcción de circuitos electrónicos digitales con la que trabaja el Arduino. Basan su funcionamiento en la presencia de transistores bipolares PNP Y NPN como elementos de entrada y salida. Trabaja con dos niveles lógicos, bajo (0-0,8V) y alto (2,4-Vcc).

- **Universal Serial Bus (USB)**

2.7.- REQUISITOS DE DISEÑO

En este capítulo de la memoria se describirán las bases y datos de partida establecidos previamente a la realización del proyecto. Los requisitos de diseño conformarán el eje sobre el que girará la elaboración de este proyecto y de alguna forma, concretará la forma de realización del presente trabajo en cuestiones de software, hardware y ergonomía. A continuación se expondrán los requisitos de diseño:

- Empleo una placa Arduino para controlar el robot.

- Construcción de chasis del robot en metacrilato o con tecnología 3D.

- El robot deberá salir autónomamente, sin ningún tipo de ayuda exterior, de un laberinto.

- No se especifican las dimensiones de las calles del laberinto, pero sí que debe presentar forma de matriz cuadrada.

- El robot deberá tener suficiente autonomía para realizar un mínimo de expediciones dentro del laberinto. No se especifica el número, pero una cifra aceptable podría ser de 10 expediciones.

- El avance del robot deberá ser en todo momento controlado, no pudiendo de ninguna manera avanzar dando tumbos por el laberinto, es decir, sin ningún tipo de control.

- No se especifican dimensiones mínimas ni máximas del robot. Tampoco se especifican los componentes que deben ser utilizados para la construcción del mismo. Estos serán elegidos a la par que avance el estudio y surjan diferentes necesidades. Es necesario justificar cada elección.

- La programación y los componentes utilizados deberán ser explicados lo más claro y simple posibles para ayudar al completo entendimiento de cualquier usuario, de forma que su uso sea intuitivo para él y que incluso sirva de plataforma para realizar diferentes pruebas y facilitar el aprendizaje de la programación en Arduino.

2.8.- ANÁLISIS DE SOLUCIONES

En este capítulo se indicarán y explicarán las diferentes alternativas estudiadas en el desarrollo del proyecto, así como las motivaciones que llevaron a

tomar y a rechazar cada uno de los diferentes caminos explorados hasta elegir el definitivo, que conducirá a la construcción y posterior correcta puesta en marcha del robot.

2.8.1.- Elección del microprocesador

Uno de los requisitos del diseño del proyecto es utilizar el Arduino, pero no el modelo, por lo que habrá que decantarse por una de las muchas opciones que el mercado ofrece.

Con el estudio de las características de las principales opciones de placas Arduino que ofrece el mercado ya realizado, es preciso establecer una serie de criterios de selección que definan la solución óptima para el proyecto, que serán los siguientes:

Prestaciones requeridas. Dependiendo de las prestaciones requeridas por el sistema, cada placa será más o menos adecuada para el proyecto. Dentro de este primer criterio de selección se engloban las siguientes características:

- Velocidad del procesador. Determina la rapidez con la que el sistema es capaz de ejecutar las instrucciones. Aunque para la mayoría de Arduinos la cadencia es de 16 MHz, hay otras opciones más potentes como 48 MHz o 84 MHz y menos potentes, como 8 Mhz. Como se pretende que el robot avance por el laberinto controlando diferentes parámetros y actuando en consecuencia a tiempo real se determina que cuanto mayor sea la velocidad del procesador mejor se controlará el robot, ya que los datos recibidos por los diferentes sensores que se utilicen se recibirán y procesarán antes y en consecuencia, también se responderá antes.

- Tamaño registros. Junto con la velocidad del procesador vista en el punto anterior, determinan la potencia de la placa y, como en el caso anterior a mayor registros mayor potencia y, por tanto, mejor para el funcionamiento del sistema.

- EEPROM. Como ya se ha visto, la EEPROM almacena la información que el programador considere necesaria para el correcto funcionamiento del programa. Las primeras ideas sobre el diseño del programa recogían el hecho, como al final se realizará en el programa definitivo, de almacenar el camino recorrido por el robot dentro del laberinto, por lo que la memoria EEPROM tendría importancia como criterio de selección del Arduino. Aunque no todas las placas tienen memoria EEPROM, si que permiten simularla, por lo que esto no lo habrá que tener en cuenta, pero sí la capacidad de memoria. Estudiando la capacidad de las EEPROM de todas las placas y la capacidad requerida se determina que esta característica tampoco es determinante, ya que el recorrido del robot nunca excederá ninguna capacidad que las diferentes opciones ofrecen.

- Memoria SRAM. Almacena las variables de ejecución del programa. El programa que se implementará se determina que no será muy extenso ya que ni la función que tendrá la aplicación abarca un amplio rango de funcionalidades, ni se cumpliría el requisito de implementar un programa lo más optimizado posible, es decir, que utilice el mínimo de recursos para realizar su cometido. La memoria SRAM tampoco será un criterio de selección de la placa.

Tamaño del proyecto. El programa o sketch se guarda en la memoria flash del Arduino. Como se puede comprobar en las características de cada placa estudiada, existen Arduinos con 32KB o 256 KB, entre otros. Al igual que sucedió con la memoria SRAM, la búsqueda de la mayor optimización posible, así como la funcionalidad del robot objeto de este proyecto, hacen que la memoria flash tampoco sea un criterio.

Tamaño de la placa. El tamaño de la placa es determinante a la hora de construir la estructura del robot, ya que tendrá una gran importancia en el tamaño del mismo. Aunque no es un factor de selección ni mucho menos concluyente, si lo es para tener en cuenta. Desde el tamaño del Mega o el Due hasta el más reducido del micro existen diferentes opciones.

Tamaño de la placa. El tamaño de la placa es determinante a la hora de construir la estructura del robot, ya que tendrá una gran importancia en el tamaño del mismo. Aunque no es un factor de selección ni mucho menos concluyente, si lo es para tener en cuenta. Desde el tamaño del Mega o el Due hasta el más reducido del micro existen diferentes opciones.

Periféricos. A la hora de seleccionar la placa es muy importante conocer el tipo de periféricos que se van a conectar.

El primer criterio en cuanto a periféricos sería el voltaje con el que trabajan, ya que sería óptimo que trabajasen con un voltaje similar al que ofrece Arduino. Con este criterio se realizan los primeros descartes ya que tras un vistazo por la web se determina que la mayoría de periféricos que se pueden encontrar en el mercado trabajan con unas características como son 5V de alimentación y entre 20 y 40 mA de corriente. Por este motivo, aunque con el correspondiente acondicionamiento se podría usar cualquier placa, se descartan algunas de ellas, por motivos de reducción de recursos, de costes y simplificación del sistema. Los Arduino Zero, Yun, Due y Fio quedan descartados. Por otro lado aún quedan el Arduino Uno, Mega, Ethernet, Fio, Nano y Micro.

El segundo criterio en cuanto a periféricos sería el número de entradas y salidas necesarias. Aunque en un principio no se sabe el número exacto de entradas y salidas serán necesarias, si existen unas primeras ideas o primeras nociones sobre lo que se precisará.

Como se pudo ver en las características, no todas las placas presentan ni el mismo número de entradas analógicas, ni el mismo número de salidas, ni el mismo tipo de AD para las entradas, unos de 8 bits, otros de 10 o de 12 bits. Casi ninguno cuanto con salidas analógicas basadas en un DA, sino que algunos lo sustituyen con una PWM. En realidad esto no supone un aspecto a tener en cuenta ya que no se prevén utilizar estos pines.

Se conoce que no se utilizarán ni entradas ni salidas analógicas, por ello solo habrá que fijarse en las digitales y el las PWM. En un principio se estima que

como mínimo se precisarán 8 pines para el controlador de motores y los sensores y tres PWM para el servo y el control de la velocidad de los motores. Esto hace descartar una opción de las todavía posibles, el Arduino Uno, ya que sus 10 pines digitales y sus 4 PWM dan poco margen a la hora de desarrollar el proyecto. El Ethernet y Fio con 14 y 4 respectivamente y el nano con 14 y 6 no se descartan pero hay que tener en cuenta sus limitaciones.

Comunicaciones. En los criterios establecidos del proyecto no se contempla que sea necesario ningún tipo de comunicación. Aun así se considera necesario una conexión USB para facilitar la programación, por lo que se descarta el Arduino Fio. Por otro lado para una posible colocación de una pantalla LCD es necesario que tenga comunicación I2C o SPI, como sucede con las posibilidades aún vigentes a estas alturas. La comunicación Ethernet no es necesaria ni se contempla ya que supondría un gasto de recursos considerable para la función que pudiera tener en la aplicación final, motivo por el cual el Arduino Ethernet en un principio se descarta, ya que ni esta propiedad ni su capacidad de conexión de una memoria microSD, características definitorias de esta placa, son interesantes para el proyecto.

Alimentación. La alimentación es un tema bastante crítico como ya se verá en su correspondiente punto un poco más adelante en el presente proyecto. Todas las placas aún restantes presentan las mismas características de alimentación, es decir, regulador de tensión y margen de alimentación de 6 a 20V. Aunque no todos presentan conexión tipo Jack esto no se considera importante.
Precio. Por precio definitivamente se descarta el Ethernet, ya que es similar al del Mega y las prestaciones que aporta importantes para el proyecto son superiores. Los Arduinos Fio, Nano, Leonardo y Micro presentan precios parecidos, entre 20 y 30 €, mientras que el Mega asciende a un total de 50 €.

Información y comunidad. La comunidad que respalda cada placa y la información disponible para cada una también es un aspecto muy a tener en cuenta a la hora de decantarse por una opción. Sin duda la opción que gana en este aspecto es la del Mega, con una comunidad detrás muy fuerte y mucha información en libros e internet sobre el, es sin duda, junto al Uno, la opción más conocida y más

utilizada.

Elección final. Una vez establecidos y estudiado los criterios de selección, así como descartado ciertas opciones, es preciso decantarse por una opción. En cuanto a tamaño sin duda ganan el Micro y el Nano, con un tamaño muy inferior al resto de opciones y, sin duda pierde el Mega, pero el tamaño realmente no es significante ya que no es un requisito del proyecto ni el tamaño del robot ni del laberinto, por lo que tampoco es algo determinante. En cuanto al precio, el Mega también sale dagnificado, pero su superioridad en cuanto a pines digitales y pines PWM hacen decantarse por esa opción debido a la incertidumbre que causa el no conocer exactamente el número de pines necesario de cada tipo. También ayuda a esta elección la gran cantidad de información existente y la fuerte comunidad que lo respalda. Cabe mencionar que el Arduino Leonardo y el Micro son sin duda una gran opción para un proyecto como este y podrían utilizarse perfectamente.

2.8.2.- Chasis del robot

El primer prototipo del robot fue construido con el chasis del kit Smart Car para Arduino. En un primer momento se tomó esta decisión para tomar los primeros contactos con un robot de esta índole. Este prototipo, estructuralmente, contaba únicamente con los elementos de dicho kit, sin ninguna modificación.

A la vez que se iban realizando las primeras pruebas, surgió la necesidad de ir añadiendo nuevos componentes. Por ello, y por ciertos problemas que surgían de la utilización de este chasis, se apostó por la construcción de nuevos diseños más adaptados a las necesidades existentes en su momento.

Los nuevos diseños, en un primer lugar se realizaron a partir de tableros de madera de reducido espesor, pero con consistencia, es decir, que su poco espesor no hacían que se rompieran ni deformaran fácilmente. Dichos diseños solamente eran prototipos, en ningún momento se realizaron con el fin de ser el robot definitivo, que desde un primer momento, como recogen los requisitos de diseño, estaría construido en metacrilato.

En estos diseños, la posición de colocación de las ruedas fue el principal hándicap, ya que ciertos aspectos del funcionamiento del robot no eran los esperados y se probaron diferentes diseños. El principal problema que presentaba era el avance a través de las calles del laberinto sin chocar con las paredes. Esto suponía un enorme problema, ya que el primer paso para que un robot salga de un laberinto, como podría parecer en un primer momento, no es la utilización de un algoritmo que sea capaz de resolver el problema, sino que el robot sea capaz de avanzar correctamente por las calles, ya que, por muy bueno que sea el algoritmo, el robot quedará atrapado y el principal objetivo de este proyecto es que logre salir de él. El algoritmo sería el paso final, una vez que el robot ya lograse avanzar por las calles y encrucijadas propias de un laberinto.

En la búsqueda de lo anterior, en cierto momento se apostó por colocar las ruedas en la parte posterior del chasis y la rueda loca en la parte trasera. De esta forma se consiguió lograr un funcionamiento satisfactorio en las rectas. En la parte delantera de la estructura, un sensor de ultrasonidos controlaba la distancia lateral con la pared, y de acuerdo con esta distancia, por programa, los motores regulaban su velocidad para girar en un sentido u otro. La regulación era satisfactoria, algo que con otros diseños, con la misma metodología, no se conseguía. Esto se debía a la colocación opuesta, longitudinalmente, de ambas ruedas y del sensor. De esta manera el eje de giro se encontraba en la parte posterior de la estructura, por lo que se regulaba mejor la trayectoria, ya que este no se desplazaba, sino que lo hacía la parte delantera primero y al esta hacerlo, los motores actuaban en consecuencia.

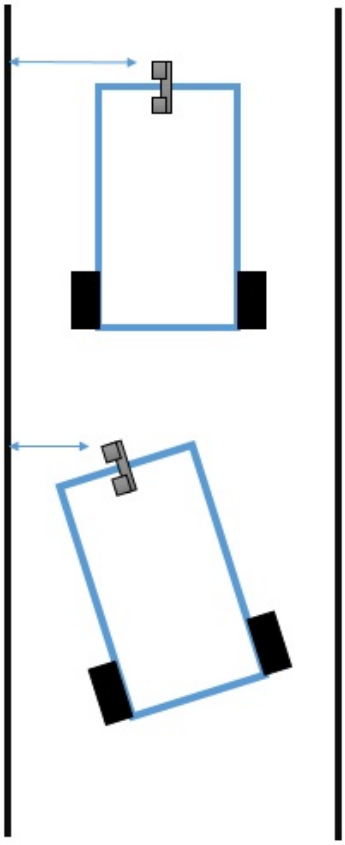


Figura 2.8.2.1.-Estructura del robot con los motores en la parte trasera.

El problema del diseño anterior venía a la hora de realizar los giros. Al situarse en la parte posterior las ruedas, el radio de giro abarcaba toda la longitud del vehículo, por lo que los giros debían de ser muy finos, una gran problemática. Por ello, el diseño de este prototipo fue desechado.

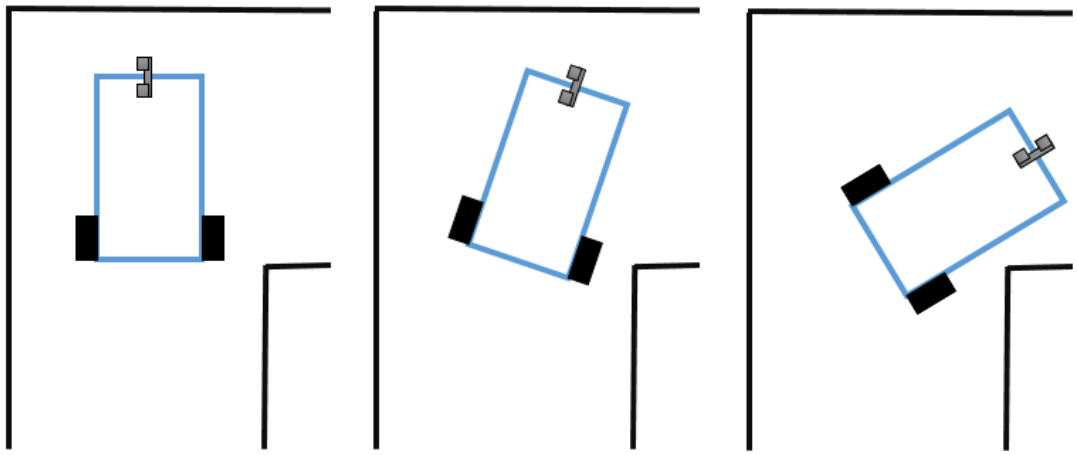


Figura 2.8.2.2.-Giro derecha con ruedas parte trasera.

Después de eso se optó por probar diferentes opciones. Una de ellas y que recibió mucha atención y experimentó numerosas pruebas fue la utilización de 4 ruedas. Una vez estudiado el caso se determinó que para el cometido que estaba siendo construido el robot, la presencia de 4 ruedas no era para nada determinante y, como se buscaba eficiencia y optimización, se desechó esta idea.

Finalmente, tras realizar numerosas pruebas, se apostó por la vuelta a la utilización del primer chasis y realizar sobre el las modificaciones necesarias, una vez ya se conocían y se disponían todos los componentes. Aunque para este montaje la regulación del avance por el pasillo no fuera tan buena, como solución general es mejor, ya que los giros los realiza sin ningún problema.

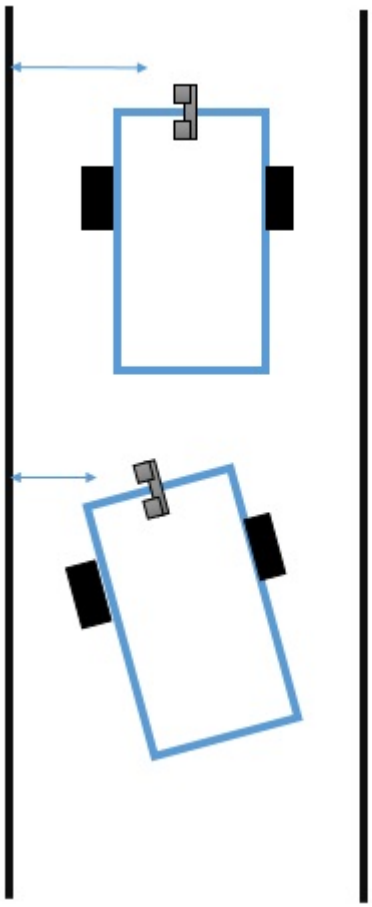


Figura 2.8.2.3.-Ruedas parte delantera.

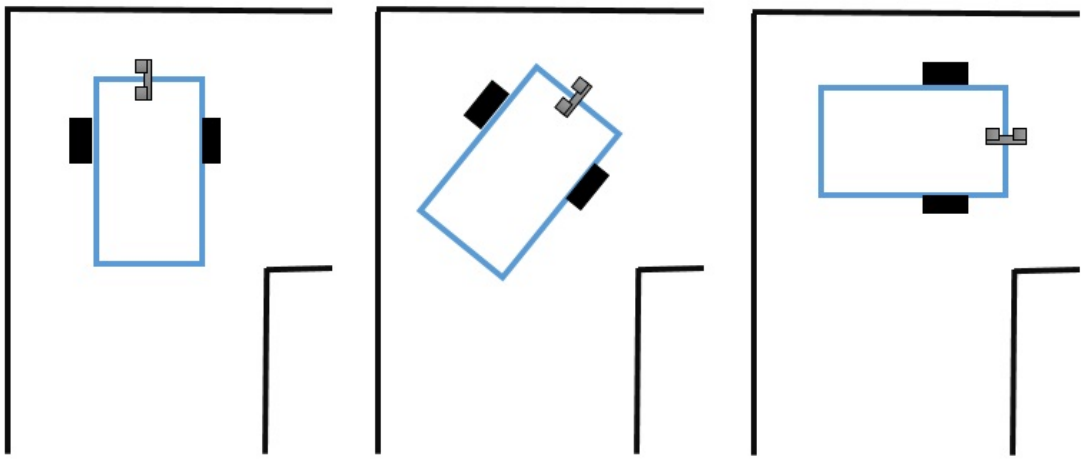


Figura 2.8.2.4.- Giro hacia la derecha con ruedas delanteras.

Para la construcción de la estructura se utilizó una plancha de metacrilato, a partir de la cual se construyeron diferentes bloques para el montaje del robot, entre los que se encuentran:

- Caja para la batería.

- Caja para el Arduino.

- Soportes para los sensores de ultrasonidos.

- Cubierta placa Arduino sobre la que se colocará el controlador de motores y el convertidor DC-DC.

- Estructuras para la conducción de los cables.

- Soporte del opto interruptor.

En un principio se intentó distribuir todos los componentes en la base del robot, pero esto resultó imposible, por lo que se tomó la decisión de montar un segundo piso para una mejor distribución.

La distribución de todos los componentes tiene un porqué y no se eligió al azar, sino que es fruto de un arduo análisis, trabajo y de la denominada prueba-error. Desde un principio hubo una idea general de como los componentes irían distribuidos, pero esta idea fue moldeándose y cambiando a la par que se construían los diferentes prototipos elaborados a lo largo de la elaboración del presente proyecto, así como del robot definitivo.

Desde el principio se tuvo clara la posición del Arduino. Este estaría en posición longitudinal, con el puerto USB apuntando a la parte trasera de la estructura y totalmente accesible para poder llevar a cabo la conexión con el ordenador con total comodidad y así realizar las pruebas pertinentes analizando el programa a través del puerto serie y las cargas de los programas.

La batería se coloca justo a la altura del eje de las ruedas, de forma perpendicular a la base del robot. Justo delante de esta, en la parte frontal se encontrará el servomotor, unido a la base desde abajo a su vez, colocado de tal forma que cuando gire, el sensor que mueve no toque con nada. Delante de este sensor, a su vez, se encuentra el otro sensor de ultrasonidos, fijado a la base. Para la colocación del servo fue preciso el uso de un taladro para darle forma a la placa base.

El convertidor DC-DC, junto al controlador de motores, se colocan en la parte superior de la estructura. Esto se debe a razones de ergonomía del robot, ya que el controlador de motores presenta 6 entradas que se conectarán al Arduino y 4 salidas que lo harán a los motores y desde una posición superior se pueden distribuir mejor los cables. En cuando al convertidor DC-DC, se debe básicamente a que debe estar accesible para regular la salida del mismo.

Los opto interruptores se colocan en la parte inferior de la placa base del robot. Para ello se construyen dos soportes con base en los motor DC que mantendrán a estos sensores en una posición fija para que pueda pedir las vueltas del encoder sin tocarlo.

2.8.3.- Conexiones eléctricas

Todos los sensores y actuadores utilizados, a excepción de los motores, trabajan con valores de tensión de 5V, por lo que se utilizarán las patillas de Arduino que aportan 5V regulados. Por ello no es necesario variar el valor de referencia del AD para las salidas analógicas como se explica en el apartado de entradas analógicas de los antecedentes, ya que todas están referenciadas a 5 V, aprovechándose todo el rango del AD.

Ante la insuficiencia de pines de alimentación de 5V y de masa de Arduino, se colocan 10 pines de ampliación de la salida de 5 V del Arduino y otros tantos de la masa. Lo mismo se hará con las entradas y las salidas del módulo convertidor de tensión, para permitir, mediante el voltímetro, leer la tensión de la batería para conocer su estado y cargarla cuando sea preciso y por otro lado, leer

la tensión de salida del regulador, para actuar en consecuencia si la salida no es la deseada.

En un principio para alimentar el sistema se conectaba directamente la salida del convertidor a la patilla Vin, pero por motivos de comodidad para el usuario se añadió un interruptor en la zona central de la placa base del montaje. Este conecta o desconecta la alimentación de la batería con la entrada del convertidor DC-DC.

2.8.4.- Alimentación

Uno de los requisitos básicos de diseño es que el robot deberá ser autónomo, por lo que precisará de una alimentación externa que no sea la del PC por medio del cable USB. Descartada esta opción, habría que decidirse entre las preparado restantes formas de alimentación para las que está preparada la placa controladora, bien sea una alimentación directa de 5V o utilizando el regulador de tensión propio con el que cuenta el Arduino.

Utilizando el regulador es importante saber que el rango de alimentación recomendado por el fabricante es de entre 7 y 12 V, aunque los límites son de 6 a 20V, pero no se recomienda llegar a estos niveles de tensión, ya que por arriba se requiere un esfuerzo excesivo para el regulador y por abajo la caída de tensión que supone el regulador impediría que llegaran los 5V que precisa para funcionar y es aconsejable utilizar el rango recomendado. Si la opción escogida fuera por la alimentación directa, sin mediar con el regulador, debería de usarse una fuente de alimentación regulada con una gran precisión que dé, exactamente, 5 V, en el pin con el mismo valor. Esta forma de alimentar, además de ser más tediosa por requerir más precisión a la hora de alimentar, supone una continua amenaza de daño para la placa, ya que cualquier pico de tensión provocaría un daño irreparable.

Para el robot la opción escogida es la de utilizar el regulador de voltaje interno de la placa, no solo porque es más cómodo y se asumen menos riesgos, sino que, como ya se vio en capítulos anteriores, se tendrá que alimentar un

módulo controlador de los motores de corriente continua del robot, ya que aunque el resto de sensores y actuadores funcionen con 5V, para el controlador se hacen insuficientes, y para obtener una velocidad aceptable de los motores DC, para el avance y giro del robot, se precisarán niveles de tensión más altos.

Existen diferentes formas de alimentar el Arduino mediante el regulador de tensión, como lo serían:

- Alimentación 6-12V por la clavija Jack.

- Alimentación 6-12V entra el pin Vin y GND.

- Alimentación de 5V por el puerto USB.

Dentro de estas dos opciones la elección escogida fué la que requiere alimentación por la patilla Vin, simplemente por ser una opción más sencilla al no tener que disponer de una entrada Jack.

Los primeros diseños del robot contaban con 4 pilas AA de 1,5 V, colocadas en serie en su correspondiente portapilas. Esta alimentación era una opción totalmente válida, pero a la larga no era una opción económica, además de que como ya se comentó, nunca aportaban un valor exacto y sufrían caídas de tensión con facilidad. Aunque puntualmente pueda echarse mano de una, no son la solución para nada ideal.

En el caso de escoger este tipo de alimentación cabe resaltar que la experiencia demostró que vale la pena utilizar pilas alcalinas de calidad. Las prestaciones que ofrecen las pilas baratas son muy pobres y no solo había modelos que conservaban poco tiempo sus características eléctricas, si no que había otros que ni siquiera permitían alimentar correctamente el sistema ni estando recién estrenadas.

El uso de este tipo de pilas desde un primer momento se estableció como provisional, para realizar los primeros montajes y las primeras pruebas.

Además de las pilas de 1,5 V, también se probó a alimentar el sistema con una pila de 9V, pero su baja capacidad impedía que la aplicación funcionara correctamente, por lo que esta posibilidad no es una opción.

Una vez que se avanzó en el estudio y ya se conocían los requerimientos que se buscaban, se empezaron a buscar alternativas para la alimentación del robot.

La opción utilizar como alimentación un banco USB nunca fue una opción real, ya que aportan solamente 5 V, que se hacen realmente cortos para alimentar los motores DC que moverán el robot y el propio Arduino. Además son caros comparados con otras opciones mejores.

Utilizar 5 pilas recargables de 1,2 V sin duda es una opción muy buena para alimentar un sistema como este y siempre fue una opción a tener en cuenta, al igual que el uso de dos baterías de litio 18650 de 3.7V o una batería NIMH, otras dos grandes opciones, y todas realmente válidas, pero la opción escogida, como ya se comentó en los antecedentes, es una batería LIPO.

Sin lugar a dudas las baterías LIPO son la opción más avanzada y más habitual en la construcción de sistemas de modelismo. Esta opción es la seleccionada para alimentar el robot.

Este tipo de batería supera con creces las especificaciones de cualquier otro tipo de baterías como los que estudiadas en el apartado de alternativas. En comparación, por ejemplo, con las NIMH, que son las más competitivas en cuanto a lo que a características se refiere, presentan una densidad de energía de entre 5 y 12 veces mayor para un mismo peso y para una misma capacidad son unas 4 veces más ligeras. Además tienen una capacidad de descarga bastante superior también.

Dentro de las baterías LIPO, el abanico de opciones es enorme, incluso más que el de diferentes tipos de Arduino. Para saber qué batería elegir, primero

es preciso analizar la carga que se tendrá (motores, sensores…) y determinar la corriente que se necesitará para el correcto funcionamiento del sistema. La corriente que aporte la batería debe ser suficiente para asegurar el funcionamiento del robot durante un periodo de tiempo aceptable así como el consumo máximo de la carga deberá ser menor que la corriente máxima que sea capaz de suministrar la batería y aparte, prever cierto margen de seguridad.

El tiempo de duración de la batería será un aspecto importantísimo, ya que es un aspecto vital para la autonomía del robot. Como ya se explicó en el correspondiente punto de los antecedentes, sabiendo la capacidad de descarga de la batería y su velocidad de descarga se puede estimar el tiempo de duración de la misma. La capacidad de descarga viene reflejada en la batería, mientras que la velocidad de descarga dependerá de cuanto consuma la carga.

Entonces, para seleccionar el modelo de batería, será preciso conocer la carga del sistema, que será calculada en el apartado 3.2.2 de los anexos, en el correspondiente a los cálculos.

Sabiendo lo que consumirá la carga como mínimo se puede empezar a acotar la búsqueda. En el mercado existen baterías LIPO con capacidades de descarga diversas, desde 500 mAh hasta 5000 mAh, pasando por valores tan diversos como 800, 1000, 1200,2000 mAh…., teniendo en cuenta que a mayor capacidad, mayor será el precio también. Por ejemplo una batería de 800 mAh, si se cumplieran los pronósticos de 807,5 mA de consumo, duraría una hora, como se indica en el apartado 3.3.3, un tiempo de duración más que aceptable y que cumpliría el requisito del proyecto correspondiente a la autonomía del robot. Como la estimación se hace por lo bajo, sería conveniente establecer un margen, por lo que se recomienda un valor mayor.

El otro criterio para seleccionar la batería es la tensión que debe proporcionar la batería. Para alimentar el Arduino es necesario un voltaje superior a 5V y es recomendable que sea menor que 12 V, además los motores deben se alimentarse como mínimo a 6V para que funcionen correctamente. Como cada celda de las que forma la batería suele presentar un voltaje de entre 3.7 y 4.2 V,

una batería de una sola celda no serviría. Entre decantarse por una batería de 7.2 V y otra de 11.1V, que serían las que cumplirían los parámetros de alimentación, finalmente se decanta por la opción de 11.1V, ya que la diferencia de precio es mínima y aporta mayor margen.

2.8.5.- Sensores

Como ya se ha establecido en el apartado de componentes finales, el robot contará con dos sensores de ultrasonidos y dos opto interruptores de infrarrojos.

Desde un principio se tuvo claro que la elección de un sensor de ultrasonidos para controlar el avance por el laberinto era la mejor opción del mercado para alcanzar este cometido. Otra opción, como la de utilizar un sensor de infrarrojos, fue desechada desde un primer momento por la problemática que producen tanto las interferencias que produce la luz en ellos como la reflexión del haz de luz infrarroja en ciertos materiales.

Lo que no estuvo tan claro en un principio y, en realidad, que no estuvo claro hasta que el proyecto empezó a estar en un estado avanzado, era el número de sensores de ultrasonidos necesarios para optimizar el funcionamiento del robot y en qué posición del mismo deberían estar colocados.

El primer prototipo solamente contaba con un sensor de ultrasonidos fijo en la parte delantera del chasis, con el objetivo de que el robot avanzara hasta que encontrar un obstáculo, momento en el que debería de parar.

Una vez se logró lo anterior surgió la necesidad de que cuando el robot detectara un obstáculo, es decir, una pared, actuara en consecuencia a la situación a la que estuviera, es decir, detectara por qué camino podría seguir, bien fuera girar a la derecha si por allí hubiese camino libre, a la izquierda o cambiar el sentido. Para ello se precisaban dos sensores de ultrasonidos más, uno que detectara presencia a la derecha y otro a la izquierda para detectar los caminos libres. Siguiendo esta disposición se modificó el prototipo existente en ese momento, modificando el programa en consecuencia y logrando el objetivo

perseguido.

El robot, con tres sensores de ultrasonidos ya sabía decidir sin problema para donde tenía que ir cuando se encontrase en una encrucijada, sin embargo, se desechó esta idea y se optó por reducir a dos el número de este tipo de sensores y montar uno en un servo que tuviera un rango de 180º, para así poder controlar ambos lados cuando se le requiriera. Esta decisión simplemente se tomó por fines didácticos, ya que como ya se expuso en el objetivo del proyecto y en los requisitos de diseño, se pretende que este documento sirva de base de aprendizaje tanto de programación en Arduino, como del control mediante el mismo de diferentes componentes y de esta manera se estudia un componente diferente más, además de ser uno muy importante y recurrente para una gran cantidad de proyectos diferentes.

El sensor solidario al servo no solamente tiene la función de determinar qué dirección tiene que tomar el robot cuando se encuentre frontalmente con una pared, si no también, controlar la distancia lateral con la pared en el avance frontal, es decir, para ayudar al robot a regular la distancia lateral para avanzar lo más centrado posible.

Dentro del abanico de sensores de ultrasonidos que ofrece el mercado, el modelo elegido es el HCSR04. Su bajo precio y su sencillez fueron, a priori, determinantes a la hora de decantarse por este dispositivo para implementar la aplicación a la espera de conocer su funcionamiento real. Una vez que se comienza a hacer pruebas con él, se determina que es la solución óptima para el cometido que se busca, no contemplándose otras opciones. Aunque no es muy preciso, la precisión no es crítica en la aplicación que se pretende implementar y cumple su cometido a la perfección.

Que el robot precisaría un sensor de ultrasonidos para detectar frontalmente los obstáculos propios del laberinto, como se mencionó anteriormente, estuvo claro desde un principio, pero no tanto el uso de uno para controlar el avance a través de las calles del mismo controlando la distancia lateral. Para este cometido se planteó el uso de una IMU, que dotaría al robot de

la capacidad de colocarse siempre paralelo a las paredes. La idea era sencilla, la IMU detectaría desplazamiento con respecto al eje longitudinal del robot, y este mediante la regulación de los motores, corregiría esta desviación para colocar siempre al robot en la posición deseada y conseguir un avance centrado y paralelo a las paredes laterales.

Por ello se adquirió una IMU, la MPU6050. Con la ayuda de librerías y ejemplos encontrados en internet se llevaron a cabo las primeras pruebas, las cuales no fueron satisfactorias ya que aunque la inclinación de los ejes x e y, es decir, los ejes paralelos al suelo, se medía correctamente, lo que en realidad se perseguía era conocer la inclinación con respecto al eje perpendicular al suelo, el eje z, y con este módulo era imposible, era necesario una IMU con un magnetómetro incorporado o un módulo magnetómetro. La opción de la IMU entonces se desecha totalmente.

En ese momento se empiezan a plantear otras alternativas a la espera de adquirir un magnetómetro y se plantea la opción de controlar el desplazamiento con uno o dos sensores de ultrasonidos. Dicho esto se empiezan a realizar las primeras pruebas y a obtener los primeros resultados. Aunque estos en un primer momento no eran satisfactorios, como ya se comentó anteriormente en el apartado que se habla de la evolución del diseño del chasis del robot, había aún mucho margen de mejora y su uso era sencillo.

Una vez adquirido el magnetómetro se realizan diferentes pruebas con él. Los primeros resultados obtenidos son satisfactorios, se obtienen por ejemplo perfectamente las componentes magnéticas o la desviación con respecto al norte magnético, pero en la práctica, al montarlo sobre el robot, el control de este con respecto a la desviación con el norte magnético se hace bastante inestable. Para obtener unos resultados satisfactorios sería necesario un control muy fino y tirar de recursos de la placa, además de que se llevarían a cabo cálculos más o menos complejos, lo que podría hacer más lenta la ejecución del programa y en consecuencia, producirse errores en el avance del robot a través de las calles del laberinto, como no detectar una pared frontal a tiempo, además de que se busca implementar un programa que permita al robot alcanzar al robot de la forma más

optimizada posibles, y con un magnetómetro, se usan demasiados recursos para una aplicación que no es realmente crítica. Finalmente se apuesta por el uso del sensor de ultrasonidos ya que los resultados eran más prometedores y su uso para la regulación del avance del robot era más sencillo, aunque se considera la opción del magnetómetro como una gran opción si se profundiza más en su funcionamiento.

El magnetómetro y la IMU seleccionadas, HMC5883L y MPU6050 respectivamente, se eligen por su bajo coste, sus características eléctricas y la capacidad de comunicación I2C.

En cuanto a los opto interruptores, serán los encargados de contar las revoluciones de ambas ruedas del robot mediante el uso de un encoder. Se determina que 35 revoluciones del encoder equivalen a una casilla del laberinto, por lo que se considera que si se cuentan 35 interrupciones de cada uno de los opto interruptores, aproximadamente se avanza una casilla. En realidad esto no es exacto, pero tampoco es determinante ya que el error nunca llegaría a ser tal que el robot cuente mal las casillas por las que circula ya que al llegar a cada giro el error que pudiera haber se pone a cero.

En el momento en el que surge la necesidad de conocer el recorrido del robot lo hace también, en consecuencia, la de contar las revoluciones de cada rueda. En el mercado existen numerosas opciones.

Por un lado están los sensores de tipo electromecánico como tacogeneradores y dinamos tacométricas, pero realmente resultan bastante obsoletos y su coste y peso son elevados por lo que quedan totalmente descartados, ya que la principal virtud que presentan, fiabilidad y robustez, no son determinantes para esta aplicación. Por otro se encuentran sistemas más empleados en la actualidad como los generadores de pulsos, que producen una cantidad de pulsos en cada unidad de tiempo proporcional a la velocidad del eje al que son solidarios. Existen numerosos tipos de generadores de pulsos, pero los más conocidos son los ópticos y los basados en el efecto Hall.

Una vez que se determina que la solución idónea para contar las vueltas de ambas ruedas y controlar así el recorrido del robot es la de utilizar un generador de pulsos, es preciso decantarse entre una de las dos opciones propuestas. Sin duda los sensores ópticos, para esta aplicación, son una mejor opción, ya que el motor DC seleccionado viene adaptado para incorporarle un encoder óptico en el eje de giro y solo habría que colocar el par led-fotodiodo correspondiente. Para usar un sensor de efecto hall habría que colocar 20 imanes en la rueda para obtener similar resolución que con el encoder óptico, además del sensor, por lo que, por simplicidad y ahorro de recursos, se descarta esta opción.

El modelo seleccionado como sensor óptico es el Modulo Sensor FZ0888, ya que se trata de un módulo que incorpora en un mismo chip el par led-fotodiodo y un circuito de acondicionamiento que envía directamente una señal al Arduino de presencia o no presencia. Se trata de un módulo barato, de reducidas dimensiones y sencillo de utilizar. Además la estructura del emisor y receptor IR es idónea para colocar en el medio un encoder como el que se dispone.

Las primeras pruebas con ambos sensores determinaron que era preciso colocar un filtro para evitar las mediciones falsas, ya que se estima que se leen del orden de 4 veces más pulsos de los que se generan realmente debido al ruido. Este sensor es altamente sensible a las interferencias y las corrientes parásitas del Arduino le afectan. Analizando las señales generadas por el optointerruptor en el osciloscopio, se observa claramente la señal cuadrada producto de la salida digital del sensor, pero también es evidente el ruido que aparece en la señal. Se produce lo que en electrónica se denominan rebotes cuando la señal pasa de nivel alto a bajo o viceversa.

Para solucionar el problema del ruido hay surgen dos opciones, por un lado colocar un filtro de condensador entre la salida digital. El valor del condensador debería ser tal que mantuviera la señal a nivel alto tanto tiempo, o un tiempo cercano, como dure el pulso del optointerruptor. Por otro lado está la opción de realizar un filtro anti rebotes por programa. Esta última es la opción elegida ya que se probó su funcionamiento y funcionó a la perfección.

Planteada la opción de implementar un filtro por software surgen dos ideas. Por un lado habría la posibilidad de realizar un filtro tal que cuando detectara una señal a nivel alto esperara unos microsegundos para comprobar que el pulso medido durara lo que debería de durar un pulso provocado por una interrupción. Otra opción sería establecer una velocidad máxima asegurando que los motores no la superarán y establecer un tiempo acorde a esa velocidad máxima en la que no debería de haber pulsos. Finalmente la opción escogida es la segunda ya que comprobadas las dos opciones es más exacta, ya que aunque la primera opción funcionaba perfectamente, en ocasiones no se contaba algún pulso. En puntos posteriores se explicará y detallará la implementación de este filtro.

El robot contará con dos sensores infrarrojos para dotarlo de la posibilidad de convertirlo en un seguidor de líneas. Esto no es uno de los objetivos del proyecto, ni ningún requisito, pero que el proyecto sirva de guía de aprendizaje para la elaboración de robots similares al construido y que este sirva de banco de pruebas para futuros usuarios sí lo son, por lo que dotar al robot de ambos sensores es interesante, ya que este, por su constitución y componentes, es idóneo para funcionar también como seguidor de líneas. Además en este proyecto de dan las nociones suficientes de programación para implementar, con el algoritmo adecuado, este sistema.

El último sensor añadido al proyecto es el receptor IR para permitir el control remoto a través de un mando. El modelo receptor IR seleccionado es el AX-1838HS. La justificación de su elección es que proporciona en un solo módulo el sensor infrarrojo junto con un circuito de acondicionamiento que permite conectar el módulo directamente al Arduino, además de presentar un precio muy reducido.

2.8.6.- Actuadores

El modelo de motores escogido ha sido seleccionado principalmente por su reducido precio y su sencillez. Es sin lugar a dudas el modelo más común en proyectos como este. Además cuentan con una carcasa compacta y de una

rueda cada uno, con su respectivo enganche, que la fija al motor para que gire correctamente al ritmo que le marcan el motor y reductora.

Para controlar los motores DC se utilizan las salidas PWM para simular una salida analógica. Las salidas PWM de Arduino Mega son de 8 bits, por lo que los valores que se envíen al controlador de motores deberán estar dentro del rango de 0 a 255.

A lo largo del proyecto se llevaron a cabo múltiples pruebas en cuanto a lo que alimentación de los motores se refiere. A mayores niveles de tensión de alimentación, mayores velocidades se obtenían. En un principio la intención era construir un robot muy veloz, pero se comprobó que cuanto más rápido fuera, más difícil se haría controlarlo, sobre todo en lo correspondiente al avance por las pasillos sin chocar con las paredes laterales y a la hora de frenar cuando se detectara una pared. En conclusión, para un robot de dimensiones tales como el construido en este proyecto, altas velocidades no son recomendadas y como no es una especificación del proyecto se desecharon ciertos valores de velocidad para el funcionamiento del robot.

Aunque las especificaciones de estos motores establecen el rango de voltaje de alimentación de los mismos con un rango de entre 1,5 V y 6 V, la práctica demostró que valores menores de 5 V no eran suficientes para que el motor lograra su cometido, es decir, hacer mover al motor y que estos admitían valores por encima de 6 V, hasta un valor comprobado de 12 V.

Según lo indicado en el apartado 3.2.1 del anexo de cálculos, para un voltaje de alimentación estimado de 6V, se tardaría alrededor de 6,2 segundos en recorrer el largo del laberinto. Como este valor de tiempo entra dentro de los parámetros esperados para el diseño, se establece 6 V como el valor de alimentación del controlador de motores y, en consecuencia, de los propios motores. Este valor será regulado mediante el convertidor DC-DC.

La justificación de la utilización de un servo ya está explicada en el punto correspondiente a los sensores del análisis de las soluciones del proyecto. En

cuanto al modelo elegido, se selecciona el modelo Micro Servo 9g SG90 de Tower Pro, ya que se trata de un servo sencillo, de pequeñas dimensiones y escaso peso que se adapta perfectamente tanto al sensor de ultrasonidos que se colocará solidario a su eje como a la estructura del robot. Sus especificaciones no son realmente altas, pero tanto su resolución de un grado, como su torque llegan más que de sobras para el cometido al que está destinado. Además cumple el requisito principal por el que se filtra la elección de cualquier componente para este proyecto, y es que permita alimentarse a 5 V.

Para el control del servo se utilizará la librería <servo.h>, que de igual manera utilizará las salidas PWM, pero cuyo uso permite al usuario realizar una programación, por decirlo de alguna manera, más cómoda, ya que permitirá escribir por programa los grados que se desea que se desplace el servo, en lugar de escribir el valor de la PWM como en el caso anterior.

2.8.7.- Regulador de tensión

La selección de la batería se hizo en consecuencia de los rangos admitidos por los componentes electrónicos a los que debe alimentar, tanto el Arduino como el controlador de motores y los propios motores DC. Pero que sean compatibles, no quiere decir que las características que aporte la batería sean las idóneas para la funcionalidad perseguida.

El controlador de motores podría alimentarse directamente de la batería ya que, como ya se ha visto, tanto en el apartado correspondiente al mencionado controlador, como en el correspondiente a la batería, los valores de tensión que da la batería y el rango que acepta el controlador son totalmente compatibles. El problema surge en el momento que la batería comienza a descargarse, ya que se producen pérdidas de tensión, que provocan la necesidad de utilizar un regulador de voltaje para obtener una tensión siempre fija para alimentar los motores.

Para reducir y regular el voltaje existe la opción de utilizar un regulador lineal o con uno conmutado. Los reguladores lineales son altamente ineficientes y su rendimiento energético es muy bajo, ya que desperdician grandes cantidades

de energía en forma de calor.

Los valores de potencia con los que se trabaja en esta aplicación no son muy elevados, pero lo suficientes para darle trabajo a un regulador lineal, por tanto el desperdicio de calor empezaría a ser un problema. Por ello la opción elegida son las fuentes conmutadas, ya que son mucho más eficientes y se evitan posibles problemas derivados de la disipación de calor y la posible necesidad de añadir un disipador de calor. Además la batería utilizada es de 11.1V y para valores cercanos a 12V ya se recomienda utilizar reguladores lineales en cascada.

Es importante saber que la eficiencia típica de un regulador lineal suele ser de un 40% y puede caer hasta un 15% con facilidad, por eso nunca deberían usarse en proyectos que funcionen a baterías, frente a un 85% típico de una fuente conmutada.

Las fuentes conmutadas son recomendables siempre que usemos circuitos alimentados por baterías, pero imprescindibles cuando el consumo aumenta por encima de más o menos medio amperio, porque el calor generado, y su evacuación, empezaran a dar problemas que irán complicándose cada vez más.

Ambos montajes son baratos, tanto los lineales como los conmutados, por lo que el precio a la hora de decantarse por una opción no influye.

La función del módulo regulador en el sistema es la de servir de nexo entre la fuente de alimentación y el controlador de motores. La velocidad con la que gira el motor, para valores similares de control PWM desde el Arduino, está relacionada directamente con la tensión con la que se alimentan. Esta relación es directamente proporcional, a mayor tensión, mayores rpm. Por tanto, si se desea tener un control total sobre el giro de los motores, como sucede en este proyecto, ya que controlar el avance del robot supone uno de los requisitos básicos de diseño, se debe controlar la tensión con la que se alimentan los motores y, en consecuencia, la tensión con la que se alimenta el controlador, ya que los motores se nutren del mismo. Esta búsqueda del valor idóneo de la alimentación hicieron

crecer la necesidad de regular el voltaje de alimentación para obtener una tensión fija, sino poder regular esta tensión fija para alimentar los motores a diferentes tensiones para realizar las pruebas pertinentes hasta establecer un voltaje de alimentación recomendado para implementar la aplicación elaborada en este proyecto. Por ello se apuesta por un módulo regulador, bien fuera elevador de tensión o reductor, permitirá variarla al gusto para probar, con un simple uso de un destornillador, diferentes alimentaciones y analizar la respuesta del robot para cada valor.

Se decide utilizar un módulo ya construido porque el mercado ofrece una gran variedad de reguladores que realmente se ajustan a los parámetros que marcan las exigencias establecidas por el sistema, a un precio muy reducido, al igual que su tamaño y dificultad de uso.

La necesidad de un módulo DC-DC es clara al igual que el tipo de conversor a utilizar. Por un lado podría utilizarse el elevador de tensión elegido y explicado en el apartado. La entrada de este módulo sería la salida de 5 V del Arduino o la de 3,3 V, de forma que el conversor elevaría estas tensiones. En la práctica se demostró que este componente no es válido para elevar la tensión de salida del Arduino para alimentar motores DC, ya que las patillas de Arduino apenas proporcionan 20 mA, corriente demasiado baja para hacerlos funcionar.

La opción idónea es sin duda la del reductor de tensión. A su entrada se conecta la salida de la batería para así poder aprovechar las características que esta presenta, idóneas para hacer funcionar motores como los elegidos para el robot. A la salida se conecta el controlador de motores.

Se selecciona el modelo LM2596S DC-DC Step Down entre la gran variedad de opciones que ofrece el mercado ya que sus características eléctricas son totalmente compatibles con las que ofrece la batería y con las que demandará la carga, tanto los voltajes de entrada, como los de salida y las corrientes admitidas.

2.8.8.- Control de motores

La necesidad de controlar el robot deriva en la necesidad de controlar los motores. Un control básico de estos siempre se realiza por medio de un puente en H, que fácilmente podría construirse, pero el mercado ofrece una gran variedad de módulos compactos, sencillos de usar y muy baratos.

El modelo seleccionado es el LN298N ya que sus características son totalmente compatibles con las buscadas, tanto las salidas de control de las que dispone, como el rango de alimentación de entrada, su reducido tamaño, su bajo precio y su facilidad de control con el Arduino.

El controlador se utilizará con el jumper de 5V, es decir, se utilizará el regulador de tensión interno, por lo que la alimentación de la batería deberá estar dentro del rango de entre 6 V y 12 V, cosa que es cierta.

2.8.9.- Construcción del laberinto

Las primeras pruebas con el robot se realizan con simples tablones de madera, probando los diferentes problemas con los que se podría encontrar el robot dentro de un laberinto. Una vez que se obtienen satisfactoriamente los resultados esperados se procede a la construcción del laberinto para programar y ajustar debidamente el robot para que logre el objetivo principal de este proyecto.

El laberinto se decide construirlo con madera y como viene recogido en el los requisitos del proyecto, en forma matricial, en este caso, de una matriz 6x6.

Para que los sensores de ultrasonidos puedan detectar sin ningún problema las paredes, estas tendrán que medir como mínimo 9 cm de alto, mientras que el ancho de calle deberá ser tal que el robot pueda avanzar y girar sin problemas, teniendo en cuenta que mide de ancho, contando con las ruedas, 16 cm.

Los tablones que formarán la estructura del laberinto se harán a partir de

planchas de madera 100x60 cm de 1 cm de espesor, siendo necesarias un total de 3 planchas. Estos tablones se cortarán con una sierra de calar eléctrica con un ancho de 12 cm y el largo determinado por el diseño escogido. Como el largo de la plancha es de 100 cm, cuando este no sea suficiente, se realizan empalmes como el mostrado en la siguiente figura utilizando cola y puntas:

Figura 2.8.9.1.- Empalmes tablas.

En cuanto a la unión de las esquinas, se realiza atornillándole un par de tornillos en el lateral de una tabla como en la imagen que se muestra a continuación:

Figura 2.8.9.2.- Esquinas estructura laberinto.

Para el diseño y construcción del laberinto es importante tener en cuenta que si se quiere un ancho de calle de 35 cm hay que contar con el ancho de cada tabla para los cálculos, es decir, si el largo del laberinto es de 6 casillas, de 35 cm cada una, el total será de 210 cm. A este total habría que sumarle 5 cm ya que cada tabla mide a centímetros de ancho.

2.8.10.- Almacenamiento del recorrido

Una vez que el robot cumple su cometido de desplazarse por el laberinto nace la necesidad de almacenar el recorrido que realizó para obtener aún un mayor control de su funcionamiento. El control del recorrido se llevará a cabo gracias al conjunto encoder óptico instalado en cada rueda del robot. Como ya se explicó, se contarán pulsos del encoder y, sabiendo que por cada pulso se avanzará 1 cm, se podrá conocer lo que avanza.

Como el laberinto está construido en forma de matriz 6x6 de 35 cm cada casilla, el recorrido se almacenará en forma matricial también. Como se realiza esta función se explicará en detalle en el apartado de programación.

Por programa se determina cuando el robot avanza por cada casilla y así hasta que alcanza la salida, pero esto es preciso almacenarlo para que quede registrado y que el usuario pueda acceder posteriormente a esos datos. Para el almacenamiento de esta información se utilizará la memoria EEPROM de la placa Arduino, cuya función es permitir grabar aquella información que el correspondiente programador quiera almacenar a largo plazo.

La EEPROM de Arduino Mega es de 4 KB, es decir, cuenta con 4095 posiciones de memoria de 1 byte cada una, por lo que sobra para el requerimiento que se le exige, ya que en el peor de los casos se ocuparán 72 posiciones de memoria. En cada posición se almacenará el número correspondiente a cada casilla del laberinto, que irá del 11 al 36.

2.8.11.- Visualización del recorrido

Para que el usuario pueda visualizar el recorrido realizado por el robot existirían dos opciones, o bien que lo hiciera a través del puerto serie del ordenador, o bien a través de una pantalla. Esta última es la opción escogida por considerarse mucho más cómoda y visual para el usuario.´

El modelo de pantalla seleccionado es Display LCD 16x2 HD44780. La justificación de su elección es que se trata de un Display sencillo, de bajo coste y que incorpora un controlador que hace que se pueda manejar directamente desde Arduino. Aunque en el mercado existen Displays similares pero con pantalla más grande, se considera que esta es suficiente para desempeñar su trabajo perfectamente.

El Display elegido cuenta con puertos de comunicación I2C. Esto fue sin duda determinante ya que aunque la comunicación SPI es más rápida y permite mensajes más amplios, para esta aplicación esas dos ventajas no son determinantes, al contrario que la sencillez de la comunicación I2C, que permite comunicarse con solo 2 pines, algo que sí es determinante.

El Display, además de la posibilidad de comunicarse mediante el protocolo SPI, también permite controlarlo por medio de los 16 pines con los que cuenta, ya descritos cuando se habló del mismo. Pero esta opción requiere de más conexiones, por lo que se descarta.

2.8.12.- Control remoto

La necesidad de control remoto surge por simple comodidad de no tener que arrancar el sistema pulsando el interruptor del robot. La opción escogida es la de utilizar la tecnología IR. Otra opción de control remoto sería a través de una red Ethernet, pero para esta aplicación no es recomendada esta opción, ya que consume muchos recursos, es más complejo y es más costosa, ya que precisa de un módulo shield Ethernet. Para la función que desempeñaría realmente no compensa una inversión mayor ni una mayor complejidad de código.

Para la recepción se utiliza el sensor ya descrito en puntos anteriores,

mientras que para la emisión de los códigos de control se utiliza un mando a distancia de cualquier tipo, ya que la mayoría de mandos a distancia convencionales utilizan un protocolo que el receptor utilizado es capaz de decodificar. Quizás podría darse algún problema con mandos antiguas, aunque no es lo más común que esto suceda.

A través del mando el usuario podrá controlar diversas funciones, como el encendido y apagado del robot o el control del programa que permite al usuario interaccionar con la aplicación por medio del LCD, de forma que podrá elegir entre ver el recorrido realizado por el robot, borrar la memoria EEPROM de la placa o ver el camino más corto entre la entrada y la salida del laberinto.

2.8.13.- Programación

La programación se realiza con la placa Arduino y con el IDE del mismo. La elaboración del código se lleva a cabo con un lenguaje de programación básico, apoyándose en las librerías que ofrece Arduino.

Para el desarrollo de la programación de aplicaciones, Arduino dispone de un paquete amplio de librerías con el objetivo de utilizarse de forma rápida y cómoda para los diversos desarrollos que pueden darse. El uso de librerías ayuda a desarrollar programas sin tener que entrar en todos los detalles de cada parte, ya que cada librería ofrece funciones o métodos ya preprogramadas, que ofrecen de una forma rápida y directa desarrollos más complejos de una manera intuitiva y cómoda. Suponen un paso más hacia los lenguajes de alto nivel, ya que permiten simplificar ciertas líneas de código de desarrollo, ocultando el más bajo nivel de programación, el nivel paso a paso. Las librerías utilizadas en este proyecto son las siguientes:

EEPROM	Permite la lectura y escritura de la memoria EEPROM
Servo	Permite controlar el sermotor

IRremote	Permite controlar el receptor de infrarrojos
LiquidCrystal_I2C	Permite controlar el LCD por I2C
Wire	Permite utilizar la comunicación I2C de Arduino

Tabla 2.8.13.1.- Librerías utilizadas.

Los programas incluidos en este proyecto siguen la estructura establecida en la figura 2.4.1.6.1.

2.8.13.- Algoritmo

El algoritmo que resuelva el laberinto deberá de ser, como se establece, no solo en los objetivos del proyecto y en los requisitos de diseño, si no en el propio título del proyecto, lo más óptimo posible. Es por ello que se estudia el algoritmo que consuma menos recursos.

Existen números algoritmos que resuelven el problema del laberinto, pero en búsqueda de la optimización y, en consecuencia, de la simplicidad, se decide elaborar un algoritmo sencillo, basado en la idea de que si el robot sigue siempre la pared izquierda del laberinto, siempre encontrará la salida del mismo.

Como ya se comentó en el apartado 2.8.6, el robot contará con dos sensores de infrarrojos que lo habilitarán para funcionar como seguidor de líneas. Esta función no se implementará en este proyecto, pero sí que se comentará brevemente un algoritmo, por si es de interés del lector, para implementar esta función. Este algoritmo se basa esencialmente en 3 puntos:

1. Si los dos sensores detectan línea, ambos motores deberán tener la misma velocidad.

2. Si el sensor de la derecha detecta línea y el de la izquierda no, deberá regularse los motores para que el robot gire a la derecha.

3. Si el sensor de la izquierda detecta línea y el de la derecha no, deberá regularse los motores para que el robot gire a la izquierda.

2.9.- RESULTADOS FINALES

En este capítulo se describirá el producto final objeto de la ejecución del presente proyecto, teniendo en cuenta tanto el apartado físico, como el de software. El código en bruto se puede encontrar en su anexo correspondiente, pero en este capítulo se explicarán todos los aspectos que se consideren necesarios para su correcta compresión, apoyándose tanto en referencias al código como en flujogramas o esquemas. En cuanto al apartado físico, para describirlo correctamente, se utilizarán tanto imágenes del mismo como referencias a los planos.

2.9.1.- Robot

En este apartado se mostrará y se explicará el robot objeto de la ejecución del proyecto desde un punto de vista del hardware. Se mostrarán diferentes imágenes que irán acompañadas por las explicaciones que se consideren necesarias para apoyar a los planos número 1, 2 y 3 para el completo entendimiento y la completa compresión de cómo es la constitución física del robot definitivo, producto final de los pasos seguidos y explicados a lo largo del presente documento, tanto para su futura utilización por parte de un usuario, como para posibles modificaciones que se quieran realizar.

2.9.1.1.- Lateral derecho

En la siguiente imagen podrá observarse el lateral derecho del robot. Rodeada se encuentra la conexión entra la batería y el resto de circuitería. Se especifica esta conexión porque para quitar la batería es preciso desconectarla. En la imagen también se señala, pero en este caso solo con una flecha, el conector de la batería al cargador.

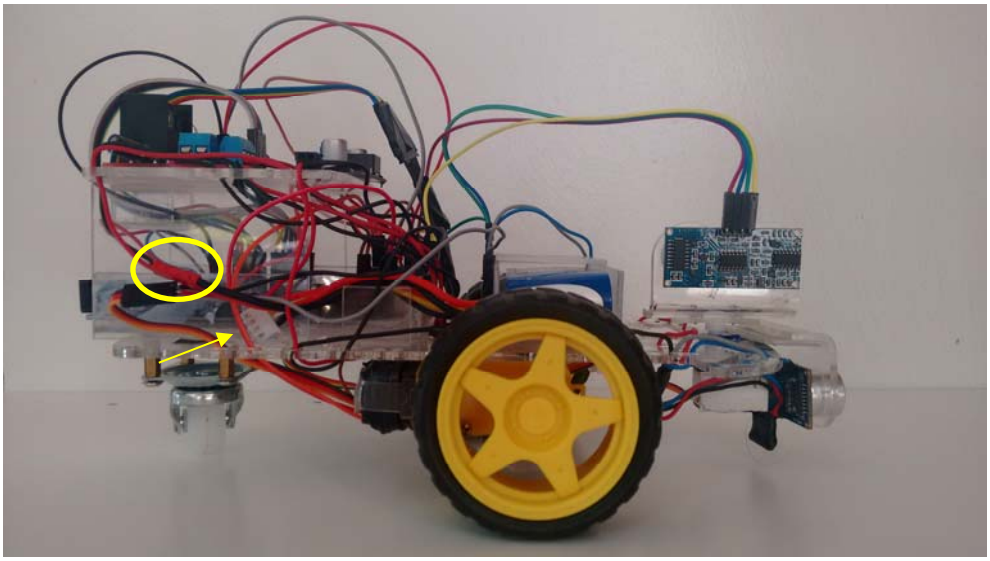


Figura 2.9.1.- Lateral derecho.

2.9.1.2.- Lateral izquierdo

En la siguiente imagen podrá observarse el lateral izquierdo del robot. Rodeado en amarillo puede observarse el conector de la pantalla LCD y el punto de testeo de voltaje para controlar la salida del regulador de tensión.

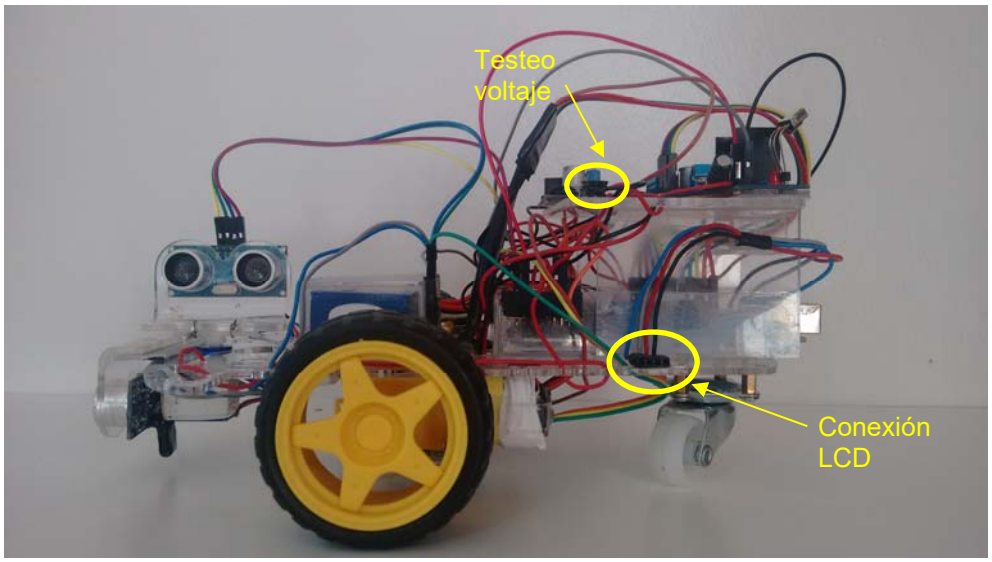


Figura 2.9.2.- Lateral izquierdo.

A continuación se recogen las conexiones del LCD y el conector:

- Cable azul → SCL.
- Cable gris → SDA.
- Cable rojo → VCC.
- Cable negro → GND.

2.9.1.3.- Parte trasera

En la siguiente imagen podrá observarse la parte trasera del robot.

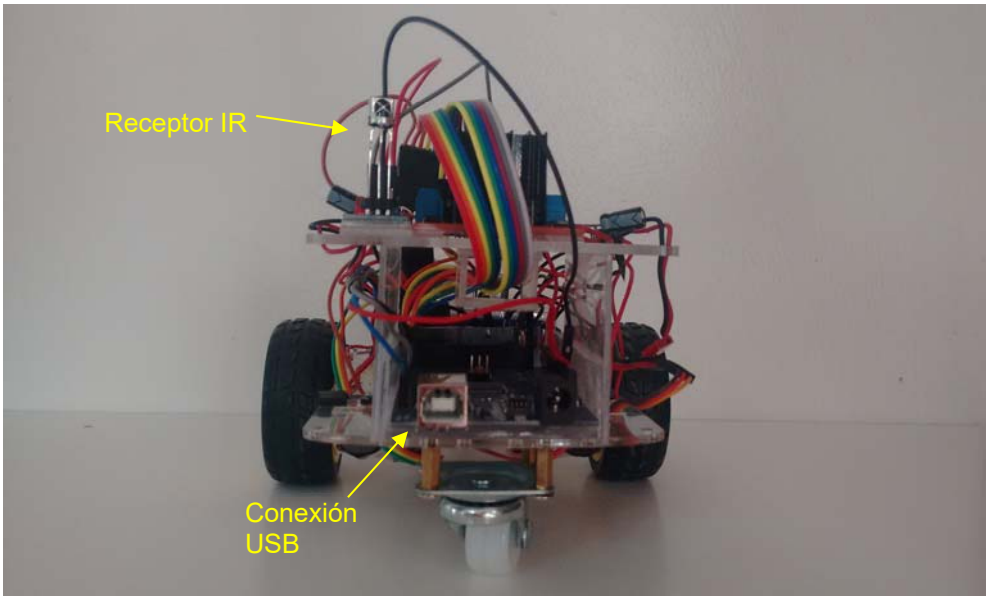


Figura 2.9.3.- Parte trasera.

2.9.1.3.- Parte delantera

En la siguiente imagen podrá observarse la parte delantera del robot. En ella se especifican los dos puntos de testeo de voltaje con los que cuenta el robot, uno para comprobar el voltaje de salida de la batería y otro para comprobar el voltaje de salida del regulador de tensión.

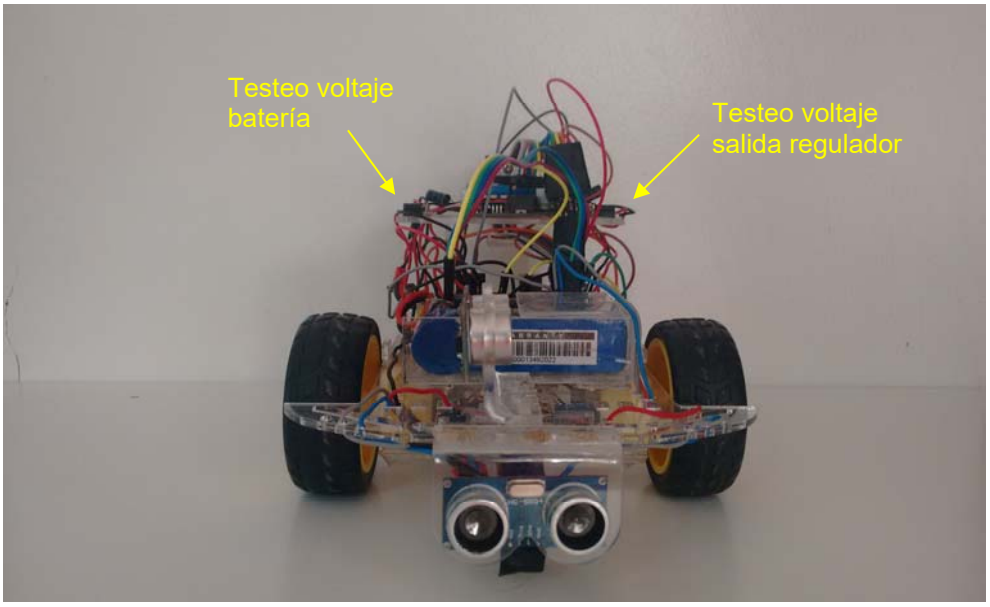


Figura 2.9.4.- Parte delantera.

2.9.1.3.- Parte inferior

En la siguiente imagen podrá observarse la parte inferior del robot. Pueden observarse en ella los 2 opto interruptores y los dos sensores IR.

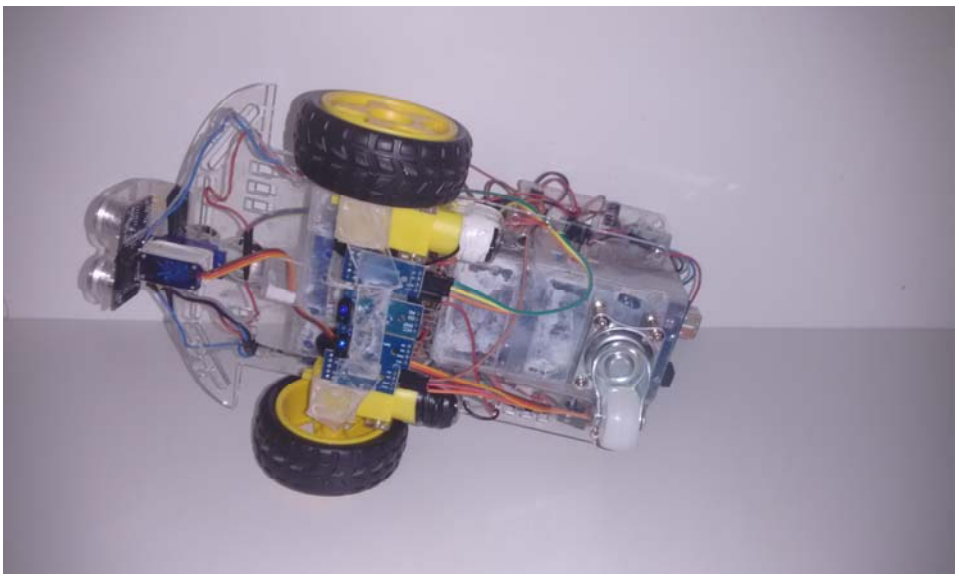


Figura 2.9.5.- Parte inferior.

2.9.1.3.- Parte superior

En la siguiente imagen podrá observarse la parte superior del robot.

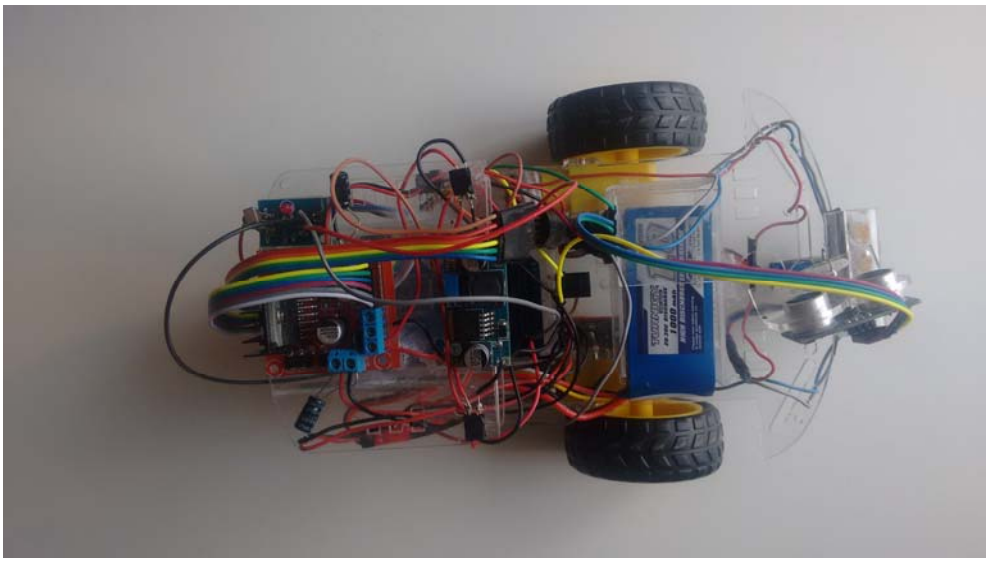


Figura 2.9.6.- Parte superior.

2.9.2.- Programación

La programación recoge todos aquellos códigos necesarios para que el robot funcione correctamente y pueda llevar a cabo las funciones descritas en la guía de usuario de los anexos. La programación del robot se estructurará en 3 partes bien diferenciadas; el control del robot, la interacción con el usuario y la extracción de los códigos de funcionamiento de un mando a distancia.

2.9.2.1.- Control del robot

En este apartado se explicará el código principal del proyecto, que es el encargado de hacer funcionar a los diferentes componentes que conforman al robot para que este funcione correctamente y alcance sus objetivos. El Aunque no esté desarrollado en ese orden exactamente, se explicará el código siguiendo un orden diferente que se considera el adecuado para su mejor entendimiento.

2.9.2.1.1.- Introducción de datos usuario

En primer lugar el usuario debe escribir en el código la casilla en la que se encuentra la entrada del laberinto tal y como se le indica en el propio código. Por defecto se establecen la fila 1 y la columna 1. Si la entrada se encuentra en un lateral, como sucede en el laberinto construido, no es necesario cambiarla para que el robot cumpla su cometido ni para que se guarde el recorrido realizado para alcanzarlo, pero se ofrece la posibilidad de cambiarla por si el usuario considera conveniente tener un criterio de direcciones.

```
//INTRODUCCIÓN DE DATOS POR PARTE DEL USUARIO
//Introducir casilla de entrada del laberinto en fila=_ y columna_:
int fila=1;
int columna=1;
```

2.9.2.1.2.- Declaraciones

Declaración de los pines, las librerías y las variables que se van a utilizar en el programa.

2.9.2.1.3.- Inicialización

Se inicializan los valores deseados antes de que comience a funcionar el programa. Incluye la inicialización del puerto serie, la configuración de los pines de Arduino, la configuración de interrupciones y la inicialización de parámetros tales como la posición inicial del servomotor, el vector que almacena el recorrido a la posición de entrada y su equivalente en la EEPROM.

2.9.2.1.4.- Control remoto

Para llevar a cabo el control remoto se llevan a cabo los siguientes pasos:

- Declaración de la librería IRremote.

```
#include <IRremote.h>
```

- Declaración del pin encargado de la recepción de datos provenientes del receptor de infrarrojos.

```
const int RECV_PIN = 52;
IRrecv irrecv(RECV_PIN);
```

- Declaración de que se usará un decodificador para leer los datos del receptor e inicialización del receptor.

```
decode_results results;
irrecv.enableIRIn();
```

- Condición de inicialización de la ejecución del programa de control del robot.

```
  while (empezar==0)
  {
     if (irrecv.decode(&results))// si existen datos recibidos
     {
        codigo=results.value;
        if (codigo==3772793023)
        {
          empezar=1;
        }
     }
  }
```

2.9.2.1.5.- Control del servomotor

Para llevar a cabo el control del servomotor se llevan a cabo los siguientes pasos:

- Declaración de la librería para controlar el servo.

```
#include <Servo.h>//Librería control servomotor
```

- Creación una instancia a objeto, un objeto Servo con el nombre servomotor, siguiendo las ideas de la programación orientada a objetos (OOP).

```
Servo servoMotor;
```

- Informar al Arduino de que en el objeto abstracto creado en la línea anterior se conectará un pin físico PWM para controlarlo.

```
servoMotor.attach(8);/*configuración del pin 8 como entrada PWM del
servo*/
```

- Control del servo por ángulos, desde 0 a 180º en sentido anti horario y por ese orden, de 0 a 180. Con 0º el sensor de ultrasonidos solidario al servo se situará recto mirando a a derecha y con 180º a la izquierda.

```
servoMotor.write(180);
```

2.9.2.1.6.- Control sensores ultrasonidos

Para llevar a cabo el control de los sensores de ultrasonidos se llevan a cabo los siguientes pasos:

- Declaración de la librería para controlar el servo.

```
#include <Servo.h>//Librería control servomotor
```

- Configuración de pines.

```
  pinMode(10, OUTPUT); /*activavión del pin 10 como salida para el
pulso ultrasónico*/
  pinMode(9, INPUT); /*activavión del pin 9 como entrada para el
pulso ultrasónico*/
```

```
  pinMode(12, OUTPUT); /*activavión del pin 12 como salida para el
pulso ultrasónico*/
  pinMode(11, INPUT);/*activavión del pin 11 como entrada para el
pulso ultrasónico*/
```

- Medida de distancia frontal por medio del Sensor fijo que se sitúa la parte delantera de la estructura del robot como se puede apreciar en el plano número 3 . En el plano 6 puede verse sus conexiones.

```
      digitalWrite(10,LOW); /* Por cuestión de estabilización del
sensor*/
      delayMicroseconds(5);
      digitalWrite(10, HIGH); /* envío del pulso ultrasónico*/
      delayMicroseconds(10);
      tiempo=pulseIn(9, HIGH); /* Función para medir la longitud del
pulso entrante. Mide el tiempo que transcurrido entre el envÃƒÂo
del pulso ultrasónico y cuando el sensor recibe el rebote, es decir:
desde que el pin 12 empieza a recibir el rebote, HIGH, hasta
que deja de hacerlo, LOW, la longitud del pulso entrante*/
```

- Medida de distancia lateral. Sensor solidario al servomotor cuya situación en el robot puede verse también en el plano número 3. En el plano 6 puede verse sus conexiones.

```
      digitalWrite(12,LOW); /* Por cuestión de estabilización del
sensor*/
      delayMicroseconds(5);
      digitalWrite(12, HIGH); /* envío del pulso ultrasónico*/
      delayMicroseconds(10);
      tiempo=pulseIn(11, HIGH); /* Función para medir la longitud
del pulso entrante. Mide el tiempo que transcurrido entre el envÃƒÂo
del pulso ultrasónico y cuando el sensor recibe el rebote, es decir:
desde que el pin 12 empieza a recibir el rebote, HIGH, hasta
que deja de hacerlo, LOW, la longitud del pulso entrante*/
```

Las mediciones anteriores solamente se realizan mientras el robot está

avanzando frontalmente, ya que se trata de la función más crítica del robot, pues requiere que esto se haga lo más rápido posible, cercano al tiempo real, para así responder rápidamente a la detección bien sea de desviaciones con respecto a la pared lateral para asegurar el avance por las calles o bien la detección de un obstáculo frontal. En caso de utilizarse funciones cada medición lleva más tiempo de procesamiento ya que las llamadas a función consumen tiempo de procesamiento. En la medición lateral el error se acumula y el avance empeora mientras que la detección frontal se retrasa y podría hacer que el robot chocara. Esto no solo lo dice la teoría, sino que se comprobó que pasaba en la práctica y los resultados eran visibles.

En el resto de mediciones que se realizan en el programa se utiliza una llamada a función para optimizar el código.

- Llamadas a función para medición de distancia.

```
distancia_lateral= medir_distancia(12,11);// Distancia lateral.
distancia= medir_distancia(10,9); //Distancia frontal.
```

- Función que mide la distancia. La función requiere del paso del valor de los pines de Echo y Trig y devolverá el resultado obtenido de la medición.

```
long medir_distancia(int pin_trig, int pin_echo)
{
  digitalWrite(pin_trig,LOW); /* Por cuestión de estabilización del
sensor*/
  delayMicroseconds(5);
  digitalWrite(pin_trig, HIGH); /* envío del pulso ultrasónico*/
  delayMicroseconds(10);
  tiempo=pulseIn(pin_echo, HIGH); /* Función para medir la longitud
del pulso entrante. Mide el tiempo que transcurrido entre el envÃƒÂo
del pulso ultrasónico y cuando el sensor recibe el rebote, es decir:
desde que el pin 12 empieza a recibir el rebote, HIGH, hasta que
 deja de hacerlo, LOW, la longitud del pulso entrante*/
   distancia= int(0.017*tiempo);
   return distancia;
```

```
}
```

2.9.2.1.7- Control de motores

A continuación se expondrán las órdenes necesarias para el control de los motores del robot. Las conexiones del controlador de motores con el Arduino y con los propios motores pueden verse en el plano número 6.

- Configuración de los pines de Arduino.

```
int motorDerAdelante=3;//El pin 3 a In1 del L298N
int motorDerAtras=4;//El pin 4 a In2 del L298N
int motorIzqAdelante=5;//El pin 5 a In3 del L298N
int motorIzqAtras=6;//El pin 6 a In4 del L298N
int VelDerecho=2;  //El pin 2 a EnA del L298N
int VelIzquierdo=7;//El pin 7 aEnB del L298N

pinMode(motorDerAdelante, OUTPUT);
pinMode(motorDerAtras, OUTPUT);
pinMode(motorIzqAdelante, OUTPUT);
pinMode(motorIzqAtras, OUTPUT);
pinMode(VelDerecho, OUTPUT);
pinMode(VelIzquierdo, OUTPUT);
```

- Llamadas a función para el control de motores. Tras cada llamada es preciso enviar a través del pin PWM de Arduino al correspondiente del controlador de motores la velocidad del motor. Estas conexiones pueden consultarse en el plano número 6, donde podrán consultarse el resto de pines que se comentarán a partir de este punto.

```
girar_derecha();
analogWrite(VelDerecho,40);
analogWrite(VelIzquierdo,40);

girar_izquierda();
analogWrite(VelDerecho,20);
analogWrite(VelIzquierdo,20);
```

```
robot_adelante();
analogWrite(VelDerecho,20);
analogWrite(VelIzquierdo,20);

robot_atras();
analogWrite(VelDerecho,20);
analogWrite(VelIzquierdo,20);

frenar_robot();
```

- Funciones control de los motores.

```
void robot_adelante()
{
  digitalWrite(motorDerAdelante,HIGH);
  digitalWrite(motorDerAtras,LOW);
  digitalWrite(motorIzqAdelante,HIGH);
  digitalWrite(motorIzqAtras,LOW);
}

void girar_derecha()
{
  digitalWrite(motorDerAdelante,LOW);
  digitalWrite(motorDerAtras,HIGH);
  digitalWrite(motorIzqAdelante,HIGH);
  digitalWrite(motorIzqAtras,LOW);
}

void girar_izquierda()
{
  digitalWrite(motorDerAdelante,HIGH);
  digitalWrite(motorDerAtras,LOW);
  digitalWrite(motorIzqAdelante,LOW);
  digitalWrite(motorIzqAtras,HIGH);
}

void frenar_robot()
{
  digitalWrite(motorDerAdelante,LOW);
  digitalWrite(motorDerAtras,LOW);
```

```
  digitalWrite(motorIzqAdelante,LOW);
  digitalWrite(motorIzqAtras,LOW);
  analogWrite(VelDerecho,0);
  analogWrite(VelIzquierdo,0);
}

void robot_atras()
{
  digitalWrite(motorDerAdelante,LOW);
  digitalWrite(motorDerAtras,HIGH);
  digitalWrite(motorIzqAdelante,LOW);
  digitalWrite(motorIzqAtras,HIGH);
}
```

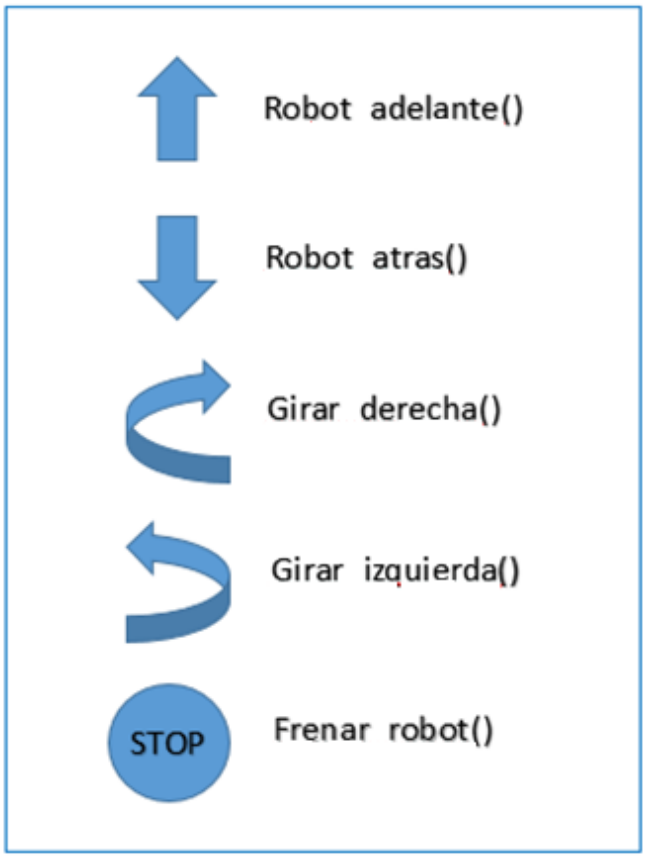


Figura 2.9.2.1.7.1.-Funciones control de motores.

2.9.2.1.8.- Control del recorrido

Para el control del recorrido del robot a lo largo de las filas y columnas que forman la matriz del laberinto se utilizan dos optointerruptores, colocados en cada rueda, una interrupción de Arduino para cada uno y dos funciones. A continuación se describirán los apartados del programa encargados del control del recorrido.

- Configuración de las interrupciones. Se definen los pines que trabajarán como entrada de la interrupción y la función a la que llamarán cuando se produzca.

```
pinMode(18 , INPUT);  //definir pin como entrada
attachInterrupt(5, calculo_direccion, RISING) ;
pinMode(19 , INPUT);  //definir pin como entrada
attachInterrupt(4, calculo_direccion, RISING) ;
```

- Cada vez que se produzca una interrupción provocada por los optointerruptores, es decir, que se detecte una ranura del encoder, se hará una llamada a la función calculo_direccion.

Dentro de la función calculo_direccion se utilizan las siguientes variables:

```
static volatile unsigned long debounce1 = 0; // Tiempo del rebote
encoder izquierdo.
static volatile unsigned long debounce2 = 0; // Tiempo del rebote
encoder izquierdo.
volatile int contador_izq = 0;
volatile int contador_dcha = 0;
int direccion_desplazamiento=1;
bool laberinto[6][6];
long camino_recorrido;
bool giro_efectuado=0;
```

En la siguiente tabla se indicará la función que tendrá cada una de las variables utilizadas en esta función:

Variable	Función
Debounce1	Almacena el tiempo en el que se produce una interruptción del optointerrptor de la izquierda.
Debounce2	Almacena el tiempo en el que se produce una interruptción del optointerruptor de la derecha.
Contador_izq	Almacena el número de veces que se produce una interrupción del optointerrptor de la izquierda.
Contador_dcha	Almacena el número de veces que se produce una interrupción del optointerrptor de la derecha
Laberinto[6][6]	Matriz que representa las 36 casillas sobre las que está estrucurado el laberinto.
Camino_recorrido	Variable entera que almacena en bruto la posición del robot. Las filas serán las decenas y las columnas las unidades.
Giro_efectuado	Variable booleana que indica si se ha efectuado un giro.
Dirección desplazamiento	Almacena la dirección de desplazamiento del robot.

Tabla 2.9.2.1.8.1.- Función variables calculo_direccion()

En cuanto a la variable dirección_desplazamiento se comentó en la tabla anterior que almacena la dirección de desplazamiento del robot. Para la programación del código se establece un criterio de direcciones en el cual se diferencian 4 direcciones diferentes. Las direcciones no son fijas, es decir, por ejemplo, la dirección de desplazamiento 1 no siempre apunta al norte, si no que depende de la entrada del laberinto. La dirección desplazamiento1 siempre será la longitudinal a la entrada del laberinto. A continuación pueden verse los valores de las direcciones para un laberinto como el siguiente:

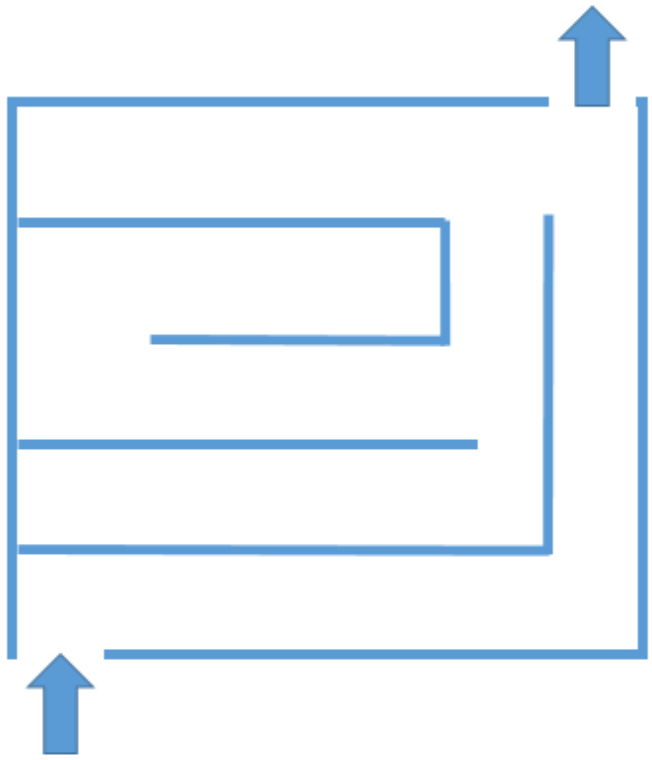


Figura 2.9.2.1.8.1.- Ejemplo de laberinto.

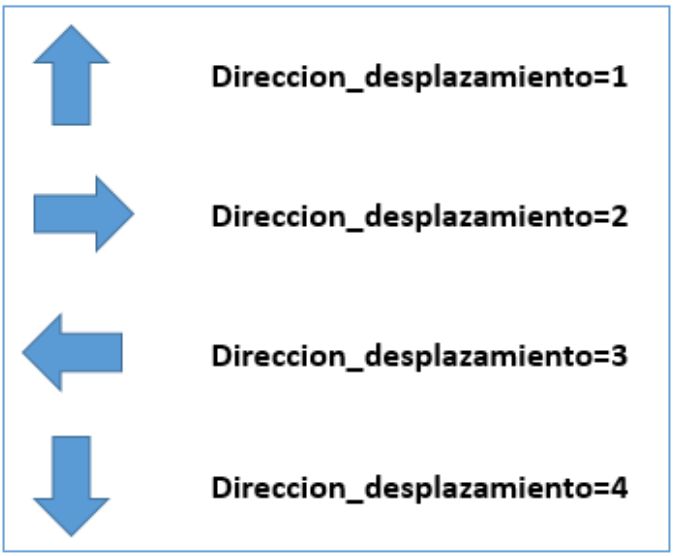


Figura 2.9.2.1.8.2.- Direcciones de desplazamiento.

A continuación se mostrará la función al completo:

```
void calculo_direccion(){

    if (digitalRead (encoder1) && (micros()-debounce1 > 20000) &&
digitalRead (encoder1) && contador_izq<35 && caso==1) {
    // Vuelve a comprobar que el encoder envia una señal buena y
luego comprueba que el tiempo es superior a 20000 microsegundos y
vuelve a comprobar que la señal es correcta.
        debounce1 = micros(); // Almacena el tiempo para comprobar
que no se cuenta el rebote que hay en la señal.
        contador_izq++;
    }  // Suma el pulso bueno que entra.
    if (digitalRead (encoder2) && (micros()-debounce2 > 20000) &&
digitalRead (encoder2) && contador_dcha<35 && caso==1) {
    // Vuelve a comprobar que el encoder envia una señal buena y
luego comprueba que el tiempo es superior a 20000 microsegundos y
vuelve a comprobar que la señal es correcta.
        debounce2 = micros(); // Almacena el tiempo para comprobar
que se cuenta el rebote que hay en la señal.
        contador_dcha++;
    }
```

```
    else if((contador_izq>=35 &&
contador_dcha>=35)||(giro_efectuado==1)){
        contador_izq=0;
        contador_dcha=0;
        giro_efectuado=0;

  //renueva matriz
        switch(direccion_desplazamiento)
        {
          case 1:
            columna=columna+1;
            laberinto[fila][columna]=1;
          break;

          case 2:
            fila=fila+1;
            laberinto[fila][columna]=1;
          break;

          case 3:
            fila=fila-1;
            laberinto[fila][columna]=1;
          break;

          case 4:
             columna=columna-1;
             laberinto[fila][columna]=1;
          break;
        }
        direccion=direccion+1;
        camino_recorrido=(fila*10)+columna;
        EEPROM.write(direccion,camino_recorrido);
    }
}
```

Esta función se estructura en tres condiciones:

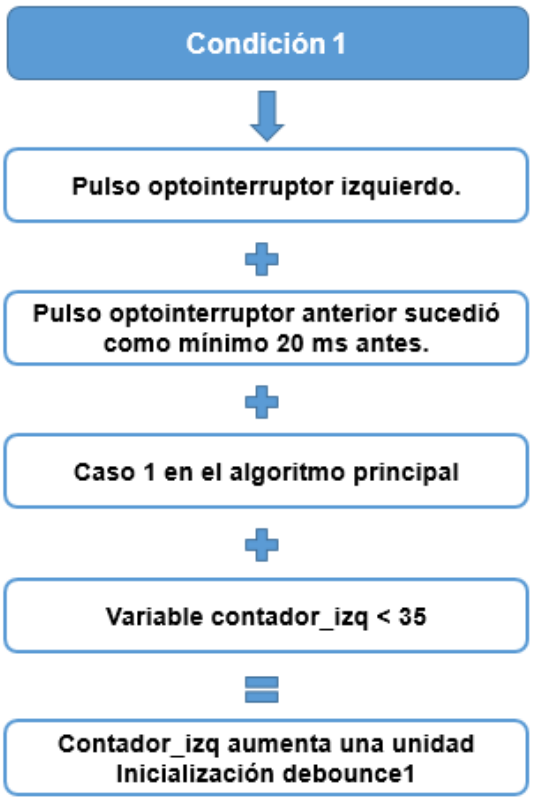


Figura 2.9.2.1.8.3.- Condición 1 calculo_direccion().

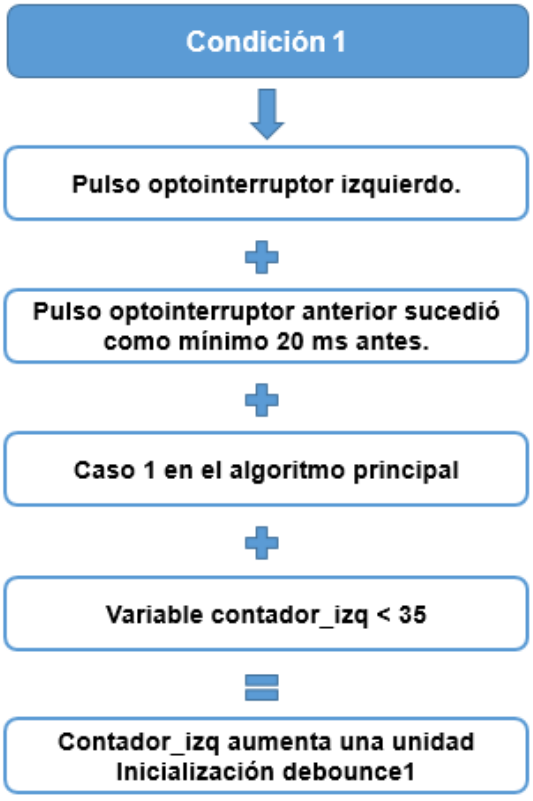


Figura 2.9.2.1.8.4.- Condición 2 calculo_direccion()

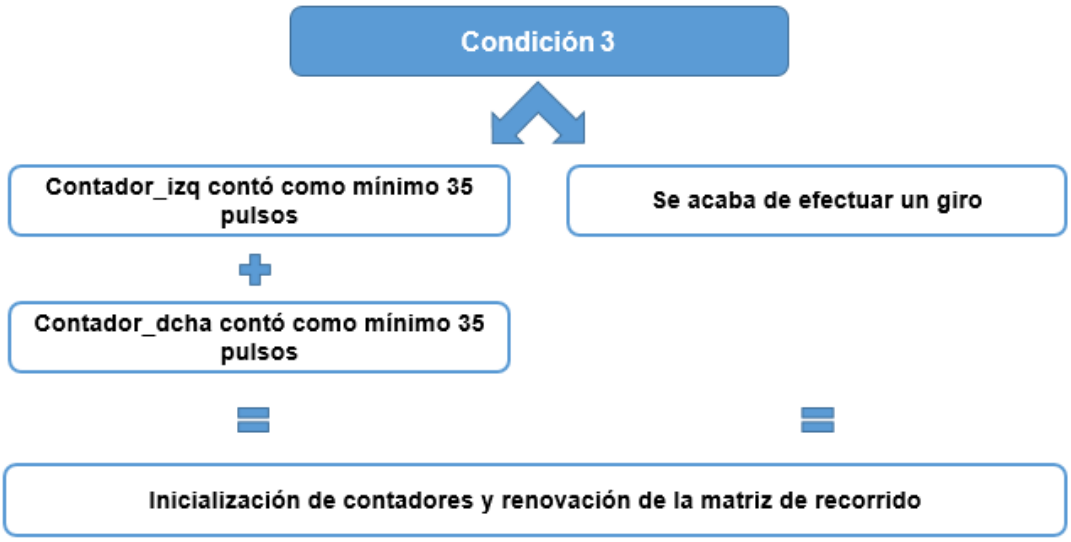


Figura 2.9.2.1.8.5.- Condición 3 calculo_direccion().

- Cada vez que se produzca un cambio de dirección y este se efectúe físicamente, mediante la ejecución de las instrucciones correspondientes en cada caso, contenidas en la parte principal del código, se seguirá siempre la misma secuencia de instrucciones:

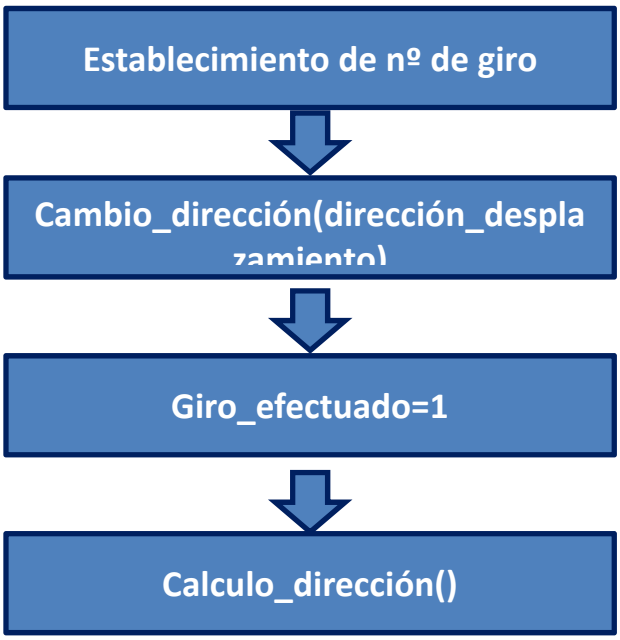


Figura 2.9.2.1.8.6.-Secuencia de instruccuiones tras cambio de dirección.

En este momento aparecen en juego nuevas variables:

```
int giro;
int direccion_anterior=1;
```

Variable	Función
Giro	Almacena el número de giro que se ha llevado a cabo.
Direccion_anterior	Almacena el valor de la variable dirección_desplazamiento después de haberse ejecutado un giro.

Tabla 2.9.2.1.8.2.- Función variables

En la secuencia anterior en primer lugar se produce el establecimiento del número de giro que se lleva a cabo. Se diferencian 3 casos de giros diferentes:

Nº giro	Función
Giro = 2	Giro a la derecha.
Giro = 3	Giro a la izquierda.
Giro = 4	Cambio de sentido.

Tabla 2.9.2.1.8.3.- Casos de giro.

Tras establecerse el número de giro se produce el llamamiento a función cambio_dirección(), a la cual se le pasa por valor la variable dirección_desplazamiento, tal como se muestra a continuación:

```
cambio_direccion(direccion_desplazamiento);
```

La función llamada tiene la función de establecer la nueva dirección de desplazamiento del robot, es decir, de la variable dirección_desplazamiento, tras

haberse efectuado un giro. La nueva dirección de desplazamiento dependerá del valor de la variable giro. En la siguiente tabla podrán verse los nuevos valores de la dirección de desplazamiento en función del número de giro y de la dirección de desplazamiento anterior:

Nº giro	Direccion 1	Direccion 2	Direccion 3	Direccion 4
Giro = 2	2	4	1	3
Giro = 3	3	1	4	2
Giro = 4	4	3	2	1

Tabla 2.9.2.1.8.4- Cálculo nueva dirección desplazamiento.

A continuación podrá verse el código de esta función:

```
void cambio_direccion(int direccion_anterior)
{
   contador_izq=0;
   contador_dcha=0;
   switch(giro)
   {
      case 2:
        if (direccion_anterior==1)
        {
          direccion_desplazamiento=2;
        }
        else if (direccion_anterior==2)
        {
         direccion_desplazamiento=4;
        }
        else if (direccion_anterior==3)
        {
         direccion_desplazamiento=1;
        }
        else if (direccion_anterior==4)
        {
         direccion_desplazamiento=3;
```

```
        }
      break;
      case 3:
        if (direccion_anterior==1)
        {
          direccion_desplazamiento=3;
        }
        else if (direccion_anterior==2)
        {
         direccion_desplazamiento=1;
        }
        else if (direccion_anterior==3)
        {
         direccion_desplazamiento=4;
        }
        else if (direccion_anterior==4)
        {
         direccion_desplazamiento=2;
        }
      break;
      case 4:
        if (direccion_anterior==1)
        {
          direccion_desplazamiento=4;
        }
        else if (direccion_anterior==2)
        {
         direccion_desplazamiento=3;
        }
        else if (direccion_anterior==3)
        {
         direccion_desplazamiento=2;
        }
        else if (direccion_anterior==4)
        {
         direccion_desplazamiento=1;
        }
      break;
  }
```

Una vez que se termina la función anterior se pone a uno la variable giro_efectuado.

```
giro_efectuado=1;
```

Por último se produce una llamada a la función calculo_direccion(), de la que ya se habló anteriormente, ya que también es llamada cuando se producen las interrupciones correspondientes a los dos optointerruptores. Pero no se habló de la acción correspondiente a la condición 3 de dicha función, que es la encargada de renovar la matriz de recorrido.

Para renovar la matriz que almacena el recorrido se utiliza el siguiente bloque de funciones:

```
switch(direccion_desplazamiento)
        {
          case 1:
            columna=columna+1;
            laberinto[fila][columna]=1;
          break;

          case 2:
            fila=fila+1;
            laberinto[fila][columna]=1;
          break;

          case 3:
            fila=fila-1;
            laberinto[fila][columna]=1;
          break;

          case 4:
             columna=columna-1;
             laberinto[fila][columna]=1;
          break;
        }
```

Este bloque solo se ejecutará cuando se den las condiciones de que o bien se avanzó una casilla del laberinto, o bien se produjo un cambio de dirección. El valor de la dirección de desplazamiento es el actual en caso de producirse por la primera condición o el nuevo valor de desplazamiento en caso de producirse por un cambio de dirección. Dependiendo de cuál sea ese valor se actualizará la matriz que guarda la posición del robot.

Una vez que se renueva la matriz se guardará el valor de la posición en la EEPROM. En primer lugar es preciso aumentar el valor de la variable dirección para acceder a la siguiente dirección de la EEPROM. Después se realiza la conversión de la posición, de forma que el número de la fila se multiplica por 10 y al resultado se le suma el valor de la columna y, por último, se almacenará este valor en la EEPROM.

```
direccion=direccion+1;
camino_recorrido=(fila*10)+columna;
EEPROM.write(direccion,camino_recorrido);
```

2.9.2.1.9.- Filtro anti rebotes opto interruptores

En este punto se explicará el filtro implementado por software mencionado en el apartado de sensores del análisis de soluciones. El funcionamiento de este filtro se basa en una parte del código ya presentada anteriormente que se encuentra dentro de la función a la que se acude cuando se produce una interrupción de los opto interruptores.

Se trata de las dos variables debounce; debounce1 y debounce2. Estas variables solo dejan que se produzca una interrupción cada 20 ms.

Haciendo cálculos se determina que para una velocidad de 100 rpm de las ruedas del robot se producirían un total de 2000 interrupciones del encoder. Esto hace un total de 33,33 interrupciones/segundo, es decir, una interrupción cada 30 ms. En este caso, para una velocidad constante, se produciría una interrupción cada 30 ms, por lo que con una condición de que se rechacen todos aquellos

pulsos que no se produzcan dentro de este margen, evitando de esta manera el ruido.

En la aplicación diseñada la velocidad no es constante, por lo que habría que establecer un valor límite tal que se asegure que nunca se pase de él. El valor escogido es 20 ms y se asegura con total certeza que no se pasará de él.

2.9.2.1.10.- Algoritmo principal

Se denomina algoritmo principal a la parte del programa encargada de controlar, a un nivel más alto de programación, que las partes anteriores logren alcanzar el avance del robot por las calles del laberinto. Se dice que lo hace a un nivel más alto porque las funciones anteriores son llamadas desde este algoritmo de tal forma que, con la conveniente organización, se consiga el cometido deseado.

El algoritmo que rige el avance del robot por el laberinto se basa en una máquina de estados en la que se diferencian un total de 15 estados, llamados casos dentro del programa. La necesidad de utilizar una máquina de estados para la elaboración del algoritmo reside, por un lado, en la simplicidad que proporciona al código, consiguiendo una programación más clara, estructurada y sencilla, ideal para la total comprensión del código por parte del lector y, por otro lado, rapidez para la ejecución de ciertas partes del programa que requieren velocidad de ejecución. A continuación se explicará la función de cada uno de los estados, y después se mostrará el flujograma que esquematiza el funcionamiento de este algoritmo.

- Caso 1. Este estado es el encargado de controlar el avance frontal del robot a lo largo de una recta. Como ya se explicó, esto se realiza controlando los dos sensores de ultrasonidos con los que se equipa al robot, uno, el solidario al servo, controlando la distancia lateral, y el otro, el situado en la parte delantera, tal y como se puede comprobar en el plano nº 3, la distancia frontal. En función de ambos valores se controlan los motores, regulando, por un lado, la velocidad de los motes en función de la distancia lateral, para evitar chocar

con las paredes laterales y, por el otro, controlando la distancia con la pared frontal por medio del sensor delantero para evitar chocar con la misma.

En consecuencia de lo explicado se establecen una serie de condiciones. La condición con mayor preferencia es la comprobación de la distancia frontal, cuyo límite establecido es de 6cm, distancia en la que se pasa a otro estado, como se apreciará en el flujograma. A continuación se llevan a cabo una serie de condiciones encargadas de la regulación lateral, las cuales se recogen, junto a la acción que se producen cuando se cumplen, en la tabla mostrada a continuación:

Condición	Acción
Distancia lateral>46 cm	Camino libre. Cuando la distancia es mayor que 46 se considera que hay un camino libre hacia la izquierda por lo que habrá que seguir ese camino.
18<= Distancia lateral <= 23	Avance deseado. Entre esos valores se considere que el robot avanza correctamente, por lo que no han de emprenderse acciones correctoras y se harán giran los dos motores a la misma velocidad.
14<= Distancia lateral <18	Corrección desviación izquierda leve. Se reducirá la velocidad del motor derecho levemente para corregir la desviación hacia la izquierda.
10<= Distancia lateral <14	Corrección desviación izquierda moderada. Se reducirá aún más la velocidad del motor derecho.
Distancia lateral <10	Corrección desviación izquierda severa. Se detiene el avance frontal y se sale del estado 1 hacia el 13 para corregir la desviación.
23< Distancia lateral <=26	Corrección desviación derecha leve. Se reducirá la velocidad del motor izquierdo levemente para corregir la desviación hacia la

	derecha.
26< Distancia lateral <=28	Corrección desviación derecha moderada. Se reducirá aún más la velocidad del motor izquierdo.
28< Distancia lateral <=35	Corrección desviación derecha severa. Se detiene el avance frontal y se sale del estado 1 hacia el 14 para corregir la desviación.

Tabla 2.9.2.1.10.1.- Condiciones que regulan el avance del robot.

- Caso 2. Tras detectar camino libre a la izquierda o un obstáculo frontal se pasa al estado 2 en el cual se detendrá el robot. Si en el avance frontal el robot detecta que hay camino libre a la izquierda, inmediatamente la máquina de estados pasa del caso 1 al 2, el robot se detiene, y se pasa al estado correspondiente para realizar el giro a izquierdas. Lo mismo pasa si el robot detecta un obstáculo frontal, pero en este caso, la máquina de estados avanzará en otra dirección. En caso de presentarse la situación de haber camino libre a la izquierda y pared frontal, el robot siempre detectará antes el camino libre.

- Caso 3. Tras detectar pared frontal se comprueba si hay camino libre a la derecha girando el servo, ya que a la izquierda no hay y, evidentemente, hacia delante tampoco. Si hay camino libre hacia la derecha se pasará al estado 4, si no, no habrá salida, así que se pasará al estado 11.

- Caso 4. Giro a la derecha de 90º. Ante de realizar el giro a la derecha se llevarán a cabo una serie de medidas correctivas en caso de ser precisas. Para detectar la desviación se utiliza un simple algoritmo que se basa en, ua vez detenido el robot tomar una medida de la distancia lateral, hacer avanzar el robot unos centímetros y tomar otra medida de la distancia lateral. Comparándolas se determina el tipo de corrección que hay q usar o, en su defecto, si no hay que usarla. Por ejemplo, si la segunda medida tomada es 2 cm mayor que la primera, querrá decir que el robot está situado en forma

diagonal hacia la derecha, a la pared del pasillo. Se diferencian 4 tipos de correcciones:

Tipo de corrección	Acción
Corrección 1. Desviación diagonal hacia la derecha	Corrección con giro a la izquierda.
Corrección 2. Desviación diagonal hacia la izquierda.	Corrección con giro a la derecha.
Corrección 3. No se produce desviación diagonal, pero el robot está demasiado pegado al lado izquierdo.	Se llevará a cabo la siguiente secuencia de instrucciones; giro hacia la izquierda, desplazamiento hacia atrás, giro a la derecha para invertir el giro anterior y avance frontal para invertir el retroceso.
Corrección 4. No se produce desviación diagonal, pero el robot está demasiado pegado al lado derecho.	Ligero avance frontal.

Tabla 2.9.2.1.10.2.- Correcciones previas a giro a la derecha.

A continuación puede observarse el flujograma de este estado:

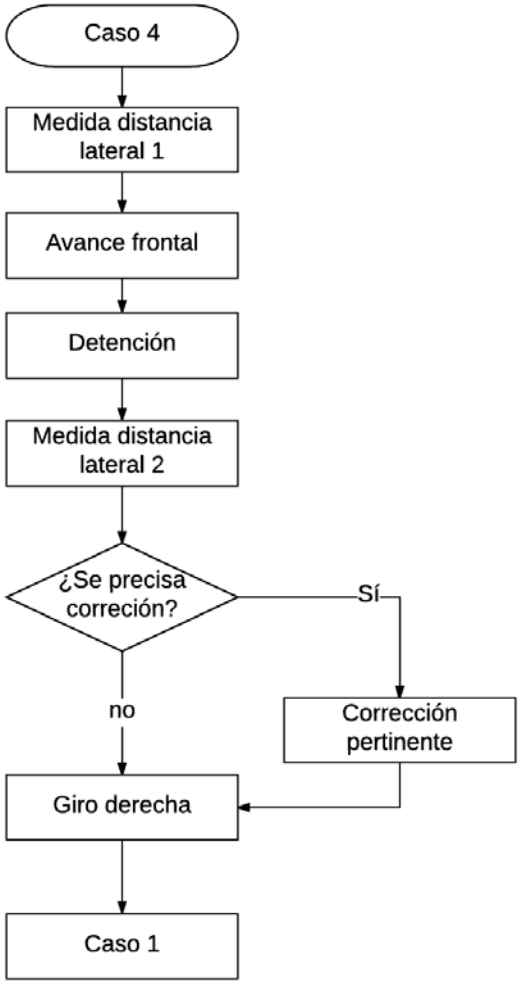


Figura 2.9.2.1.10.1.- Flujograma caso 4.

- Caso 5. Si se detectó camino libre a la izquierda el programa avanza hasta este estado. En él se comprueba que la medida tomada es correcta para descartar un posible error del sensor en la medición, o una falsa medida provocada por la inclinación del robot hacia la derecha. Para comprobarlo se realiza un ligero giro a la derecha y se toma otra muestra, para después invertir el giro realizado. Si hubo un error el robot continuará con el avance propio del caso 1, si no lo hubo comprobará si hay pared frontal o no y el programa avanzará hasta el caso 7.

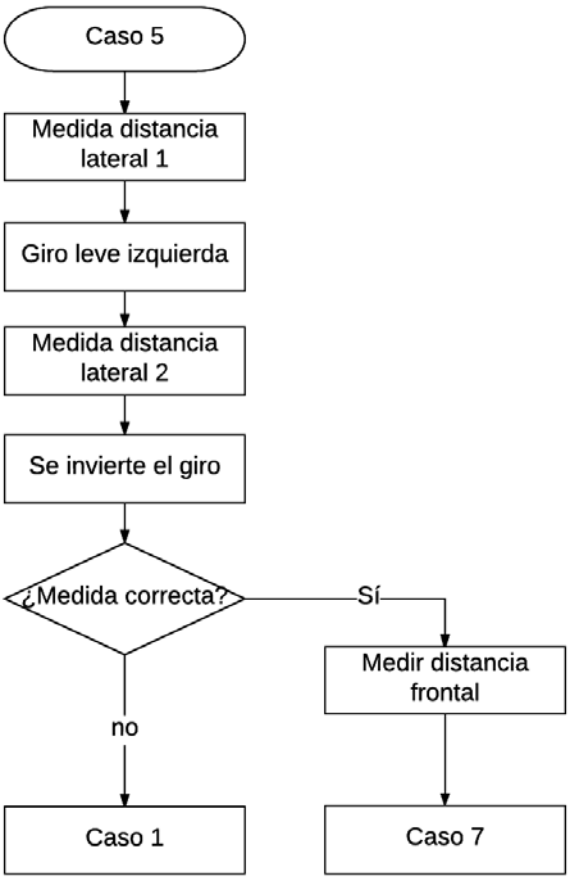


Figura 2.9.2.1.10.2.- Flujograma caso 5.

- Caso 6. Tras determinar que es preciso realizar un giro a izquierdas y que hay pared frontal, en este estado se le indica al robot que avance hasta que se detecte pared frontal a una distancia de 7 cm. Una vez que esto sucede se pasa al caso 8.

- Caso 7. Una vez que se detectó camino libre a la izquierda y se midió la distancia frontal el programa se sitúa en el estado 7 de la máquina de estados. En este estado en primer lugar se producirá una corrección de la posición, parecida a la del estado 4 pero no similar, ya que en este caso el giro se realizará a izquierdas, por lo que la regulación debe hacerse con respecto a la pared derecha. Por este motivo en primer lugar se gira el servo para 180º.

Una vez que el sensor solidario al servo apunta a la pared derecha se sigue el mismo procedimiento que en el caso 4. Se hace avanzar al robot y se toman medidas de la distancia lateral antes y después del movimiento. Con estas dos medidas se establecerán una serie de condiciones encargadas de corregir la posición del robot, si es necesario, para que se realice el giro correctamente. Estas condiciones son diferentes a las existentes en el caso 4, ya que la situación es diferente. A continuación se expondrán los tipos de correcciones:

Tipo de corrección	Acción
Corrección 1. Desviación diagonal hacia la izquierda	Corrección con giro a la derecha.
Corrección 2. Desviación diagonal hacia la derecha y no se encuentra los suficientemente pegado a la pared derecha como para chocar con ella al realizar el giro hacia la izquierda.	Corrección con giro a la izquierda.
Corrección 3. El robot está pegado a la pared derecha esté como esté colocado.	Se llevará a cabo la siguiente secuencia de instrucciones; giro hacia la derecha, desplazamiento hacia atrás, giro a la derecha para invertir el giro anterior y avance frontal para invertir el retroceso.

Tabla 2.9.2.1.10.3.- Correcciones caso 7.

Una vez que el robot se realiza la corrección anterior, la pared derecha ya no supondría un problema para realizar el giro, pero si la izquierda, por lo que tendrá que avanzar para evitar chocar con ella. El avance dependerá de si hay pared frontal o no. Si hay pared frontal se pasará al estado 6. De no haberla el robot avanzará un determinado tiempo establecido, en su momento, tras haber realizado las correspondientes pruebas. A la par que se avanza para evitar la pared izquierda también se hará una regulación para corregir la desviación del mismo.

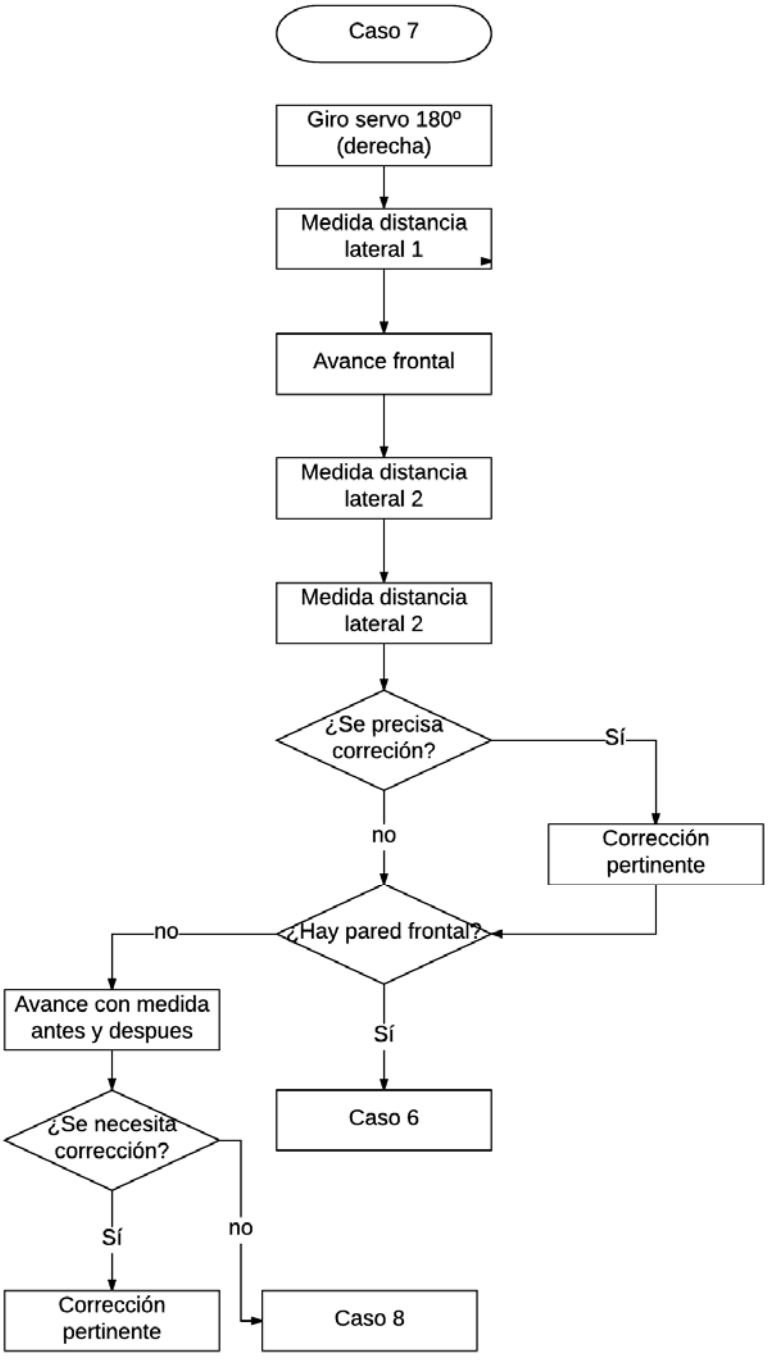


Figura 2.9.2.1.10.3.- Flujograma caso 7.

- Caso 8. Como en estados anteriores se preparó al robot para colocarlo en una posición en la que pudiera realizar un giro a izquierdas de 90º correctamente, en este estado simplemente se realizará el giro a izquierdas.

- Caso 9. Tras producirse el giro a izquierdas, el robot debe avanzar para conocer cuál es su situación tras el giro, es decir, que acción tendrá que emprender. En este estado simplemente se realiza el citado avance.

- Caso 10. Tras girar a la izquierda y avanzar, se deberá determinar cuál es el siguiente paso que tiene que dar el robot para continuar avanzando a través de las calles del laberinto. Para ello habrá que devolver al sensor de ultrasonidos solidario al servo a su posición natural, apuntando a la izquierda. Tras esto se realizará una medida de la distancia con el mencionado sensor y dependiendo del valor de esta se actuará en consecuencia. Se pueden dar 3 casos diferentes:

Tipo de corrección	Acción
Caso 1. No hay obstáculos ni a la izquierda ni a la derecha	Salida del laberinto. Fin del programa.
Caso 2. Camino libre hacia la izquierda	Caso 6 si hay pared frontal o caso 9 si no la hay.
Caso 3.	Caso 1.

Tabla 2.9.2.1.10.4.- Posibilidades dentro del caso 10.

- Caso 11. Tras comprobar que no hay ni camino libre a la izquierda, ni camino libre hacia delante, ni hacia la derecha, el robot deberá hacer un cambio de sentido, es decir, un giro de 180º. Como en los casos anteriores, siempre que se realiza un giro, antes habrá que corregir la posición del robot. En este caso la corrección es similar a la descrita en el estado 4, ya que la regulación también se hace respecto a la pared izquierda, exceptuando que la corrección número 4 de dicho estado, ya que en el mismo, cuando el robot se encontraba alejado de la pared izquierda más de 22 cm (solamente habría 4 cm entre la rueda derecha y la pared derecha) no había que hacer ningún tipo de corrección, pero ahora sí, ya que si no el robot chocaría.

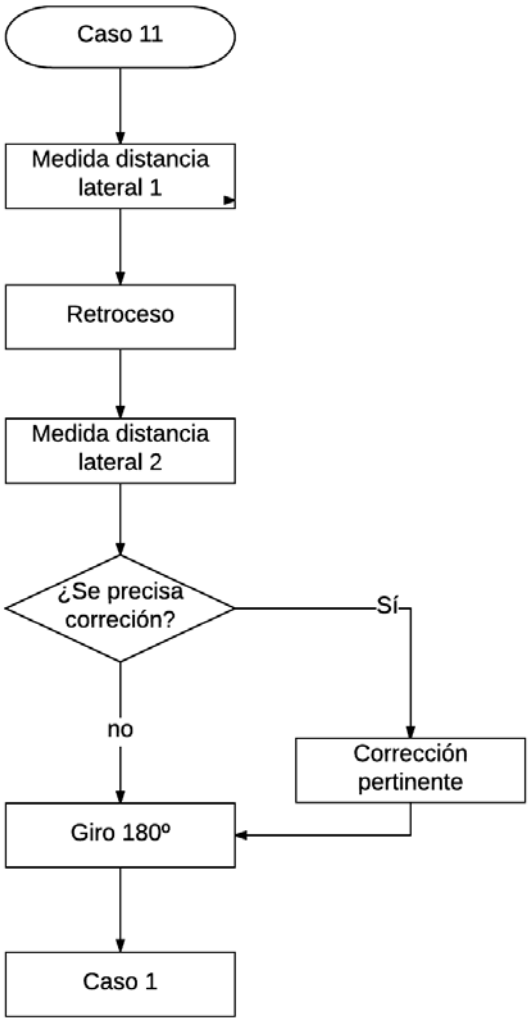


Figura 2.9.2.1.10.4.- Flujograma caso 11.

- Casos 12 y 13. Ambos casos son llamados desde el caso 1 como se indicó en el punto respectivo a ese caso. En ambos se realiza una secuencia de instrucciones cuya función es apartar al robot de la pared lateral a la que se encuentren cercana para evitar que choque contra la misma y situarlo lo más centrado posible. Si el robot supera el límite de distancia establecido, entonces el robot frenará. Cuando eso suceda el robot siempre estará girado hacia la pared, es decir, la parte delantera estará apuntando a la pared. Para que no choque, después de frenar el robot retrocederá unos centímetros, para después girar unos grados, en el caso 12 hacia la izquierda y en 13 hacia la derecha. Por último avanzará brevemente para después volver al caso 1 y continuar con el avance.

Por último, en este apartado, se mostrará el flujograma general del algoritmo de control del desplazamiento del robot.

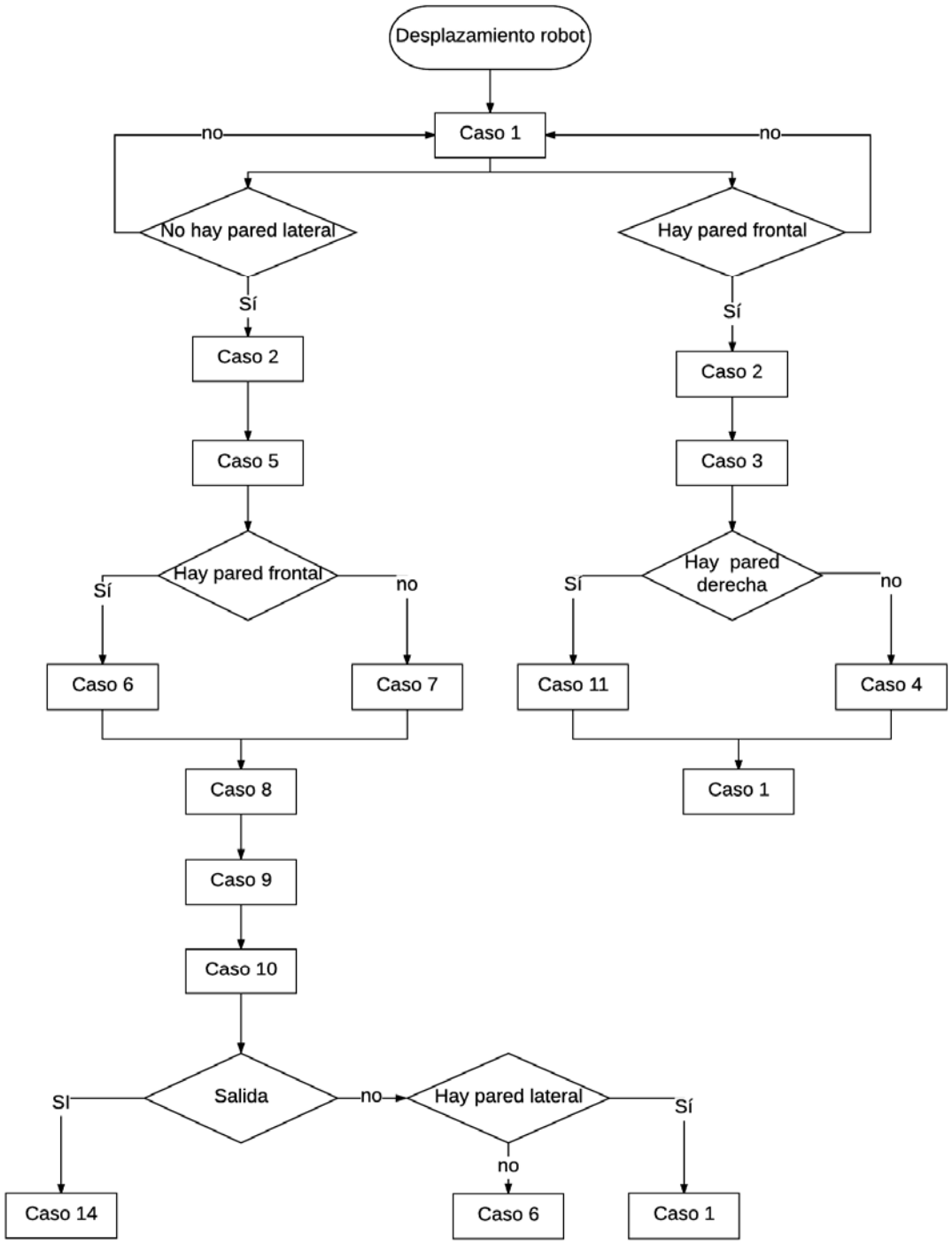


Figura 2.9.2.1.10.5.- Flujograma general desplazamiento robot.

2.9.2.2.- Interacción con el usuario

En este apartado se explicará la parte de código relacionada con la interacción de la aplicación con el usuario. Como ya se ha mencionado, se proporcionará la posibilidad de conectar una pantalla LCD para que el usuario pueda realizar diferentes acciones, manejando el sistema con un mando a distancia.

Como en la explicación del programa que controla el funcionamiento del robot, en este caso, también se tratará cada parte del programa por separado.

2.9.2.2.1.- LCD

Para utilizar el LCD será preciso llevar a cabo una serie de pasos.

- Declaración de librerías. El primer paso es, como siempre, la declaración de las librerías que controlan el LCD y la comunicación I2C.

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>
```

- Especificar la dimensión de la pantalla LCD.

```
LiquidCrystal_I2C  lcd(0x27,16,2); // especifica la dimensión de la
pantalla
```

- Creación de caracteres que no están definidos en la librería del LCD y que se necesitarán para representar en la pantalla. La pantalla del LCD contará con 2 filas y 16 columnas de caracteres de, a su vez, 8 filas de 5 columnas de pixeles. De esta forma para crear un carácter habrá que rellenar los pixeles de la estructura con unos y ceros, que luego, gracias a la librería Liquidcrystal, se podrá representar en un carácter de la pantalla. Los caracteres creados se tratarán de una flecha apuntando hacia la derecha y otra hacia la izquierda.

```
//creación de caracteres
byte flecha_dcha[8] = {
  B00000,
  B11000,
  B11100,
  B11110,
  B11111,
  B11110,
  B11100,
  B11000,
};

byte flecha_izq[8] = {
  B00000,
  B00011,
  B00111,
  B01111,
  B11111,
  B01111,
  B00111,
  B00011,
};
```

Una vez creadas las estructuras se procederá a asignarlas a un carácter para luego poder utilizarlas a través de la librería del LCD.

```
 lcd.createChar (0,flecha_dcha);
 lcd.createChar (1,flecha_izq);
```

- Configurar el LCD. Para utilizar el LCD será preciso utilizar una serie de instrucciones propias de la librería LiquidCrystal .

```
 lcd.init();
 lcd.clear();
 lcd.noBacklight();
 lcd.Backlight();
 lcd.setCursor(0,0);
 lcd.print("Recorrido Robot");
 lcd.write (byte (1));
```

A continuación se explicará la función de cada una de las instrucciones anteriores:

Instrucción	Función
lcd.init()	Inicializa el LCD.
lcd.clear()	Borra la pantalla del LCD.
lcd.Backlight()	Enciende la luz de fondo del LCD.
lcd.noBacklight()	Apaga la luz de fondo del LCD.
lcd.setCursor(0,0)	Coloca el cursor donde se le indique.
lcd.Print(“ “)	Escribe literal lo escrito dentro de las comillas.
lcd.Print()	Escribe el valor de la variable puesta entre los paréntesis.
lcd.Write()	Escribe en la pantalla los caracteres creados.

Tabla 2.9.2.2.1.1.- Instrucciones control LCD.

2.9.2.2.2.- Control remoto

Para llevar a cabo el control remoto se llevan a cabo los siguientes pasos:

- Declaración de la librería IRremote.

```
#include <IRremote.h>
```

- Declaración del pin encargado de la recepción de datos provenientes del receptor de infrarrojos.

```
const int RECV_PIN = 52;
IRrecv irrecv(RECV_PIN);
```

- Declaración de que se usará un decodificador para leer los datos del receptor e inicialización del receptor.

```
decode_results results;
irrecv.enableIRIn();
```

- Control del programa. A través del mando a distancia se controlará el funcionamiento del programa. Este control se llevará a cabo con una serie de botones que se mostrarán, junto a la función que desempeñan y a su código a continuación:

Figura 2.9.2.2.2.1- Controles mando a distancia.

2.9.2.2.3.- Algoritmo

A continuación se explicará el algoritmo en función del botón pulsado. Para ello se recurrirá tanto a menciones del código, como a imágenes del resultado final obtenido.

- ON/OFF. Con este botón se encenderá y apagará la pantalla. Una vez encendida aparecerá el siguiente mensaje:

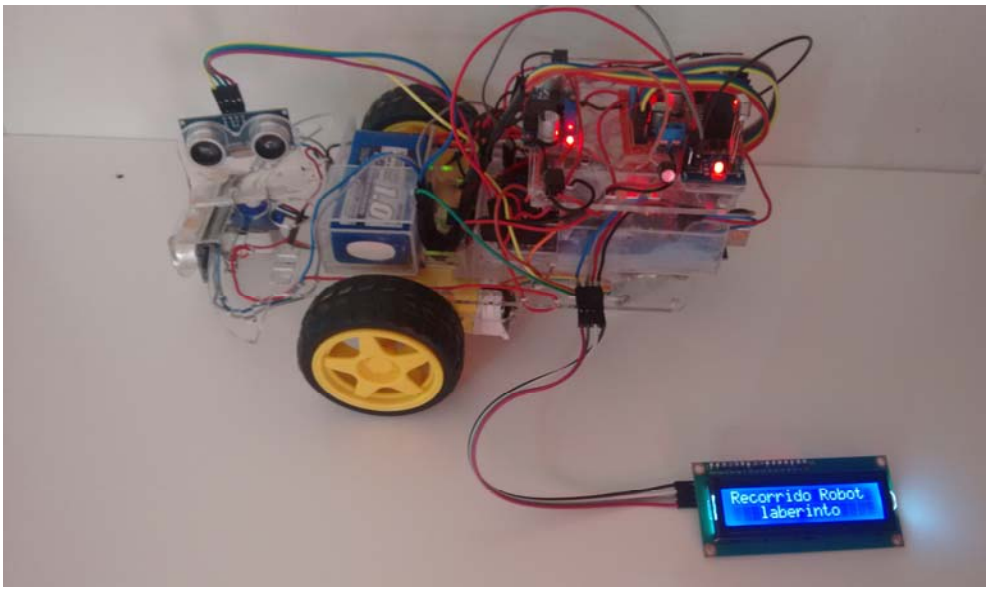


Figura 2.9.2.2.3.1- Conexión LCD.

En la imagen anterior también puede observarse como se conecta el LCD, cuyas conexiones ya fueron explicadas.

Mientras la pantalla muestra el mensaje de inicio, internamente el programa se realiza la lectura de la EEPROM. A la par se almacena el valor del total de las casillas recorridas.

```
for (int i=0; i <= 80; i++){
  camino[i]=EEPROM.read (i);
  if (camino[i]==0)
  {
    casillas_recorridas=i;
    break;
  }
}
```

Una vez que se realiza lo anterior se calcula el camino más corto. Para hallar el camino más corto se utiliza un algoritmo basado en un bucle anidado en el cual se trabajará con el vector que almacena el recorrido del robot. Se irá comparando cada una de las posiciones con las sucesivas. Si una posición está repetida entonces quiere decir que el robot realizó un recorrido que no era el

correcto y volvió a pasar por esa posición, por lo que para realizar el camino más corto habría que eliminar ese camino extra. Cuando esto sucede se guarda la posición repetida y se repite el proceso de comparación pero ahora se empieza a comparar la posición inmediatamente posterior a la posición que se repitió. A continuación podrá verse un flujograma del algoritmo:

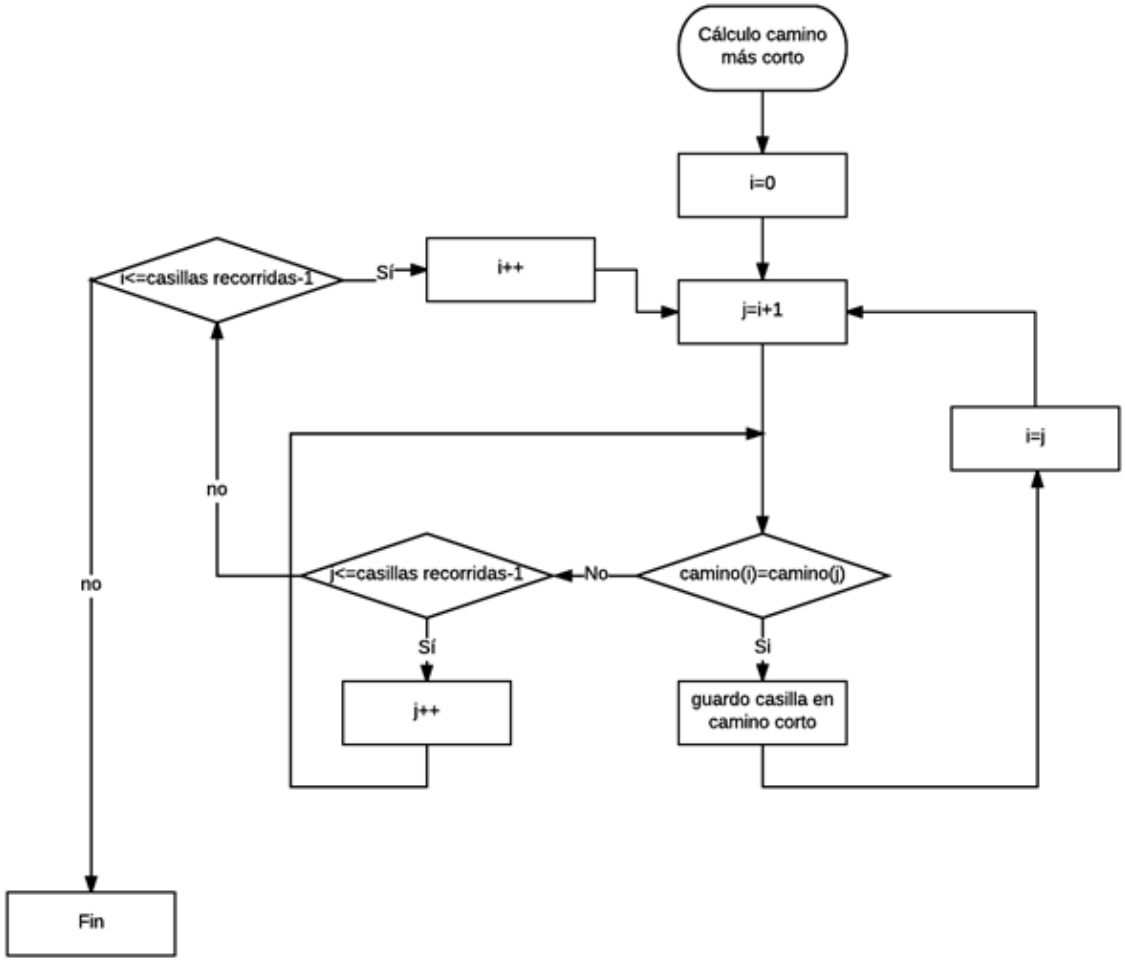


Figura 2.9.2.2.3.2- Flujograma camino más corto.

El algortimo será el siguiente:

```
for (int i=0; i <= casillas_recorridas-1; i++){
  for (int j=i+1; j <= casillas_recorridas-1; j++)
  {
     if (camino[i]==camino[j])
     {
```

```
            atajo=1;
            posicion=j;
            break;
         }
      }
      if (atajo==1)
      {
         camino_corto[casillas_camino_corto]=camino[posicion];
               i=posicion;
         casillas_camino_corto=casillas_camino_corto+1;
         atajo=0;
      }
      else
      {
         camino_corto[casillas_camino_corto]=camino[i];
         casillas_camino_corto++;
      }
}
```

- Derecha/izquierda. Ambos botones permiten desplazarse por el menú.

Figura 2.9.2.2.3.3- Menú 1.

Figura 2.9.2.2.3.4- Menú 2.

- Botón 1. Se accede al recorrido.

Figura 2.9.2.2.3.5- Recorrido.

- Botón 2. Se borra la EEPROM.

Figura 2.9.2.2.3.6- Borrado de EEPROM.

- Botón 3. Camino más corto. La pantalla del camino más corto será similar al del recorrido.

- Arriba/Abajo. Permiten desplazarse por las pantallas de recorrido y recorrido más corto de arriba abajo, desde la primera casilla hasta la última y viceversa.

2.9.2.3.- Extracción códigos mando

Se proporciona un programa para extraer los códigos de un mando ya que cada modelo tiene los suyos propios. Sigue el procedimiento utilizado para la recepción infrarroja en los programas anteriores.

2.9.3.- Laberinto

Siguiendo los pasos descritos en el apartado 2.8.9 se construye un laberinto como el que se muestra en la siguiente figura:

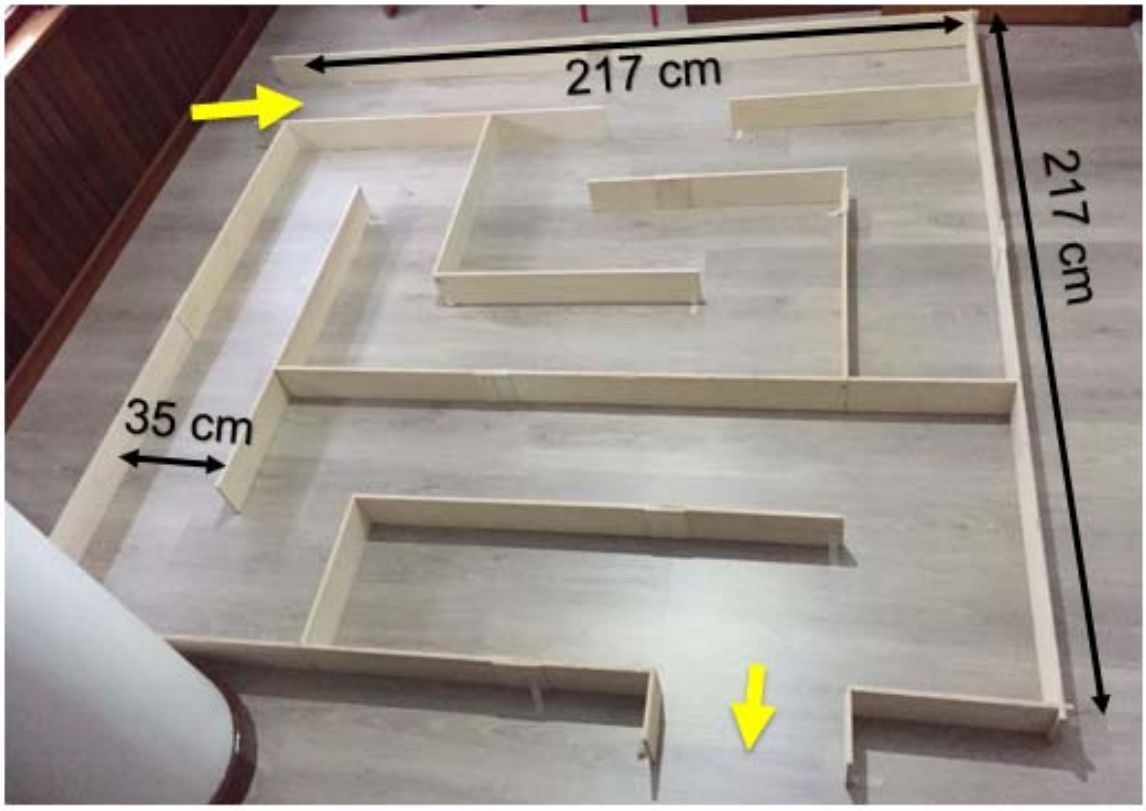


Figura 2.9.2.3.1.- Laberinto.

Cada lado del laberinto medirá 217 cm. Las tablas medirán 12 cm de alto y todas las calles tendrán un ancho de 35 cm.

2.9.4.- Pruebas finales

Una vez construidos los productos finales objeto de este proyecto, tal y como se ha descrito en los puntos anteriores, se debe proceder, como recoge el alcance del proyecto, a llevar a cabo una serie de pruebas que confirmen el buen funcionamiento del robot. Para catalogar el funcionamiento del producto como

válido deben superarse las siguientes pruebas:

- Avanzar a través de un pasillo una longitud equivalente a lo largo del laberinto, es decir, unos 2 metros.

- El robot debe ser capaz de resolver todas las situaciones que se le presenten dentro del laberinto, tanto giros a derecha, como giros a izquierda, como cambios de sentido.

- Alcanzar la meta en un laberinto de dimensiones similares al construido y detenerse tras hacerlo.

- Confirmar funcionamiento del control remoto.

- Confirmar que el camino recorrido que se muestra al usuario es correcto, al igual que el camino más corto mostrado.

Superadas con éxito todas estas pruebas se considera que el proyecto cumple los requerimientos básicos exigidos previamente, así como los objetivos establecidos antes del comienzo del proyecto, por lo que se considera que el robot es apto para su uso.

TÍTULO: **OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO**

ANEXOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**
AVDA. 19 DE FEBRERO, S/N
15405 – FERROL

FECHA: **SEPTIEMBRE 2017**

AUTOR: **JUAN RAMÓN COSTA RODRÍGUEZ**

Fdo.: Juan Ramón Costa Rodríguez

3.- ANEXOS

3.1.- DOCUMENTACIÓN DE PARTIDA

Escuela Universitaria Politécnica Ferrol
UNIVERSIDADE DA CORUÑA

Intranet eup

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: Costa Rodríguez,Juan Ramón
APELIDOS E NOME:

DNI: [redacted] Fecha de Solicitud: Feb2017
DNI: Fecha de Solicitude:

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Optimización del algoritmo de un robot tipo laberinto

Número TFG: 770G01A127

TUTOR:(Titor) Rivas Rodriguez,Juan Manuel

COTUTOR/CODIRECTOR: Esteban Jove Pérez

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento:

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Lunes, 27 de Febrero del 2017

Retirei o meu Traballo Fin de Grado o dia _____ de _____ do ano _____

Fdo: Costa Rodríguez,Juan Ramón

DESCRIPCIÓN Y OBJETIVO:Construcción y programación de un robot tipo ?laberinto?

OBJETO:

El proyecto consistirá en diseñar un algoritmo eficaz para un robot del tipo ?laberinto?.

Se puede implementar sobre un robot existente o sobre un nuevo diseño.

Se puede modificar el robot o añadir los sensores que sean necesarios de acuerdo con la solución tomada.

Además se debe realizar su calibración y demostración en un determinado circuito.

Se deberá detallar en el informe los procedimientos de prueba y sus resultados.

ALCANCE:

Se debe fabricar el robot y probar lo citado anteriormente. Se valorará la fabricación de varias unidades para la EUP.

3.2.- CÁLCULOS

En este capítulo de los anexos se llevan a cabo los cálculos necesarios para justificar aquellas soluciones adoptadas que lo precisen.

3.2.1.- Cálculo voltaje alimentación motores DC

En primer lugar se calculará la longitud de la circunferencia:

$$Longitud\ circunferencia = \pi * diámetro = \pi * 66mm = 20{,}7\ cm \qquad (3.2.1.1)$$

El robot avanzará 20,7 centímetros por cada revolución.

El laberinto, como ya se explicó, está diseñado en forma de matriz cuadrada de 35x35 cm cada casilla de la matriz.

$$Revoluciones\ por\ casilla = {}^{35cm}/_{20{,}7cm} = 1{,}69 \qquad (3.2.1.2)$$

Por cada 1,67 giros de la rueda se recorrerá una casilla.

Se sabe, porque lo proporciona el fabricante, que, sin carga y con 6 V de alimentación, la rueda giraría a 100 RPM. Por tanto para recorrer todo un lado del laberinto el robot tardaría:

$$Longitud\ lado\ laberinto = 35cm * 6 + 5 = 215\ cm \qquad (3.2.1.3)$$

$$Revoluciones\ necesarias = {}^{215\ cm}/_{20{,}7cm} = 10{,}38\ revoluciones \qquad (3.2.1.4)$$

$$Tiempo = \left({}^{1\ min}/_{100\ rev}\right) * 10{,}38\ rev * 60\ segundos = 6{,}2\ segundos \qquad (3.2.1.5)$$

Se deduce que para recorrer todo el largo del laberinto construido para las pruebas, se tardaría poco más que 6,2 segundos, a casi un segundo por casilla. Se considera que este valor de tiempo es más que aceptable, por lo que se

establece, 6 V como un voltaje adecuado para la alimentación de los motores.

3.2.2.- Estimación del consumo del sistema

Para la selección de la batería es preciso hacer una estimación de la corriente mínima que va a consumir el sistema, es decir, de la carga total.

La carga no se conoce exactamente y tampoco puede conocersese, ya que el sistema aún no está implementado, pero sí que se conocen valores aproximados. Se sabe que el Arduino consumirá entorno a 40 mA, los dos sensores de ultrasonidos 30 mA, los dos optointerruptores 40mA y el servo unos 300 mA. Por su parte, los motres DC, para un valor estimado de 100 rpm y 6V de alimentación consumen 180 mA cada uno. Como la alimentación de los motores y las velocidades van a andar más o menos sobre eses niveles, se estima que entre los dos motores se consumirán 360 mA. El resto de componentes, tanto el controlador de motores, como el convertidor DC-DC y el receptor IR también consumirán corriente.En total se estima que como mínimo la carga consumirá unos 800 mA.

$$Consumo\ estimado = Arduino + (Optointerruptor * 2) + (2 * motor\ DC) + (2 * Sensor\ ultrasonidos) + Servomotor + Convertidor\ DC\ DC + Receptor\ IR + Controlador\ motores \quad (3.2.1.6)$$

$$Consumo\ estimado = 40mA + (20mA * 2) + (2 * 180mA) + (2 * 15mA) + 300mA + 5\ mA + 2{,}5mA = 807{,}5\ mA$$

Se considera el consumo calculado anteriormente como una estimación de carga mínima que consumirá el sistema, asumiendo que la carga final consumirá más corriente.

3.2.3.- Dimensionamiento capacidad de la batería

El consumo mínimo de corriente para la carga que se conectará a la batería se estima que es de unos 800 mA. Para una batería con capacidad de 800 mAh el tiempo de duración sería:

$$Tiempo(\min) = \frac{Capac.(mAh)}{Velocidad\ descarga(mA)} = \frac{800\ mA}{800\ mA} = 1\ hora \qquad (3.2.3.1)$$

La batería finalmente seleccionada tiene una capacidad de 1000mAh. Una vez esta ya está seleccionada y se construye el robot se llevan a cabo diferentes mediciones y se determina que la corriente consumida será de 1,2 A, por lo que el robot tendrá una autonomía de:

$$Duración\ batería(\min) = \frac{1000}{1200} = 50\ minutos \qquad (3.2.3.2)$$

Uno de los requisitos de diseño era que el robot tuviera suficiente autonomía como para llevar a cabo, como mínimo, 10 expediciones. Para calcular si se cumplirá el requisito habrá que situarse en el peor de los casos, en el cual el robot recorrerá 72 casillas. En el peor de los casos, el robot necesitará regular su avance una de queda dos casillas, por lo que si como se estimó en el apartado 3.2.1, tardará más o menos 1 segundo en avanzar una casilla, entonces tardará casi 4 segundos en avanzar dos casillas con la regulación. Se estima que realizar un giro a derechas con regulación de la posición lleva unos 4 segundos y que hacerlo a izquierdas, con regulación también unos 8. En el caso en el que se están haciendo los cálculos, se realizarán 10 giros a izquierdas, 10 giros a derechas y 1 cambio de sentido, que con regulación se estima que durará unos 4 segundos. A esto hay que sumarle además la comprobación de la salida, que sumará otro giro a izquierdas más la respectiva comprobación, por lo que en el peor de los casos llevaría 10 segundos más. En total se obtiene un tiempo de 278 segundos, es decir, 4 minutos y 38 segundos.

En el peor de los casos el robot tardaría en salir del laberinto 4,6 minutos. Si se diera la casualidad de que se recorriera 10 veces el laberinto y todas las veces el robot se pusiera en el caso anterior, este tomaría un total de 46 minutos para encontrar la salida 10 veces. Se determina entonces que la batería escogida cumple el requisito de diseño respectivo a la autonomía del robot.

3.3.- Guía usuario

En este capítulo se desarrolla un manual para que el usuario utilice correctamente el dispositivo. Se detallarán todos los pasos ordenadamente, intentando que sean lo más claros posibles para que cualquier usuario pueda poner en marcha el robot, así como navegar por el sistema. Se recomienda que el lector se apoye en la lectura del punto 2.9.2.2.3, ya que en él aparecen imágenes de los diferentes estados del menú.

3.3.1.- Carga del programa

Por defecto, el sistema se encuentra cargado con el algoritmo principal que resuelve el problema del laberinto. En caso de no estar cargado, será preciso conectar el puerto USB del ordenador con el del Arduino y cargar el programa de nombre Algoritmo_Laberinto.

Por defecto también, la entrada del laberinto se establece en la casilla (1,1). Si se quiere cambiar, solamente deberá modificar en el código del programa la fila y la columna donde se encuentra la correspondiente indicación.

3.3.2.- Puesta en marcha

Para poner en marcha el sistema es preciso seguir tres sencillos pasos. El primer paso será encender el interruptor, que permita que la alimentación llegue a los circuitos que gobiernan el robot. El segundo paso será colocar el robot en la entrada del laberinto, procurando colocarlo lo más centrado posible. El siguiente y último paso será pulsar el botón de encender en el mando a distancia. Hay que tener en cuenta que el sistema de control remoto es similar al de las televisiones, por lo que habrá que tener las mismas precauciones e intentar apuntar con el

mando al receptor IR.

Es de suma importancia que el usuario compruebe siempre la tensión de la batería antes de poner en marcha el robot ya que hay que controlar que no baje de un valor de 10,8V.

Una vez que el robot resuelve el laberinto, para volver a utilizarlo es preciso apagar el interruptor para resetear el programa.

3.3.4.- Apagar el sistema

Para apagar el sistema simplemente habrá que apagar el interruptor.

3.3.5.- Navegar por el menú

Para navegar por el menú es preciso conectar el Arduino al PC y cargar el programa de nombre Menu_Robot_Laberinto. Una vez cargado el usuario tiene dos opciones; o bien mantener el robot conectado al PC, o bien desconectarlo y encender el interruptor. De ambas formas funcionará. Una vez cargado, el usuario dispone de una serie de botones para navegar por el menú.

En primer lugar debe encender el LCD pulsando el botón ON/OFF.

Una vez encendido el usuario deberá escoger una opción. Para desplazarse de izquierda a derecha para ver todas las opciones usará los botones que tienen una flecha hacia la izquierda y otra a la derecha. Para escoger una opción simplemente deberá pulsar un botón del 1 al 3. Con el botón 1 accederá a ver el recorrido del robot, con el 2 borrará la EEPROM y con el 3 accederá a ver el camino más corto. Para visualizar el recorrido es preciso desplazarse usando los botones P+ y P-.

3.3.6.- Extraer códigos mando

Para que el usuario pueda utilizar cualquier mando a distancia que tenga a

mano, se proporciona un programa con el cual pueda extraer los códigos de su mando. Para realizar esto el usuario únicamente tiene que conectar el Arduino al PC y cargar el programa de nombre Extraccion_Codigos_Mando. Una vez que se haya cargado el programa, deberá abrir el puerto serie desde el ordenador y pulsar aquellos botones de los cuales quiera conocer el código que tienen asociado, que se representará en el puerto serie. Una vez que los tenga únicamente deberá sustituir en el código del programa, los códigos de los botones que desee usar.

3.4.- Códigos de programación

3.4.1.- Algoritmo_laberinto

```
//INTRODUCCIÓN DE DATOS POR PARTE DEL USUARIO
//Introducir casilla de entrada del laberinto en fila=_ y columna_:
int fila=1;
int columna=1; /*Por defecto será fila 1 y columna 1. No es necesario
cambiarla para que el
robot cumpla su cometido ni para que se guarde el recorrido realizado
para alcanzarlo,
pero se ofrece la posibilidad de cambiarla por si el usuario considera
conveniente tener
un criterio de direcciones. El criterio de los valores de las direccones
de desplazamiento
depende del valor de la casilla de entrada*/

// Configuración de pines
int motorDerAdelante=3;//El pin 3 a In1 del L298N
int motorDerAtras=4;//El pin 4 a In2 del L298N
int motorIzqAdelante=5;//El pin 5 a In3 del L298N
int motorIzqAtras=6;//El pin 6 a In4 del L298N
int VelDerecho=2;  //El pin 2 a EnA del L298N
int VelIzquierdo=7;//El pin 7 aEnB del L298N
const int RECV_PIN = 52;//Entrada receptor IR
int encoder1=18;//Entrada opto interruptor 1
int encoder2=19;//Entrada opto interruptor 2

//Definición de librerías
#include <EEPROM.h>//Se incluye la librería EEPROM
```

```
#include <Servo.h>
#include <IRremote.h>

//declaraciones receptor de infrarrojos
IRrecv irrecv(RECV_PIN);
decode_results results;

//declaración de variables
Servo servoMotor;
int caso=1;
int caso_anterior;
bool pared_lateral;
bool pared_frontal;
bool giro_efectuado=0;
long distancia;
long tiempo;
long tiempo_lateral;
long distancia_lateral;
long distancia_lateral2;
bool salida=0;
static volatile unsigned long debounce1 = 0; // Tiempo del rebote encoder
izquierdo.
static volatile unsigned long debounce2 = 0; // Tiempo del rebote encoder
derecho.
volatile int contador_izq = 0; //contador pulsos encoder izquierdo
volatile int contador_dcha = 0; //contador pulsos encoder derecho
bool laberinto[6][6]; //matriz que almacena el recorrido del robot
long camino_recorrido; //almacenamiento en bruto (como nº entero) del
recorrido
bool pared_detectada_frontal;
int giro;
int direccion_desplazamiento=1;
int direccion=0;
int direccion_anterior=1;
bool empezar;
unsigned long codigo;

void setup() {

  Serial.begin(9600);//Se inicializa el puerto serie a 9600 baudios
```

```
  //configuración interrupciones encoder
  pinMode(18 , INPUT);  //definir pin 18 como entrada
  attachInterrupt(5, calculo_direccion, RISING) ; //definir interrupcion
  pinMode(19 , INPUT);  //definir pin 19 como entrada
  attachInterrupt(4, calculo_direccion, RISING) ; //definir interrupción

  //configuración de pines
  pinMode(motorDerAdelante, OUTPUT);
  pinMode(motorDerAtras, OUTPUT);
  pinMode(motorIzqAdelante, OUTPUT);
  pinMode(motorIzqAtras, OUTPUT);
  pinMode(VelDerecho, OUTPUT);
  pinMode(VelIzquierdo, OUTPUT);
  pinMode(10, OUTPUT); /*activavión del pin 10 como salida para el pulso
ultrasónico*/
  pinMode(9, INPUT); /*activavión del pin 9 como entrada para el pulso
ultrasónico*/
  pinMode(12, OUTPUT); /*activavión del pin 12 como salida para el pulso
ultrasónico*/
  pinMode(11, INPUT);/*activavión del pin 11 como entrada para el pulso
ultrasónico*/
  servoMotor.attach(8);/*configuración del pin 8 como entrada PWM del
servo*/

  //inicialización
  servoMotor.write(180);/*se inicializa servomotor a 180º (posición
lateral izquierda*/
  camino_recorrido=(fila*10)+columna; /*se inicializa el camino recorrido
por el robot a la posición
  de entrada del laberinto*/
  EEPROM.write(direccion,camino_recorrido);/*se escribe en la dirección 0
de la memoria EEPROM
  el valor de la casilla de entrada del laberinto. Durante la ejecución
del programa y el consecuente
  avance del robot por las calles del laberinto, se irá guardando en las
direcciones posteriores
  de esta memoria las casillas recorridas por el robot.*/

  //Control del encendido por control remoto
  while (empezar==0)
  {
```

```
     if (irrecv.decode(&results))
     {
        codigo=results.value;
        if (codigo==3772793023)// boton ON
        {
          empezar=1;
        }
     }
  }
}

//Calculo de la nueva direccion de desplazamiento tras recorrer una
casilla
void calculo_direccion(){

    if (digitalRead (encoder1) && (micros()-debounce1 > 20000) &&
digitalRead (encoder1) && contador_izq<35 && caso==1) {
    // Comprobacion de que el encoder envia una señal buena y luego
comprueba que el tiempo es superior a 20 ms.
        debounce1 = micros(); // Almacena el tiempo para comprobar que no
se cuenta el rebote que hay en la señal.
        contador_izq++;
    }  // Suma el pulso bueno que entra.
    if (digitalRead (encoder2) && (micros()-debounce2 > 20000) &&
digitalRead (encoder2) && contador_dcha<35 && caso==1) {
    // Comprobacion de que el encoder envia una señal buena y luego
comprueba que el tiempo es superior a 20 ms.
        debounce2 = micros(); // Almacena el tiempo para comprobar que se
cuenta el rebote que hay en la señal.
        contador_dcha++;
    }
    else if((contador_izq>=35 &&
contador_dcha>=35)||(giro_efectuado==1)){//Se ha recorrido una casilla
        contador_izq=0;
        contador_dcha=0;
        giro_efectuado=0;

        switch(direccion_desplazamiento)//Renovacion matriz que almacena
el recorrido
        {
          case 1:
```

```
            columna=columna+1;
            laberinto[fila][columna]=1;
          break;

          case 2:
            fila=fila+1;
            laberinto[fila][columna]=1;
          break;

          case 3:
            fila=fila-1;
            laberinto[fila][columna]=1;
          break;

          case 4:
             columna=columna-1;
             laberinto[fila][columna]=1;
          break;
        }
        direccion=direccion+1; // se aumenta la direccion de la EEPROM
        camino_recorrido=(fila*10)+columna; //Posición casilla recorrida
        EEPROM.write(direccion,camino_recorrido); //Se guarda valor en la
EEPROM
    }
}

//Calcula nueva direccion despues de un giro
void cambio_direccion(int direccion_anterior)
{
   contador_izq=0;
   contador_dcha=0;
   switch(giro)
   {
      case 2:// giro a derechas
        if (direccion_anterior==1)
        {
          direccion_desplazamiento=2;
        }
        else if (direccion_anterior==2)
        {
          direccion_desplazamiento=4;
```

```
        }
        else if (direccion_anterior==3)
        {
          direccion_desplazamiento=1;
        }
        else if (direccion_anterior==4)
        {
          direccion_desplazamiento=3;
        }
      break;
      case 3://Giro a izquierdas
        if (direccion_anterior==1)
        {
          direccion_desplazamiento=3;
        }
        else if (direccion_anterior==2)
        {
          direccion_desplazamiento=1;
        }
        else if (direccion_anterior==3)
        {
          direccion_desplazamiento=4;
        }
        else if (direccion_anterior==4)
        {
          direccion_desplazamiento=2;
        }
      break;
      case 4://Cambio de sentido
        if (direccion_anterior==1)
        {
           direccion_desplazamiento=4;
        }
        else if (direccion_anterior==2)
        {
          direccion_desplazamiento=3;
        }
        else if (direccion_anterior==3)
        {
          direccion_desplazamiento=2;
        }
```

```
        else if (direccion_anterior==4)
        {
          direccion_desplazamiento=1;
        }
      break;
   }
}

//Manejo de los sensores d eultrasonidos para medir distancia.
long medir_distancia(int pin_trig, int pin_echo)
{
  digitalWrite(pin_trig,LOW); /* Por cuestión de estabilización del
sensor*/
  delayMicroseconds(5);
  digitalWrite(pin_trig, HIGH); /* envío del pulso ultrasónico*/
  delayMicroseconds(10);
  tiempo=pulseIn(pin_echo, HIGH); /* Función para medir la longitud del
pulso entrante. Mide el tiempo que transcurrido entre el envio
                            del pulso ultrasónico y la recepcion   */
   distancia= int(0.017*tiempo);
   return distancia;
}
void robot_adelante() //los dos motores giran en sentido antihorario
{
  digitalWrite(motorDerAdelante,HIGH);
  digitalWrite(motorDerAtras,LOW);
  digitalWrite(motorIzqAdelante,HIGH);
  digitalWrite(motorIzqAtras,LOW);
}
void girar_derecha()//Motor derecha horario e izquierda antihorario
{
  digitalWrite(motorDerAdelante,LOW);
  digitalWrite(motorDerAtras,HIGH);
  digitalWrite(motorIzqAdelante,HIGH);
  digitalWrite(motorIzqAtras,LOW);
}
void girar_izquierda()//Motor izquierda horario y derecha antihorario
{
  digitalWrite(motorDerAdelante,HIGH);
  digitalWrite(motorDerAtras,LOW);
  digitalWrite(motorIzqAdelante,LOW);
```

```
  digitalWrite(motorIzqAtras,HIGH);
}
void frenar_robot()//Detener giro de los motores
{
  digitalWrite(motorDerAdelante,LOW);
  digitalWrite(motorDerAtras,LOW);
  digitalWrite(motorIzqAdelante,LOW);
  digitalWrite(motorIzqAtras,LOW);
  analogWrite(VelDerecho,0);
  analogWrite(VelIzquierdo,0);
}
void robot_atras() //los dos motores giran en sentido antihorario
{
  digitalWrite(motorDerAdelante,LOW);
  digitalWrite(motorDerAtras,HIGH);
  digitalWrite(motorIzqAdelante,LOW);
  digitalWrite(motorIzqAtras,HIGH);
}
void loop() {
  switch(caso)
  {
    case 1://Avance frontal
    {
      digitalWrite(10,LOW);
      delayMicroseconds(5);
      digitalWrite(10, HIGH);
      delayMicroseconds(10);
      tiempo=pulseIn(9, HIGH);
      distancia= int(0.017*tiempo);//Distancia frontal

      digitalWrite(12,LOW);
      delayMicroseconds(5);
      digitalWrite(12, HIGH);
      delayMicroseconds(10);
      tiempo=pulseIn(11, HIGH);
      distancia_lateral=int(0.017*tiempo); //Distancia lateral
      if(distancia==6)
      {
        caso=2;
        pared_frontal=1;
        pared_detectada_frontal=1;
```

```
      }
      else if(distancia_lateral>=18 && distancia_lateral<=23)
      {
        robot_adelante();
        analogWrite(VelDerecho,40);
        analogWrite(VelIzquierdo,40);
      }
      else if(distancia_lateral<=14 && distancia_lateral<18)
      {
        robot_adelante();
        analogWrite(VelDerecho,35);
        analogWrite(VelIzquierdo,40);
      }
      else if(distancia_lateral<=10 && distancia_lateral<14)
      {
        robot_adelante();
        analogWrite(VelDerecho,30);
        analogWrite(VelIzquierdo,40);
      }
      else if(distancia_lateral<10)
      {
         detachInterrupt(5);
         detachInterrupt(4);
         caso=12;
         attachInterrupt(5, calculo_direccion, RISING) ;
         attachInterrupt(4, calculo_direccion, RISING) ;
      }
      else if(distancia_lateral>23 && distancia_lateral<=26)
      {
        robot_adelante;
        analogWrite(VelDerecho,40);
        analogWrite(VelIzquierdo,35);
      }
      else if(distancia_lateral>=27 && distancia_lateral<=28)
      {
        robot_adelante();
        analogWrite(VelDerecho,40);
        analogWrite(VelIzquierdo,30);
      }
      else if(distancia_lateral>28 && distancia_lateral<=36)
      {
```

```
          detachInterrupt(5);
          detachInterrupt(4);
          caso=13;
          attachInterrupt(5, calculo_direccion, RISING) ;
          attachInterrupt(4, calculo_direccion, RISING) ;
       }
       else if(distancia_lateral>=42)//camino libre a la izquierda
       {
         frenar_robot();
         caso=2;
         pared_lateral=1;
         delay(1000);
       }
     }break;

  case 2: //detección de pared frontal o camino libre hacia la izquierda
  {
     frenar_robot();
     delay(1000);
     if(pared_frontal==1)//se detecta pared frontal y hay pared leteral:
podrá girar derecha o 180º
     {
       caso=3;
       pared_frontal=0;
     }
     else if(pared_lateral==1)//no hay pared lateral a la izquierda
     {
       pared_lateral=0;
       caso=5;
     }
  }
  break;

  case 3: //tras deteccion de pared frontal se comprueba camino libre a
la derecha
  {
      servoMotor.write(0);
      delay(1500);//tiempo de espera para que servo se mueva a la posición
ordenada
      distancia_lateral= medir_distancia(12,11);
      if (distancia_lateral> 30)
```

```
    {
      caso=4; //giro a derechas
      servoMotor.write(180);
      delay(1500);
    }
    else
    {
       caso=11; //giro 180º
       servoMotor.write(180);
       delay(1000);
    }
}break;
case 4: //girar derecha 90º
{
  distancia_lateral= medir_distancia(12,11);
  robot_atras();
  analogWrite(VelIzquierdo,40);
  analogWrite(VelDerecho,40);
  delay(500);
  frenar_robot();
  delay(300);
  distancia_lateral2= medir_distancia(12,11);
  if ( distancia_lateral2>=(distancia_lateral+2)){
      girar_izquierda();
      analogWrite(VelIzquierdo,40);
      analogWrite(VelDerecho,40);
      delay(300);
      frenar_robot();
      delay(300);
    }
    else if( distancia_lateral>=(distancia_lateral2+2)){
      girar_izquierda();
      analogWrite(VelIzquierdo,40);
      analogWrite(VelDerecho,40);
      delay(300);
      frenar_robot();
      delay(300);
    }
    else if(distancia_lateral2<17){
      girar_izquierda();
      analogWrite(VelIzquierdo,40);
```

```
        analogWrite(VelDerecho,40);
        delay(300);
        frenar_robot();
        delay(300);
        robot_atras();
        analogWrite(VelIzquierdo,40);
        analogWrite(VelDerecho,40);
        delay(400);
        frenar_robot();
        delay(300);
        girar_derecha();
        delay(450);
        frenar_robot();
        delay(300);
        robot_adelante();
        analogWrite(VelIzquierdo,40);
        analogWrite(VelDerecho,40);
        delay(400);
        frenar_robot();
        delay(300);
      }
      else if(distancia_lateral2>22){
          robot_adelante();
          analogWrite(VelIzquierdo,40);
          analogWrite(VelDerecho,40);
          delay(200);
          frenar_robot();
          delay(300);
      }
      girar_derecha();
      analogWrite(VelDerecho,80);
      analogWrite(VelIzquierdo,80);
      delay(350);
      frenar_robot();
      delay(300);
      servoMotor.write(180);
      delay(1500);
      giro=2;
      cambio_direccion(direccion_desplazamiento);
      giro_efectuado=1;
      calculo_direccion();
```

```
      caso=1;
    }
    break;

    case 5: //se detectó camino libre por lo que tiene que asegurarse de
si eso es cierto y actuar en consecuencia
    {
      distancia_lateral= medir_distancia(12,11);
      robot_adelante();//leve avance y giro a la izquierda para comprobar
camino libre
      analogWrite(VelDerecho,40);
      analogWrite(VelIzquierdo,40);
      delay(200);
      girar_izquierda();
      analogWrite(VelDerecho,40);
      analogWrite(VelIzquierdo,40);
      delay(200);
      frenar_robot();
      delay(500);
      distancia_lateral2= medir_distancia(12,11); // se mide la distancia
lateral para la comprobación
      robot_atras(); //ahora se deshará el movimiento realizado para la
comprobación
      analogWrite(VelDerecho,40);
      analogWrite(VelIzquierdo,40);
      delay(200);
      girar_derecha();
      analogWrite(VelDerecho,40);
      analogWrite(VelIzquierdo,40);
      delay(200);
      frenar_robot();
      if (distancia_lateral>45 && distancia_lateral2>45) //comprobación
camino libre
      {
            distancia= medir_distancia(10,9); // medición de distancia
frontal
            if (distancia<45)
            {
               pared_frontal=1;
            }
            else
```

```
            {
               pared_frontal=0;
            }
            caso=7;
      }
      else{
        frenar_robot();
        delay(500);
        caso=1;
      }
    }break;

    case 6: //avance frontal hasta detectar pared
    {
        distancia= medir_distancia(10,9);
        robot_adelante();
        analogWrite(VelDerecho,40);
        analogWrite(VelIzquierdo,40);
        if (distancia==7)
        {
          frenar_robot();
          delay(1000);
          caso=8; //giro izquierdas
        }
    }break;
    case 7: // avance de unos centimetros para realizar el posterior giro
con correción de posición
    {
        servoMotor.write(0);
        delay(1500);
        distancia_lateral= medir_distancia(12,11); //primera captura de
primera referencia lateral
        robot_adelante(); //avance robot
        analogWrite(VelIzquierdo,40);
        analogWrite(VelDerecho,40);
        delay(500);
        frenar_robot();
        delay(300);
        distancia_lateral2= medir_distancia(12,11); //segunda captura de
referenecia lateral
        //correciones de posición del robot
```

```
        if ( distancia_lateral2>=(distancia_lateral+2)){
           girar_derecha();
           analogWrite(VelIzquierdo,40);
           analogWrite(VelDerecho,40);
           delay(300);
           frenar_robot();
           delay(300);
        }
        else if(( distancia_lateral>=(distancia_lateral2+2))&&
distancia_lateral2>17){
           girar_izquierda();
           analogWrite(VelIzquierdo,40);
           analogWrite(VelDerecho,40);
           delay(300);
           frenar_robot();
           delay(300);
        }
        else if(distancia_lateral2<17){
           girar_derecha();
           analogWrite(VelIzquierdo,40);
           analogWrite(VelDerecho,40);
           delay(300);
           frenar_robot();
           delay(300);
           robot_atras();
           analogWrite(VelIzquierdo,40);
           analogWrite(VelDerecho,40);
           delay(450);
           frenar_robot();
           delay(300);
           girar_izquierda();
           analogWrite(VelIzquierdo,40);
           analogWrite(VelDerecho,40);
           delay(300);
           frenar_robot();
           delay(300);
           robot_adelante();
           analogWrite(VelIzquierdo,40);
           analogWrite(VelDerecho,40);
           delay(400);
           frenar_robot();
```

```
          delay(300);
      }
      if (pared_frontal==1){
           pared_frontal=0;
           caso=6;
      }else{
         distancia_lateral= medir_distancia(12,11);
         robot_adelante();
         analogWrite(VelIzquierdo,40);
         analogWrite(VelDerecho,40);
         delay(500);
         frenar_robot();
         distancia_lateral2= medir_distancia(12,11);
         if ( distancia_lateral2>(distancia_lateral+2)){
            girar_derecha();
            analogWrite(VelIzquierdo,40);
            analogWrite(VelDerecho,40);
            delay(300);
            frenar_robot();
            delay(300);
         }
         else if( distancia_lateral>(distancia_lateral2+2)){
            girar_izquierda();
            analogWrite(VelIzquierdo,40);
            analogWrite(VelDerecho,40);
            delay(300);
            frenar_robot();
            delay(300);
         }
      }
      servoMotor.write(180);
      delay(1500);
      caso=8;
  }break;

  case 8: //girar izquierda 90º
  {
      girar_izquierda();
      analogWrite(VelDerecho,80);
      analogWrite(VelIzquierdo,80);
      delay(350);
```

```
        frenar_robot();
        delay(300);
        giro=3;
        cambio_direccion(direccion_desplazamiento);
        giro_efectuado=1;
        calculo_direccion();
        caso=9;
    }break;

    case 9: //avance frontal sin detección de pared
    {
        robot_adelante();
        analogWrite(VelDerecho,60);
        analogWrite(VelIzquierdo,60);
        delay(700);
        frenar_robot();
        delay(300);
        caso=10;
        delay(1000);
    }break;

    case 10: // comprobación de estado del robot: salida, camino libre a
la izquierda o avance frontal
    {
        distancia_lateral= medir_distancia(12,11);
        distancia= medir_distancia(10,9);
        delay(500);
        servoMotor.write(180);
        delay(1500);
        if(distancia_lateral>120)//comproación de salida del laberinto
        {
             servoMotor.write(0);
             delay(1500);
             distancia_lateral=medir_distancia(12,11);
             if (distancia_lateral>100)
             {
                 salida==1;
                 caso=19; //salida
             }else
             {
                 servoMotor.write(180);
```

```
                delay(1500);
                caso=6;
             }
        }
        else if (distancia_lateral>40 && distancia < 35)//camino libre a
la izquierda y pared frontal
        {
           caso=6;
        }
        else if (distancia_lateral>40)//camino libre a la izquierda
        {
           caso=9;
        }
        else //avance frontal
        {
           caso=1;
        }
    }break;
    case 11: //giro 180º
    {
        distancia_lateral= medir_distancia(12,11);
        robot_atras();
        analogWrite(VelIzquierdo,40);
        analogWrite(VelDerecho,40);
        delay(500);
        frenar_robot();
        delay(500);
        distancia_lateral2= medir_distancia(12,11);
        if ( distancia_lateral2>=(distancia_lateral+2)){
            girar_derecha();
            analogWrite(VelIzquierdo,40);
            analogWrite(VelDerecho,40);
            delay(300);
            frenar_robot();
            delay(300);
        }
        else if( distancia_lateral>=(distancia_lateral2+2)){
            girar_izquierda();
            analogWrite(VelIzquierdo,40);
            analogWrite(VelDerecho,40);
            delay(300);
```

```
            frenar_robot();
            delay(300);
        }
        else if(distancia_lateral2<17){
             girar_izquierda();
             analogWrite(VelIzquierdo,40);
             analogWrite(VelDerecho,40);
             delay(300);
             robot_atras();
             analogWrite(VelIzquierdo,40);
             analogWrite(VelDerecho,40);
             delay(300);
             girar_derecha();
             analogWrite(VelIzquierdo,40);
             analogWrite(VelDerecho,40);
             delay(450);
             digitalWrite(motorDerAdelante,LOW);
             frenar_robot();
             delay(300);
        }
        else if(distancia_lateral2>22){
             girar_derecha();
             analogWrite(VelIzquierdo,40);
             analogWrite(VelDerecho,40);
             delay(300);
             robot_atras();
             analogWrite(VelIzquierdo,40);
             analogWrite(VelDerecho,40);
             delay(300);
             girar_izquierda();
             analogWrite(VelIzquierdo,40);
             analogWrite(VelDerecho,40);
             delay(450);
             frenar_robot();
             delay(300);
       }
       girar_derecha();
       analogWrite(VelIzquierdo,80);
       analogWrite(VelDerecho,80);
       delay(650);
       frenar_robot();
```

```
      delay(500);
      giro=4;
      cambio_direccion(direccion_desplazamiento);
      giro_efectuado=1;
      calculo_direccion();
      caso=1;
   }break;

   case 12:{ //correción 1 avance frontal
       robot_atras();
       analogWrite(VelDerecho,40);
       analogWrite(VelIzquierdo,40);
       delay(400);

       girar_izquierda();
       analogWrite(VelDerecho,50);
       analogWrite(VelIzquierdo,50);
       delay(250);
       frenar_robot();
       delay(300);
       robot_adelante();
       analogWrite(VelDerecho,40);
       analogWrite(VelIzquierdo,40);
       delay(300);
       frenar_robot();
       delay(500);
       caso=1;
   }break;

   case 13:{ //correción 2 avance frontal
       robot_atras();
       analogWrite(VelDerecho,40);
       analogWrite(VelIzquierdo,40);
       delay(400);
       girar_derecha();
       analogWrite(VelDerecho,40);
       analogWrite(VelIzquierdo,40);
       delay(250);
       frenar_robot();
       delay(300);
       robot_adelante();
```

```
        analogWrite(VelDerecho,40);
        analogWrite(VelIzquierdo,40);
        delay(300);
        frenar_robot();
        delay(300);
        caso=1;
    }break;
 }
}
```

3.4.2.- Menu_Robot_Laberinto

```
#include <EEPROM.h>
#include <IRremote.h>
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

LiquidCrystal_I2C  lcd(0x27,16,2); // especifica la dimensión de la
pantalla

//creación de caracteres
byte flecha_dcha[8] = {
B00000,
B11000,
B11100,
B11110,
B11111,
B11110,
B11100,
B11000,
};

byte flecha_izq[8] = {
B00000,
B00011,
B00111,
B01111,
B11111,
B01111,
```

```
B00111,
B00011,
};

const int RECV_PIN = 52;//Pin receptor IR
IRrecv irrecv(RECV_PIN);
decode_results results;

int direccion=0;
int camino[80];
int camino_corto[50];
bool atajo=0;
unsigned long codigo;
bool encendido=0;
bool corto=0;
int posicion;
int casillas_recorridas;
int casilla_actual;
int casillas_camino_corto;
int fila;
int columna;
int caso;

void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();
  lcd.init();
  lcd.clear();
  lcd.noBacklight();
  lcd.createChar (0,flecha_dcha);
  lcd.createChar (1,flecha_izq);
}

void loop() {

   if (irrecv.decode(&results))//si existen datos recibidos
   {
      codigo=results.value;
      switch (codigo)
      {
        case 3772793023: //encender/apagar
```

```
        if (encendido==0) {
           corto=0;
           encendido=1;
           lcd.clear();
           lcd.backlight();
           lcd.setCursor(0,0);
           lcd.print("Recorrido Robot");
           lcd.setCursor(0,1);
           lcd.print("   laberinto");
           for (int i=0; i <= 80; i++){
              camino[i]=EEPROM.read (i);
              if (camino[i]==0)
              {
                 casillas_recorridas=i;
                 break;
              }
           }
           for (int i=0; i <= casillas_recorridas-1; i++){
              for (int j=i+1; j <= casillas_recorridas-1; j++){
                 if (camino[i]==camino[j])
                 {
                    atajo=1;
                    posicion=j;
                    break;
                 }
              }
              if (atajo==1)
              {
                  camino_corto[casillas_camino_corto]=camino[posicion];

                  i=posicion;
                  casillas_camino_corto=casillas_camino_corto+1;
                  atajo=0;
              }
              else
              {
                  camino_corto[casillas_camino_corto]=camino[i];
                  casillas_camino_corto++;
              }
           }
        }
```

```
          else
          {
            lcd.clear();
            lcd.noBacklight();
            encendido=0;
            casilla_actual=0;
          }
          delay(3000);
          lcd.setCursor(0,0);
          lcd.print("1)Ver recorrido");
          lcd.write (byte (0));
          lcd.setCursor(0,1);
          lcd.print("2)Borrar EEPROM");
        break;

        case 3772784863: //1) Ver recorrido
          fila= camino[casilla_actual]/10;
          columna=camino[casilla_actual]%10;
          casilla_actual=0;
          lcd.clear();
          lcd.setCursor(0,0);
          lcd.print("Casilla num");
          lcd.setCursor(13,0);
          lcd.print(casilla_actual);
          lcd.setCursor(0,1);
          lcd.print("Fil: ");
          lcd.print(fila);
          lcd.setCursor(9,1);
          lcd.print("Col:");
          lcd.print(columna);
        break;

        case 3772795063: //P+
          casilla_actual++;
          if (casilla_actual==casillas_recorridas && corto==0)
          {
             casilla_actual=0;
          }
          else if (casilla_actual==casillas_camino_corto && corto==1)
          {
             casilla_actual=0;
```

```
        }
        if (corto==1)
        {
           fila= camino_corto[casilla_actual]/10;
           columna=camino_corto[casilla_actual]%10;
        }
        else
        {
           fila= camino[casilla_actual]/10;
           columna=camino[casilla_actual]%10;
        }
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Casilla num");
        lcd.setCursor(13,0);
        lcd.print(casilla_actual);
        lcd.setCursor(0,1);
        lcd.print("Fil: ");
        lcd.print(fila);
        lcd.setCursor(9,1);
        lcd.print("Col:");
        lcd.print(columna);
      break;

      case 3772817503: //Borrar EEPROM
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("    Borrando");
        lcd.setCursor(0,1);
        lcd.print("    EEPROM...");
        for (int i = 0 ; i < EEPROM.length() ; i++) {
           EEPROM.write(i, 0);
        }
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Recorrido Robot");
        lcd.setCursor(0,1);
        lcd.print("   laberinto");
        casilla_actual=0;
        delay(3000);
        lcd.setCursor(0,0);
```

```
          lcd.print("1)Ver recorrido");
          lcd.write (byte (0));
          lcd.setCursor(0,1);
          lcd.print("2)Borrar EEPROM");
          corto=0;
        break;

        case 3772778743: //P-
          casilla_actual--;
          if (casilla_actual<=0)
          {
             casilla_actual=casillas_recorridas-1;
          }
          if (corto==1)
          {
             fila= camino_corto[casilla_actual]/10;
             columna=camino_corto[casilla_actual]%10;
          }
          else
          {
             fila= camino[casilla_actual]/10;
             columna=camino[casilla_actual]%10;
          }
          lcd.clear();
          lcd.setCursor(0,0);
          lcd.print("Casilla num");
          lcd.setCursor(13,0);
          lcd.print(casilla_actual);
          lcd.setCursor(0,1);
          lcd.print("Fil: ");
          lcd.print(fila);
          lcd.setCursor(9,1);
          lcd.print("Col:");
          lcd.print(columna);
          casilla_actual--;
        break;

        case 3772794553: //derecha
          casilla_actual=0;
          lcd.clear();
          lcd.setCursor(0,0);
```

```
          lcd.print("3)Camino mas");
          lcd.setCursor(0,1);
          lcd.write (byte (1));
          lcd.print(" corto");
        break;

        case 3772819033: //izquierda
          casilla_actual=0;
          lcd.clear();
          lcd.setCursor(0,0);
          lcd.print("1)Ver recorrido");
          lcd.write (byte (0));
          lcd.setCursor(0,1);
          lcd.print("2)Borrar EEPROM");
        break;

        case 3772801183: //3) Camino mas corto
           fila= camino_corto[casilla_actual]/10;
           columna=camino_corto[casilla_actual]%10;
           casilla_actual=0;
           lcd.clear();
           lcd.setCursor(0,0);
           lcd.print("Casilla num");
           lcd.setCursor(13,0);
           lcd.print(casilla_actual);
           lcd.setCursor(0,1);
           lcd.print("Fil: ");
           lcd.print(fila);
           lcd.setCursor(9,1);
           lcd.print("Col:");
           lcd.print(columna);
           corto=1;
        break;
      }
      irrecv.resume(); //recibe el siguiente valor
   }
   delay(100);
}
```

3.4.3- Extracción_Códigos_Mando.

```
#include <IRremote.h>
const int RECV_PIN = 52;
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup()
{
   Serial.begin(9600);
   irrecv.enableIRIn();
}

void loop()
{
   if (irrecv.decode(&results))
   {
      Serial.println(results.value);
      irrecv.resume();
   }
   delay(500);
}
```

TÍTULO: **OPTIMIZACIÓN DEL ALGORTIMO DE UN ROBOT TIPO LABERINTO**

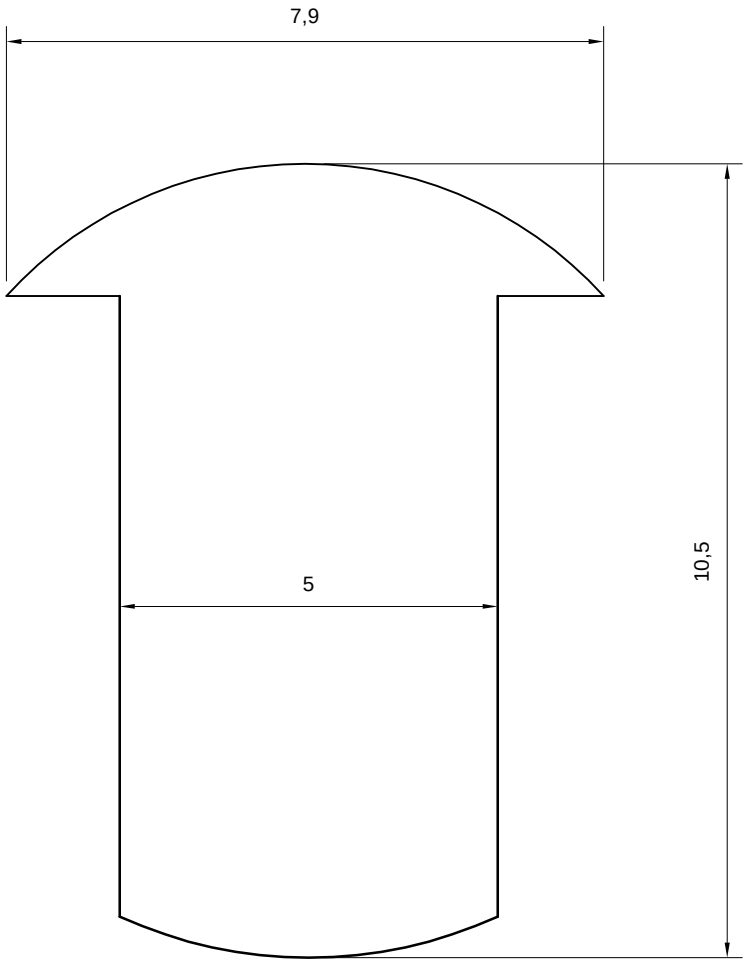
PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**
AVDA. 19 DE FEBRERO, S/N
15405 – FERROL

FECHA: **SEPTIEMBRE 2017**

AUTOR: **JUAN RAMÓN COSTA RODRÍGUEZ**

Fdo.: Juan Ramón Costa Rodríguez



UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01045

TÍTULO DEL TFG:

OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO

TÍTULO DEL PLANO:

PLACA BASE CHÁSIS

FECHA: SEPTIEMBRE-2017

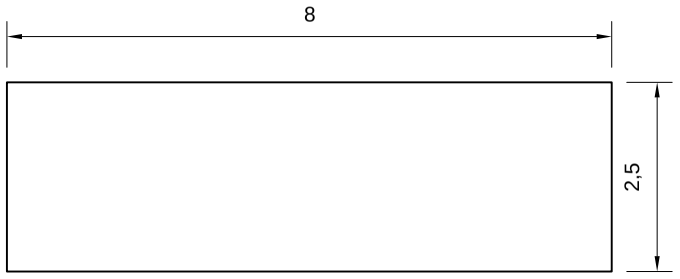
ESCALA: 1:2

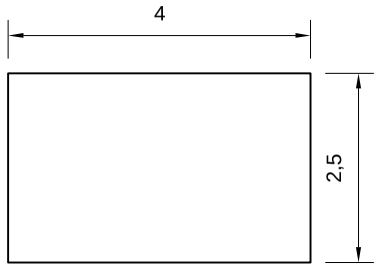
AUTOR:

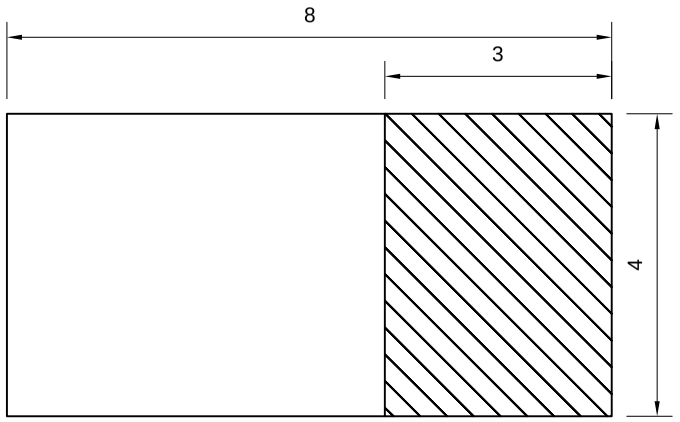
JUAN RAMÓN COSTA RODRÍGUEZ

FIRMA:

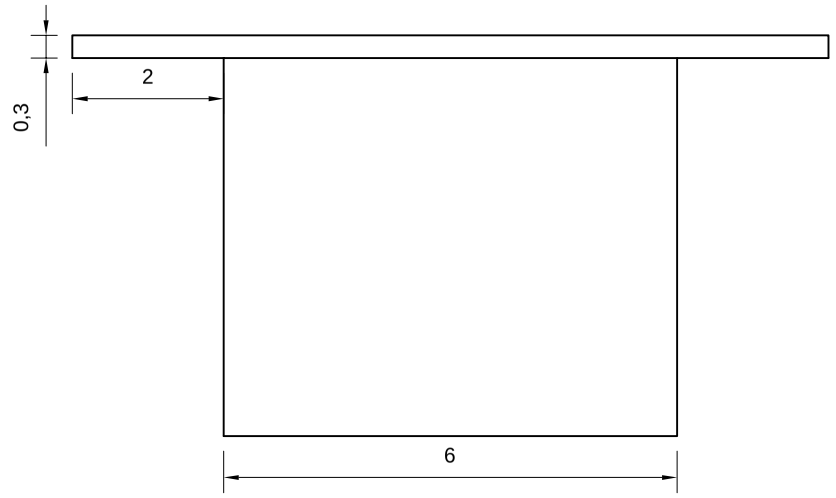
PLANO Nº: 1

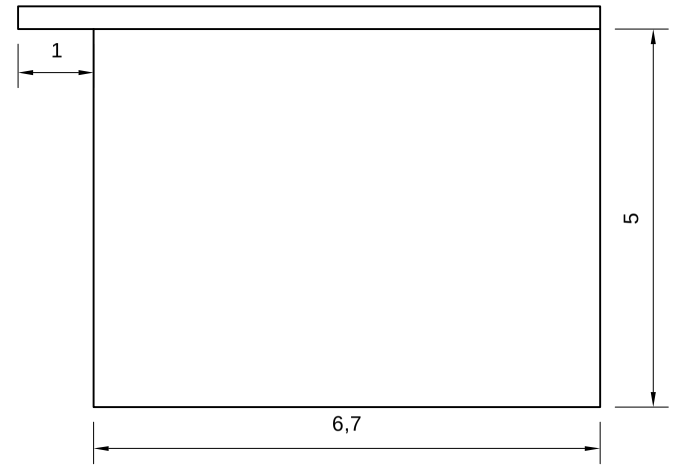




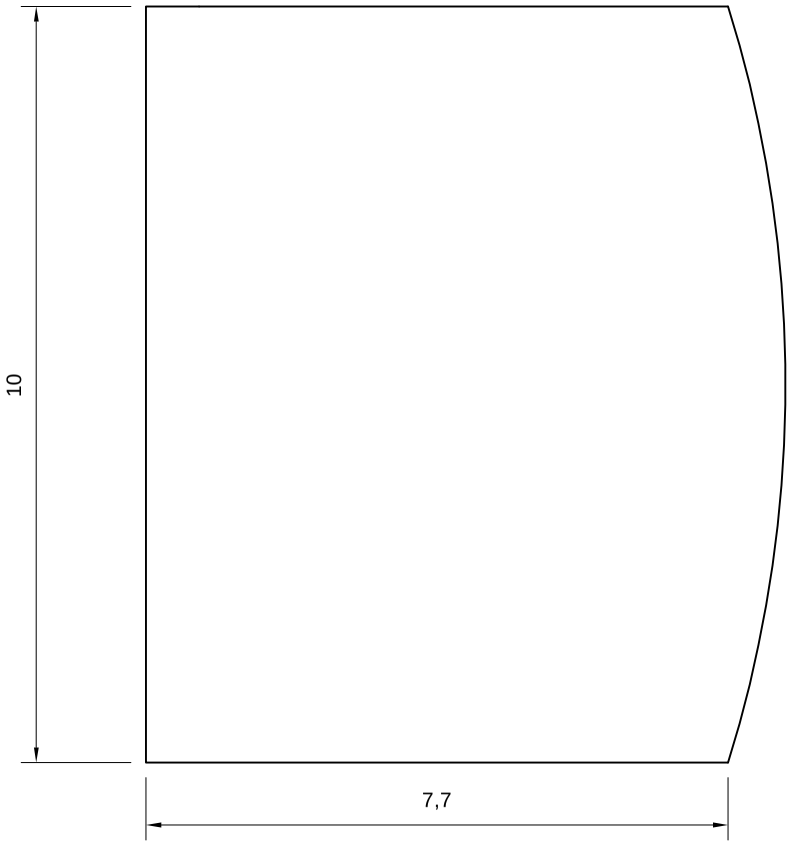


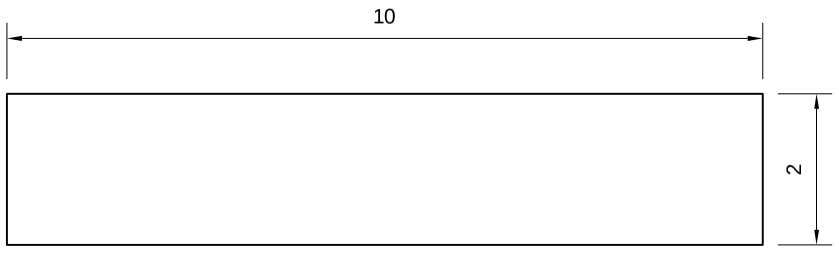
CAJA DE BATERÍA

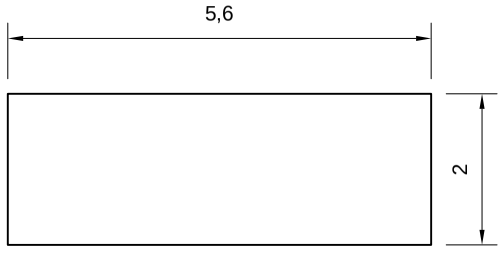


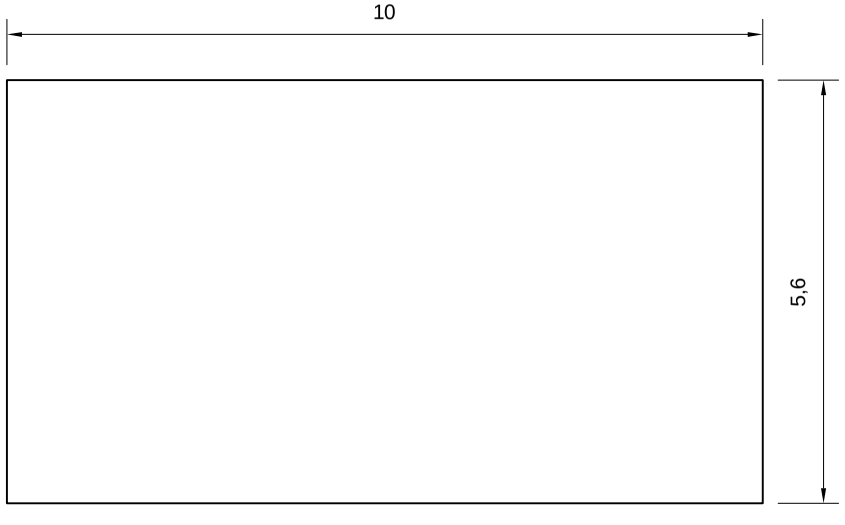


ESTRUCTURA SUPERIOR CHASIS









CAJA ARDUINO

UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01045

TÍTULO DEL TFG:

OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO

TÍTULO DEL PLANO:

ESTRUCTURA CHÁSIS

FECHA: SEPTIEMBRE-2017

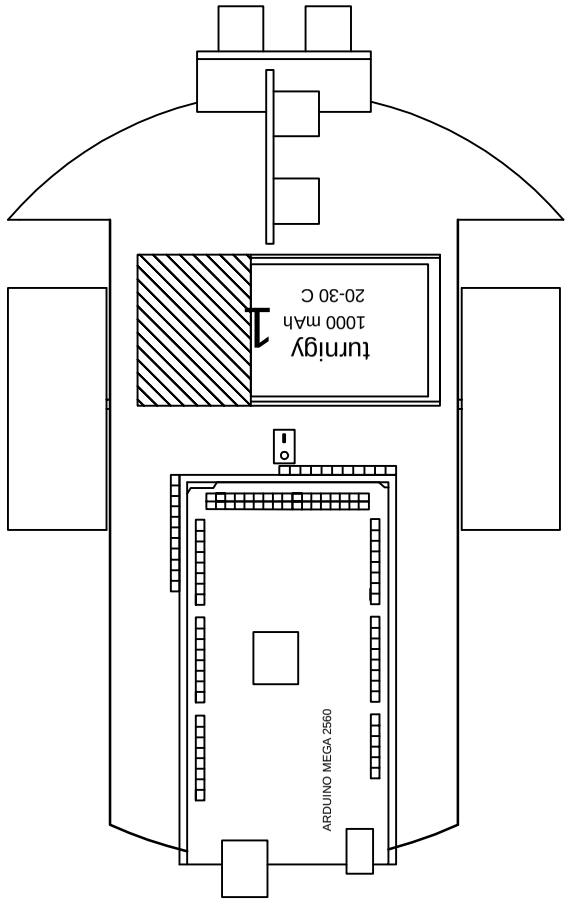
ESCALA: 1:1

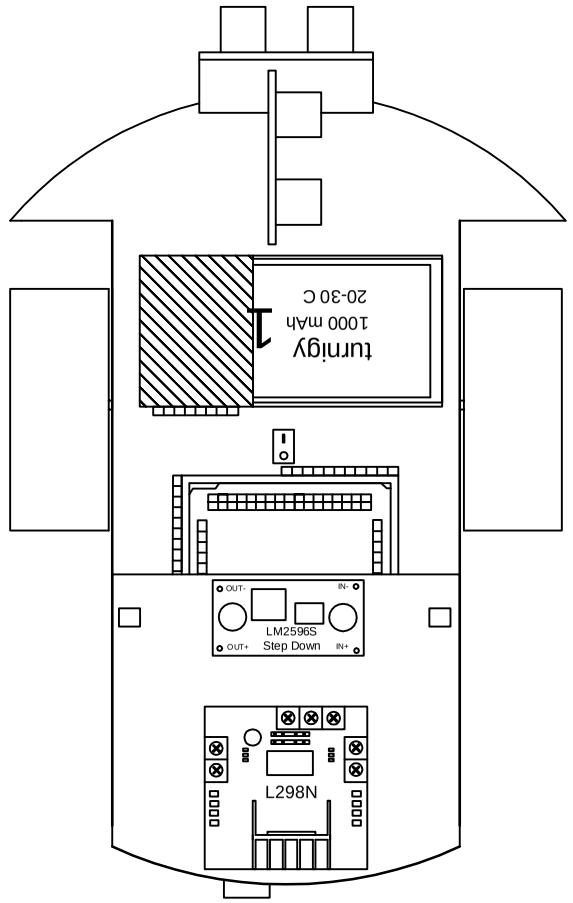
AUTOR:

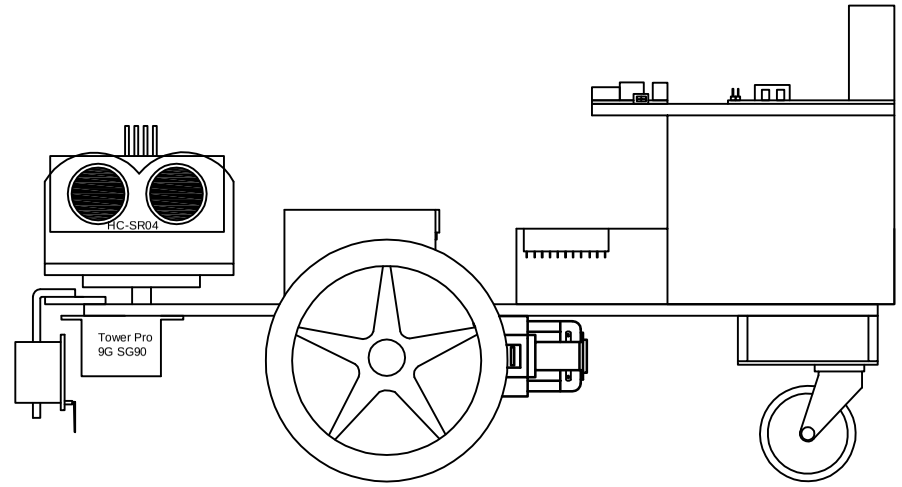
JUAN RAMÓN COSTA RODRÍGUEZ

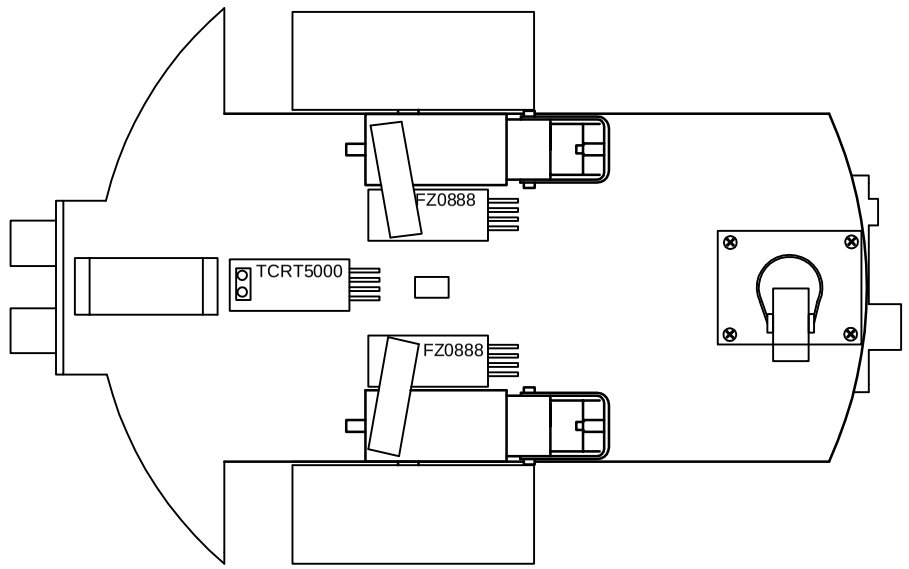
FIRMA:

PLANO Nº: 2









UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01045

TÍTULO DEL TFG:

OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO

TÍTULO DEL PLANO:

VISTAS MÓDELO DEFINITIVO ROBOT

FECHA: SEPTIEMBRE-2017

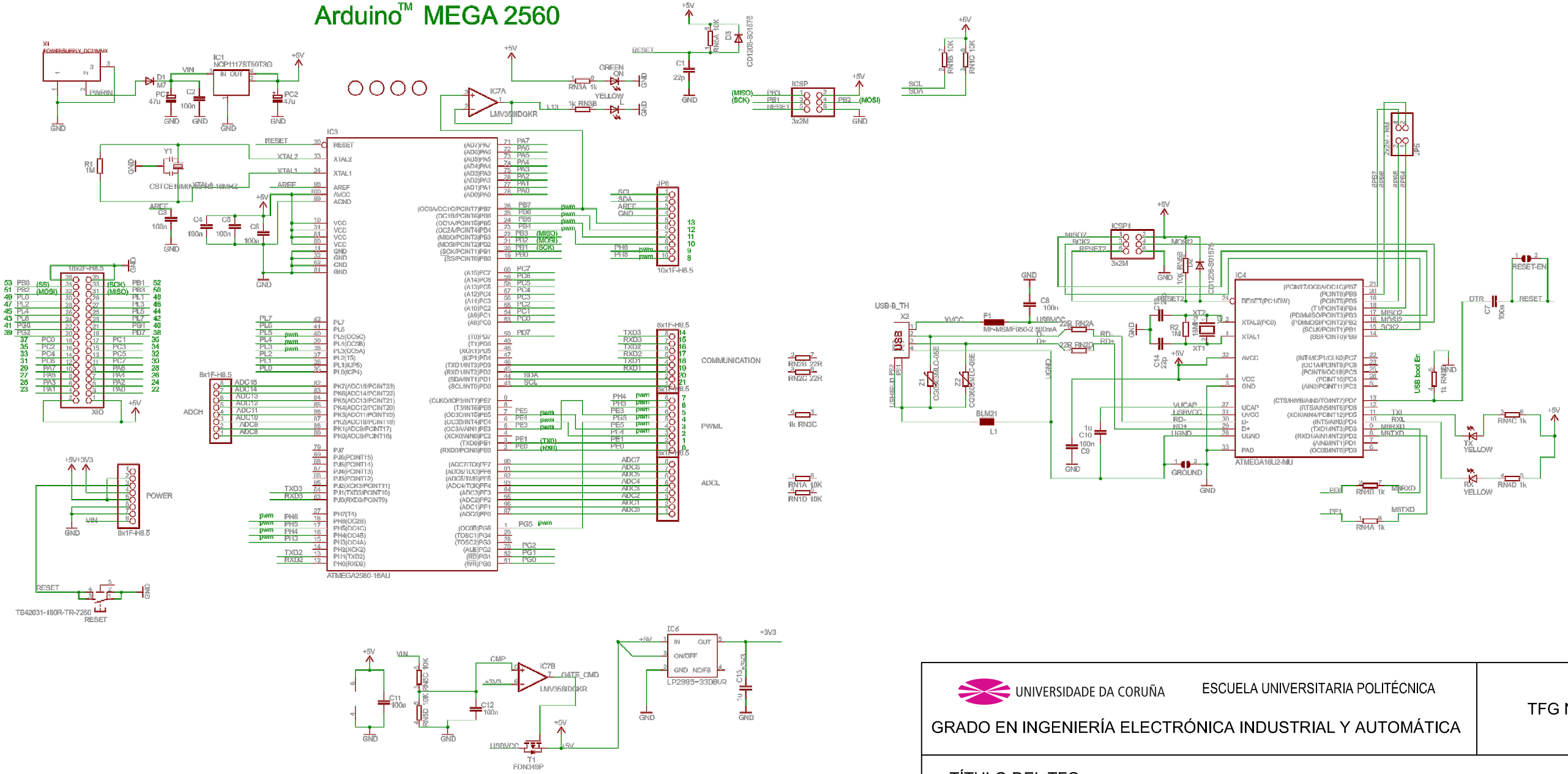
ESCALA: 1:1

AUTOR:

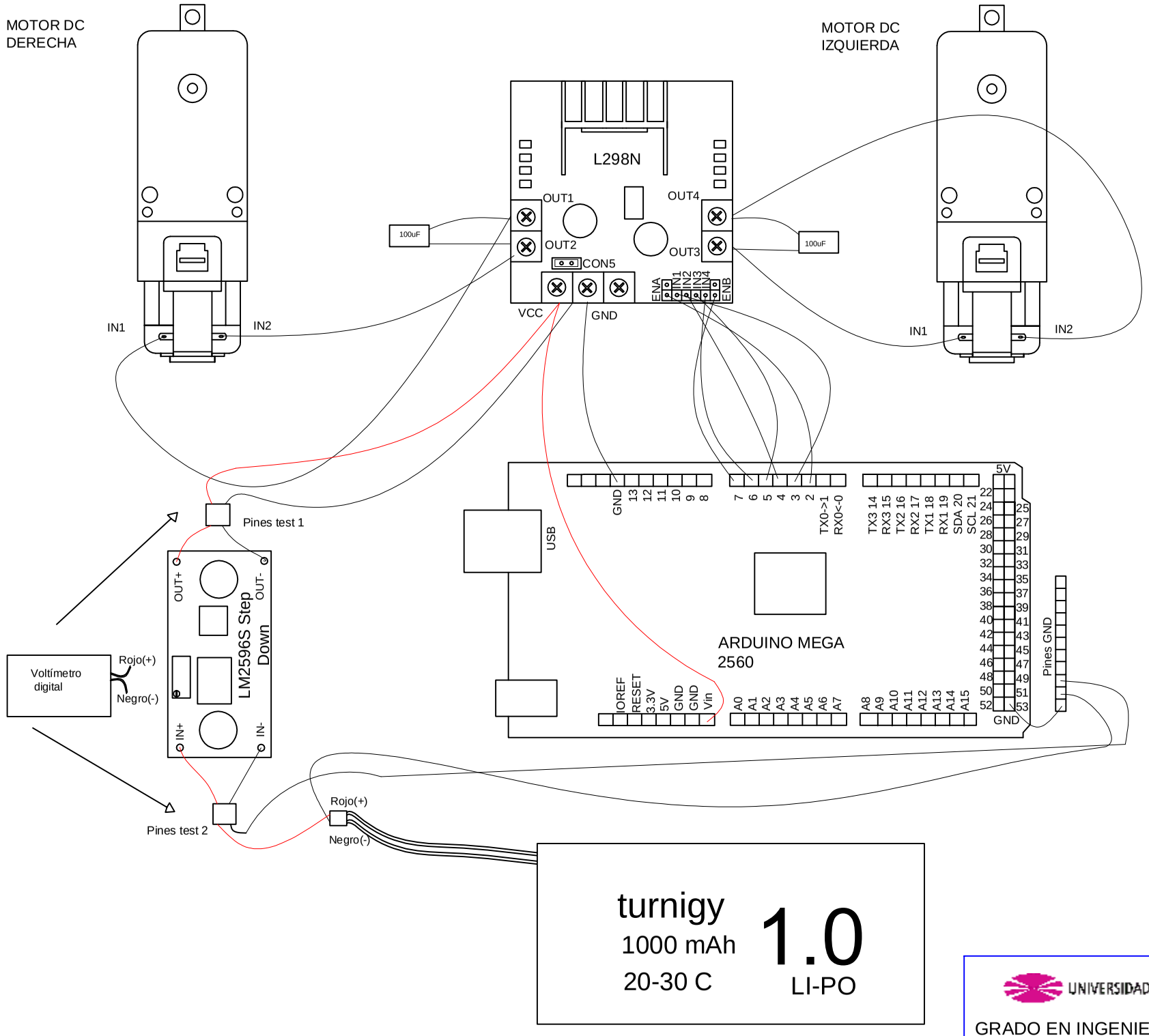
JUAN RAMÓN COSTA RODRÍGUEZ

FIRMA:

PLANO Nº: 3



UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G0104
TÍTULO DEL TFG: OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO	
TÍTULO DEL PLANO: ESQUEMÁTICO ARDUINO MEGA	FECHA: SEPTIEMBRE-2017
	ESCALA: 1:1
AUTOR: JUAN RAMÓN COSTA RODRÍGUEZ — FIRMA:	PLANO Nº: 4



Conexiones eléctricas	
Batería cable rojo (+)	Batería cable rojo (+)
Batería cable negro (-)	Batería cable negro (-)
OUT (+) LM2596S Step Down	VCC L298N
OUT (-) LM2596S Step Down	GND L298N
GND L298N	GND Arduino Mega
Pines test 1	Voltímetro digital. Consulta salida regulador.
Pines test 2	Voltímetro digital. Consulta voltaje de la batería.
ENA L298N	Pin digital 2 Arduino Mega
IN1 L298N	Pin digital 3 Arduino Mega
IN2 L298N	Pin digital 4 Arduino Mega
IN3 L298N	Pin digital 5 Arduino Mega
IN4 L298N	Pin digital 6 Arduino Mega
ENB L298N	Pin digital 7 Arduino Mega
OUT1 L298N	IN1 Motor DC Derecha
OUT2 L298N	IN2 Motor DC Derecha
OUT3 L298N	IN1 Motor DC Izquierda
OUT4 L298N	IN2 Motor DC Izquierda

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01045

TÍTULO DEL TFG:

OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO

TÍTULO DEL PLANO:

CONEXIONADO 1

FECHA: SEPTIEMBRE-2017

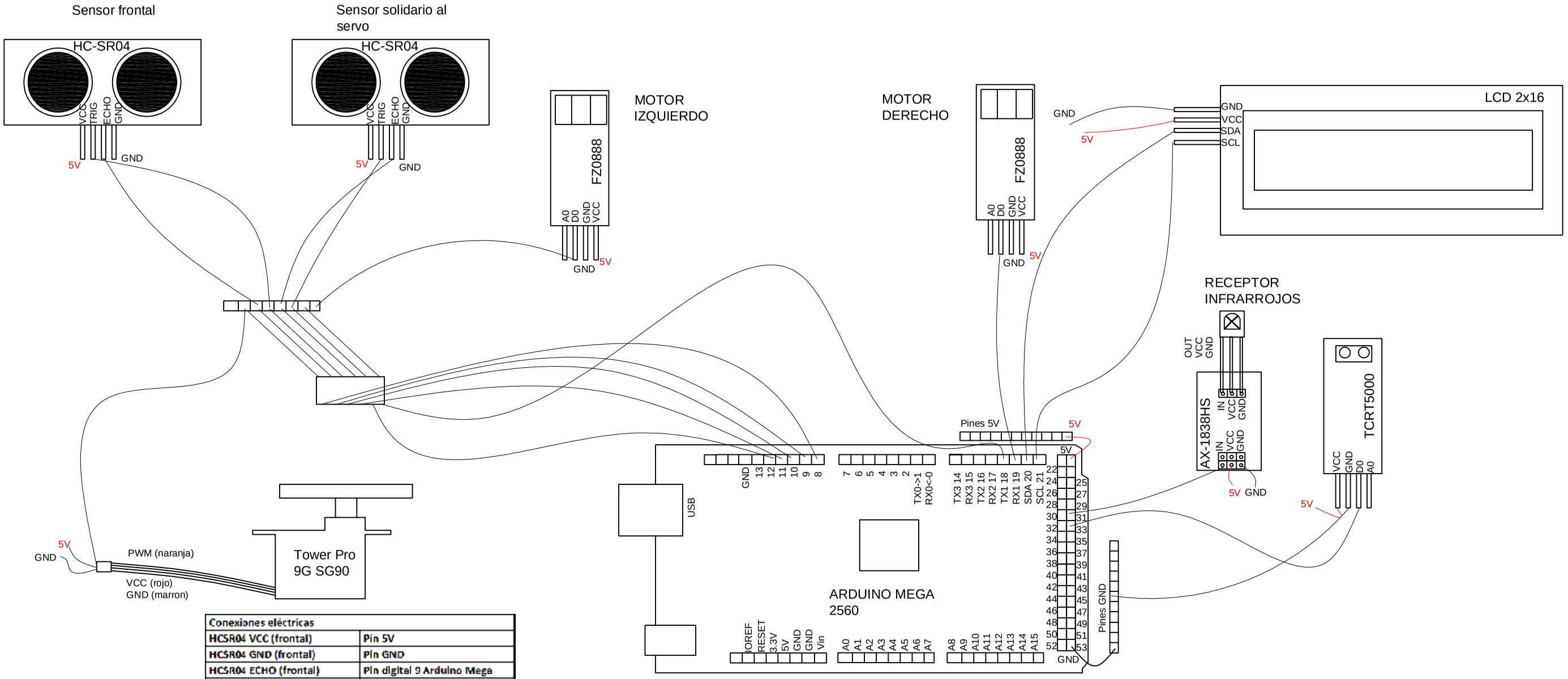
ESCALA: 1:1

AUTOR:

JUAN RAMÓN COSTA RODRÍGUEZ

FIRMA:

PLANO Nº: 5



Conexiones eléctricas	
HCSR04 VCC (frontal)	Pin 5V
HCSR04 GND (frontal)	Pin GND
HCSR04 ECHO (frontal)	Pin digital 9 Arduino Mega
HCSR04 TRIG (frontal)	Pin digital 10 Arduino Mega
HCSR04 VCC (solidario servo)	Pin 5V
HCSR04 GND (solidario servo)	PinGND
HCSR04 ECHO (solidario servo)	Pin digital 11 Arduino Mega
HCSR04 TRIG (solidario servo)	Pin digital 12 Arduino Mega
Servo Tower Pro 9G SG90 PWM(naranja)	Pin digital 8 Arduino Mega
Servo Tower Pro 9G SG90 VCC(roja)	Pin 5V
Servo Tower Pro 9G SG90 GND(marron)	Pin GND
FZ088 A0 (Izquierdo)	-
FZ088 D0 (Izquierdo)	Pin digital 18 Arduino Mega
FZ088 GND (Izquierdo)	Pin GND
FZ088 VCC (Izquierdo)	Pin 5V
FZ089 A0 (Derecho)	-
FZ088 D0 (Derecho)	Pin digital 19 Arduino Mega
FZ088 GND (Derecho)	Pin GND
FZ088 VCC (Derecho)	Pin 5V
Receptor infrarrojos OUT	AX-1838HS IN
Receptor infrarrojos VCC	Pin 5V
Receptor infrarrojos GND	Pin GND
AX-1838HS IN	Pin digital 31 Arduino Mega
AX-1838HS VCC	Pin 5V
AX-1838HS GND	Pin GND
TCRT5000 VCC	Pin 5V
TCRT5000 GND	Pin GND

UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01045

TÍTULO DEL TFG:

OPTIMIZACIÓN DEL ALGORITMO DE UN ROBOT TIPO LABERINTO

TÍTULO DEL PLANO:

CONEXIONADO 2

FECHA: SEPTIEMBRE-2017

ESCALA: 1:1

AUTOR:

JUAN RAMÓN COSTA RODRÍGUEZ

FIRMA:

PLANO Nº: 6

TÍTULO: **OPTIMIZACIÓN DEL ALGORTIMO DE UN ROBOT TIPO LABERINTO**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**
AVDA. 19 DE FEBRERO, S/N
15405 – FERROL

FECHA: **SEPTIEMBRE 2017**

AUTOR: **JUAN RAMÓN COSTA RODRÍGUEZ**

Fdo.: Juan Ramón Costa Rodríguez

5.- PLIEGO DE CONDICIONES

El pliego de condiciones tiene como misión establecer las condiciones técnicas, económicas, administrativas y legales para que el objeto del TFG pueda materializarse en las condiciones especificadas, evitando posibles interpretaciones diferentes de las deseadas.

5.1.- ESPECIFICACIONES DE MATERIALES Y ELEMENTOS CONSTITUTIVOS

5.1.1.- Listado completo de materiales

Los materiales y componentes empleados para el montaje del presente proyecto serán aquellos que cumplan con las especificaciones y las necesidades del proyecto, procurando, además, incurrir en el mínimo gasto posible.

A continuación se listarán todos aquellos materiales que intervienen a lo largo del proyecto para la construcción del robot definitivo. No se listan aquellos materiales que, interviniendo en algún momento en la elaboración del proyecto, no contribuyen a la construcción del modelo definitivo.

Los materiales y componentes utilizados para la elaboración del proyecto serán mencionados a continuación:

- Kit Smart Robot Car.
- Sensores ultrasonidos HCSR04 (2).
- Micro Servo 9g SG90 de Tower Pro.
- Batería Turnigy 1.0 1000 mAh 20-30C.
- Cargador batería modelo B3AC 2S 3S 7.4V 11.1V Power Lipo RC de la marca Imax.
- Módulo sensor TCRT5000 (2).
- Motores DC (2).
- Cinta aislante.
- Arduino Mega 2560.
- Optointerruptor Modulo Sensor FZ0888 (2).
- Convertidor DC DC LM2596S.
- Tira de pines hembra.
- Plancha metacrilato 2 mm de espesor.

- Controlador de motores L298N
- Rollo de cable (2).
- Cables macho hembra
- Condensadores 1uF (2).
- Receptor IR AX1838HS.
- Mando a distancia Samsung.
- Loctite 5 ml.
- Cúter.
- Soldador eléctrico 30 W.
- Flux Liquido RF800 100ml.
- Rollo estaño 60/40 SN/PB PLOMO 2.5% Flux 10g 1mm.
- Polímetro.
- LCD

Los materiales utilizados para la construcción del laberinto serán mencionados a continuación:

- Planchas de madera 100X60X1 cm (3).
- Sierra de calar.
- Escuadra.
- Puntas acero.
- Cola especial madera.
- Tornillos.
- Lijadora.

Los componentes que constituirán al robot serán:

- Kit Smart Robot Car.
- Sensores ultrasonidos HCSR04 (2).
- Micro Servo 9g SG90 de Tower Pro.
- Batería Turnigy 1.0 1000 mAh 20-30C.
- Cargador batería modelo B3AC 2S 3S 7.4V 11.1V Power Lipo RC de la marca Imax.
- Arduino Mega 2560.
- Optointerruptor Modulo Sensor FZ0888 (2).

- Convertidor DC DC LM2596S.
- Controlador de motores L298N.
- Módulo sensor TCRT5000 (2).
- Motores DC (2).
- Condensadores electrolíticos 100uF (2).

5.1.2.- Calidades mínimas a exigir de los materiales

Todos los componentes utilizados que conformen el robot definitivo deberán de tener obligatoriamente el certificado CE que garantice el cumplimiento de las directivas europeas de calidad.

Como se explicó en el análisis de soluciones, el aspecto económico tiene mucho peso a la hora de justificar la elección de los componentes que conforman el robot, por lo que no se recomienda, por motivos económicos, adquirir otros modelos de componentes diferentes a los seleccionados, ya que no se podría garantizar su correcto funcionamiento y en la solución ofrecida sí. Además el ahorro sería mínimo.

Por otro lado queda a elección del futuro lector y proyectista de un proyecto similar a este, elegir otros modelos más caros y de mayor calidad. Cabe recalcar que se recomienda adquirir unos motores DC de mayor calidad. En la elaboración del proyecto se utilizaron 8 motores DC, ya que se estropean con suma facilidad. No se adquirieron motores de mayor calidad por cuestiones de presupuesto, pero sabiéndolo desde un principio, sin duda, es mejor adquirir motores más robustos y fiables. Si se realiza esto habría que recalcular ciertos aspectos de este proyecto, pero simplemente sería jugar con el voltaje de alimentación de los motores variando el potenciómetro del regulador y con la velocidad que se les indica por programa a
través de los pines PWM.

5.1.3.- Pruebas y ensayos de verificación

Queda a expensas del usuario cuidar la batería y alargar su vida útil. Para ello se le proporciona la posibilidad de medir la tensión de la misma de forma

rápida y sencilla, como ya se explicó en la guía del usuario, para que la cargue cuando sea preciso. Se recomienda empezar a realizar medidas del voltaje con relativa frecuencia a partir de la media hora de funcionamiento. La tensión mínima recomendada es de 10.8 V, por debajo de esta tensión no se asegura ni el correcto funcionamiento del robot, ni que no se estropee la misma.

Ante cualquier error de funcionamiento del sistema con los códigos proporcionados en este documento, en primer lugar se recomienda medir el voltaje de la batería, aunque este no debería de ser el motivo si se cuida la batería como se explicó en el párrafo anterior. Si el motivo no es este debe comprobarse el voltaje de salida del convertidor DC DC y llevar a cabo la regulación pertinente si el problema es este. Si no es este el problema se recomienda apagar y encender de nuevo el sistema y, si aún persiste, llevar a cabo una serie de comprobaciones para asegurar el funcionamiento de los componentes, cuyo orden de ejecución quedaría a elección del usuario, ya que el mismo puede intuir, o ver claramente, de donde viene el problema.

Las comprobaciones consistirán en la elaboración de un código solamente con las instrucciones necesarias para controlar uno de los módulos conectados a Arduino. En este documento se detalla a la perfección, de forma clara y sencilla. cómo se maneja cada uno de los componentes, por lo que no debería de haber ningún problema para elaborar códigos básicos de comprobación siguiendo los pasos descritos en este documento. Se cargará este código en el Arduino y se llevarán a cabo las pruebas necesarias para determinar si el funcionamiento es el correcto. En caso de no obtener los resultados deseados se recomienda sustituir el componente que falla por otro similar.

TÍTULO: **OPTIMIZACIÓN DEL ALGORTIMO DE UN ROBOT TIPO LABERINTO**

ESTADO DE MEDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**
AVDA. 19 DE FEBRERO, S/N
15405 – FERROL

FECHA: **SEPTIEMBRE 2017**

AUTOR: **JUAN RAMÓN COSTA RODRÍGUEZ**

Fdo.: Juan Ramón Costa Rodríguez

6.- ESTADO DE MEDICIONES

El Estado de Mediciones tiene como misión definir y determinar las unidades de cada partida que configuran la totalidad del producto, obra, instalación, servicio o software (soporte lógico) objeto del TFG. Las mismas pueden consultarse en el apartado 5.1.1. De todas formas se incluirá en este apartado de nuevo una la lista de todos los componentes que intervienen en el proyecto, pero ahora solo se incluirán aquellos componentes que fueron necesarios adquirirlos para elaborar el proyecto y que formarán parte del presupuesto.

6.1.- UNIDADES QUE CONFORMAN EL ROBOT

- Kit Smart Robot Car.
- Sensores ultrasonidos HCSR04 (2).
- Micro Servo 9g SG90 de Tower Pro.
- Batería Turnigy 1.0 1000 mAh 20-30C.
- Cargador batería modelo B3AC 2S 3S 7.4V 11.1V Power Lipo RC de la marca Imax.
- Arduino Mega 2560.
- Optointerruptor Modulo Sensor FZ0888 (2).
- Convertidor DC DC LM2596S.
- Controlador de motores L298N.
- Módulo sensor TCRT5000 (2).
- Motores DC (2).
- Condensadores electrolíticos 1uF (2).
- Plancha metacrilato 2 mm de espesor.
- Receptor IR AX1838HS.
- Soporte Sensor ultrasonidos (2).

6.2.- UNIDADES QUE INTERVIENEN EN LA CONSTRUCCIÓN DEL ROBOT

- Cinta aislante.
- Tira de pines hembra (2).
- Rollo de cable (2).

- Cables macho hembra
- Condensadores 100uF (2).
- Loctite 5 ml.
- Soldador eléctrico 30 W.
- Flux Liquido RF800 100ml.
- Rollo estaño 60/40 SN/PB PLOMO 2.5% Flux 10g 1mm.

6.3.- UNIDADES QUE CONFORMAN EL LABERINTO

- Planchas de madera 100X60X1 cm (3).
- Puntas acero.
- Cola especial madera.
- Tornillos.

6.4.- DEFINICIÓN DE COMPONENTES DE LA SOLUCIÓN FINAL

Las características, modelo y dimensiones de cada componente que conforma el robot objeto del TFG pueden consultarse en los apartados 2.4.1. y 2.4.2. de este documento.

TÍTULO: **OPTIMIZACIÓN DEL ALGORTIMO DE UN ROBOT TIPO LABERINTO**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**
AVDA. 19 DE FEBRERO, S/N
15405 – FERROL

FECHA: **SEPTIEMBRE 2017**

AUTOR: **JUAN RAMÓN COSTA RODRÍGUEZ**

Fdo.: Juan Ramón Costa Rodríguez

7.- PRESUPUESTO

El presupuesto tiene como misión determinar el coste total de la elaboración del TFG. En el mismo no se tendrán en cuenta aquellos componentes que intervienen en la elaboración del mismo pero que por diferentes motivos no forman parte de la solución final adoptada.

Cabe destacar que aunque no se recoge ni en estado de mediciones ni en el presupuesto, las pilas supusieron un gran costo durante las primeras fases del proyecto.

7.1.- PRESUPUESTO DE MATERIALES

Componente	Precio
Kit Smart Robot Car.	14 €
Sensores ultrasonidos HCSR04 x 2	3,2 €
Soporte sensores ultrasonidos x 2	2,68 €
Micro Servo 9g SG90 de Tower Pro.	3,2 €
Batería Turnigy 1.0 1000 mAh 20-30C.	14 €
Arduino Mega 2560.	43 €
Optointerruptor Modulo Sensor FZ0888 x 2	4 €
Convertidor DC DC LM2596S.	2,5 €
Controlador de motores L298N.	4 €
Módulo sensor TCRT5000 (2).	3 €
Condensadores electrolíticos 100uF x2	1,5 € (Paquete de 10)
Plancha metacrilato 2 mm de espesor.	14 €
Receptor IR AX1838HS.	1,5 €

Tira de pines hembra x 2	1 €
LCD + Adaptador I2C	3,49 €
TOTAL ROBOT	**131,41 €**

Tabla 7.1.1.- Coste de componentes del robot.

Componente	Precio
Cinta aislante.	1,5 €
Rollo de cable 3m (2).	2,5 €
Cables macho hembra	2,85 €
Loctite 5 ml.	6 €
Soldador eléctrico 30 W.	7 €
Flux Liquido RF800 100ml.	2,5 €
Rollo estaño 60/40 SN/PB PLOMO 2.5% Flux 10g 1mm.	1,43 €
Cargador batería modelo B3AC 2S 3S 7.4V 11.1V Power Lipo RC de la marca Imax.	18 €
TOTAL	**41,78 €**

Tabla 7.1.2.- Precio de las unidades que intervienen en la construcción del robot.

El costo total de los materiales asciende a **173,19 €.**

7.2.- PRESUPUESTO RECURSOS HUMANOS.

Concepto	Precio/Hora	Nº horas	Total

Ingeniería	20 €	400	8000 €
Montaje	15 €	20	300 €
TOTAL			**8300** €

Tabla 7.2.1.- Presupuesto de recursos humanos.

7.3.- PRESUPUESTO FINAL

El coste total del proyecto asciende a **8.473,19 €.**

7.4.- PRECIO DE VENTA

El precio de venta de este robot sería bastante elevado, pero una vez que se conocen los componentes necesarios podrían adquirirse en grandes cantidades, reduciendo considerablemente el precio de coste que conllevan. Además las unidades que intervendrían en la construcción de los robots se aprovecharían para la construcción de varias unidades, por lo que también se reduciría considerablemente el coste que esto suponía. El coste correspondiente al montaje también se reduciría enormemente, ya que se conocerían los mecanismos de montaje una vez está construido el primer modelo.

Los costes correspondientes a la ingeniería serían únicos y supondrían un único gasto, no se volvería incurrir en ellos, a no ser que llevaran a cabo mejoras.