ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO INDUSTRIAL

Título del proyecto:

DISEÑO Y CONSTRUCCIÓN DE UN ROBOT SIGUE LÍNEAS CONTROLADO A DISTANCIA CON INTERFAZ ANDROID

José Luis Pérez Motos

Javier Goicoechea Fernández

Pamplona, 21 de febrero

ÍNDICE

Lista de figuras

Lista de tablas

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS

1.1. INTRODUCCIÓN

El presente proyecto abarca el diseño, construcción y configuración de un robot sigue líneas, así como la configuración de una vía de comunicación inalámbrica mediante un enlace Bluetooth y un dispositivo Android. Como su propio nombre indica, un robot sigue líneas es un dispositivo móvil capaz de seguir una trayectoria marcada en el suelo de forma autónoma.

El robot deberá contar con una serie de sensores para deducir en qué posición se encuentra el robot en relación al camino que debe seguir. Para ello el sistema será gobernado por un microcontrolador, el cual tendrá un algoritmo de control que será el encargado de variar la velocidad de los motores para corregir la posición del robot.

Como se ha comentado anteriormente, el robot se podrá comunicar con otros dispositivos a través de tecnología bluetooth.

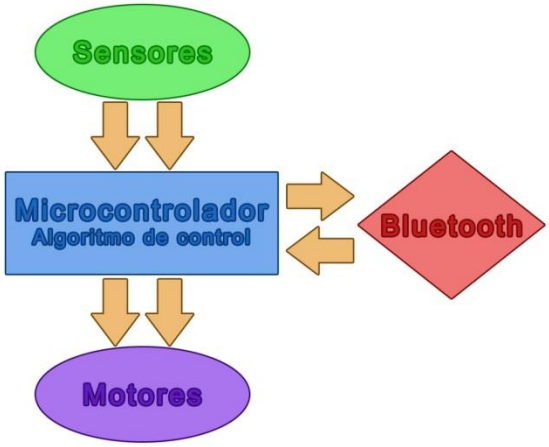


Figura 1-1: Diagrama de todo el sistema

1.2. OBJETIVOS

El objetivo último es dotar al robot de lo necesario para que pueda seguir una trayectoria sin desviarse de ella. Para ello se deberán conseguir otros objetivos:

1. Manipulación de sensores.
2. Control de motores.
3. Diseño de un algoritmo de control.
4. Programación de un microcontrolador.
5. Comunicación entre el sistema y dispositivos externos mediante bluetooth.

CAPÍTULO 2. ROBOT SIGUE LÍNEAS

En las competiciones de robots existen tres modalidades distintas de robots sigue líneas. Todas ellas comparten la forma en la que se presenta la trayectoria: el camino que debe seguir el robot está marcado por una línea negra sobre un fondo blanco. Sin embargo, se diferencian en el tipo de trayectoria que deben seguir los robots así como del objetivo final que debe alcanzar. Por lo tanto, tanto las características como la programación del robot difieren de una modalidad a otra. A continuación se va a describir cada una de estas modalidades:

- Laberinto

En esta modalidad los robots deben resolver un laberinto lo antes posible. Para ello se les permite dar dos vueltas. La primera de ellas es de reconocimiento. En ella el robot deberá grabar en su memoria interna los giros que debe de realizar para que en la segunda vuelta complete el recorrido en el menor tiempo posible.

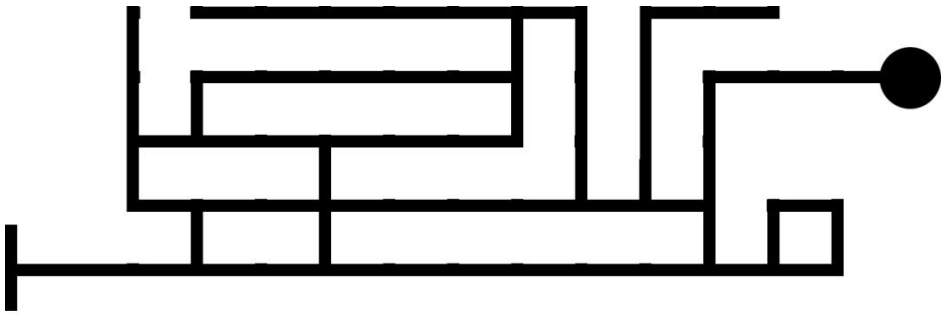


Figura 2-1: Imagen de un circuito tipo laberinto

Como se puede ver en la imagen anterior estos circuitos se caracterizan porque todos los giros que presenta la trayectoria son de 90°. Esto hace que la programación del robot sea más sencilla ya que solamente habrá que programarlo para que realice giros de 90° y de 180°.

Por otro lado, el punto interesante de esta modalidad, en cuanto a programación se refiere, está en cómo almacenar la información del camino correcto que debe tomar el robot.

- Velocistas

En esta modalidad los robots realizan una carrera de persecución por parejas dentro de un circuito cerrado. Los robots comienzan en puntos opuestos del trazado y ambos avanzan en el mismo sentido. La prueba acaba cuando uno de los dos llega a contactar con el otro.

Generalmente existe una primera ronda en la que los participantes recorren el circuito en solitario y solo los que consigan mejor tiempo pasarán a la siguiente fase de la competición.

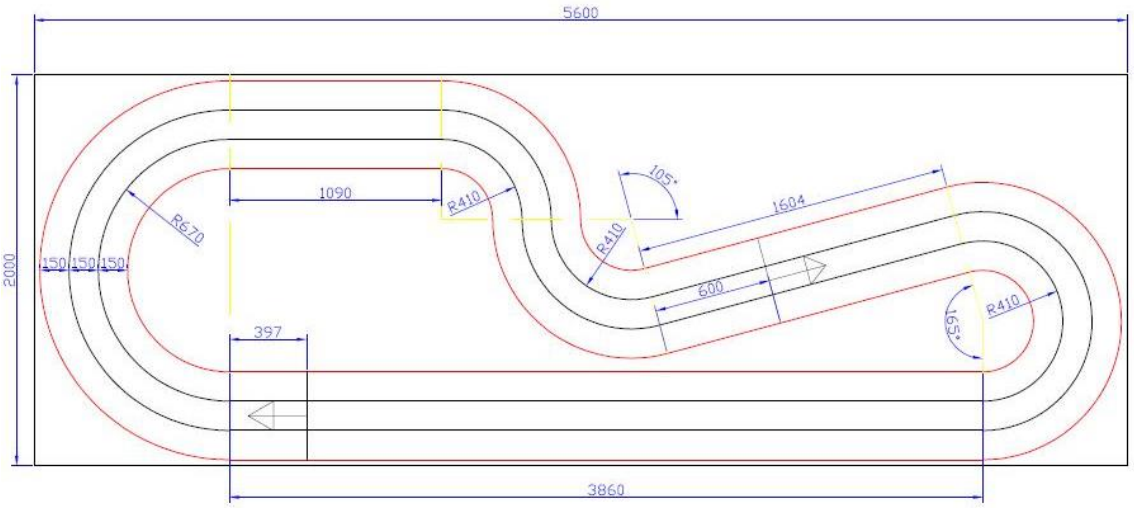


Figura 2-2: Imagen de un circuito para robots velocistas

Como se observa en la imagen anterior el este circuito es bastante diferente al laberinto. En este circuito no hay intersecciones, pero sí que presenta curvas, por lo que, en este sentido, la programación del robot será más compleja que en el caso del laberinto ya que se deberá de regular la velocidad de los motores para tomar las curvas de forma correcta.

Los robots velocistas no están obligados a mantenerse sobre una línea, sino que los robots pueden elegir por cuál de cualquiera de las dos líneas ir o incluso pueden ir entre ambas. El robot quedará descalificado si toca cualquier línea roja.

- Rastreadores

Esta modalidad es una mezcla de las dos anteriores. Los robots deberán completar un circuito lo antes posible, pero este circuito está lleno de curvas y bifurcaciones.

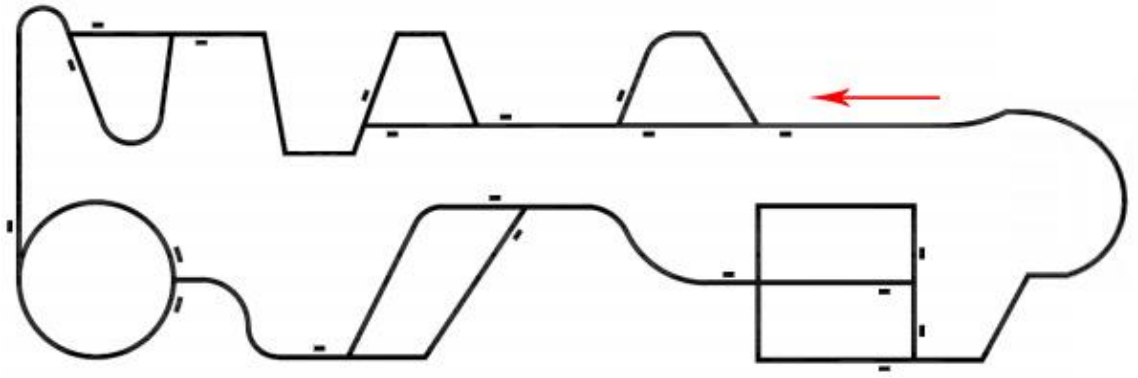


Figura 2-3: Imagen de un circuito para robots rastreadores

Si se observa la imagen anterior se pueden apreciar una serie de pequeñas marcas en los lados del circuito. Estas líneas marcan la dirección que deberá tomar el robot en la siguiente intersección. Si antes de la intersección se encuentra una marca a la izquierda del trayecto en la siguiente intersección deberá girar a la izquierda. Análogamente, si se encuentra la marca a la derecha, en la siguiente intersección deberá girar a la derecha. Sin embargo, si se encuentran marcas en ambos lados de la trayectoria en la siguiente intersección deberá continuar recto.

De esta forma esta modalidad recoge las dificultades de las otras dos, ya que el robot debe seguir una determinada dirección en las intersecciones y además debe poder regular la velocidad de los motores debido a que el circuito presenta curvas.

Una vez analizadas las tres modalidades existentes de robots sigue líneas se escogió la modalidad de rastreador. Como ya se ha explicado, reúne características de la modalidad de laberinto y de velocista, por lo que la modalidad de rastreador requiere

robots más completos ya que deben combinar una gran capacidad de detección, un buen seguimiento del camino y ejecutar con rapidez las maniobras.

En los siguientes apartados se pasará a analizar en profundidad las características que tienen las pruebas de rastreadores y diferentes modelos de rastreadores ya existentes.

2.1. REGLAMENTO ROBOTS RASTREADORES

En este apartado se comentarán las principales normas que debe cumplir todo robot que quiera participar en la prueba de rastreadores[11].

2.1.1. REQUISITOS DEL ROBOT

- Los robots rastreadores deberán ser **autónomos**. No podrán recibir datos que alteren su comportamiento. La organización permite que los robots envíen datos siempre y cuando se informe a los jueces y homologadores de dicha característica. Como excepción se permite que los robots dispongan de un sistema de detención a distancia, ya sea por infrarrojos o radiofrecuencia, dicho sistema podrá ser controlado por un juez de pista o miembro del equipo.
- El robot deberá tener unas **dimensiones** tales que quepan dentro de una caja de 20x20 cm con los sensores de contacto (microrruptores) plegados en caso de que fuera necesario. Su altura podrá ser cualquiera. No se permite diseñar el robot de forma que cuando comience la carrera se separe en diferentes piezas.
- El **peso** máximo de los robots será de 2000 gramos incluyendo todas las partes.

2.1.2. PISTA

- Las pistas consistirán en una superficie de color blanco con una línea negra (camino a seguir) de una anchura de 20±2mm. Las marcas estarán formadas del mismo material que la línea del circuito, su anchura será la misma que la del camino a seguir y su longitud será de 40±5mm. El espacio entre marcas equivaldrá a 20±3mm.

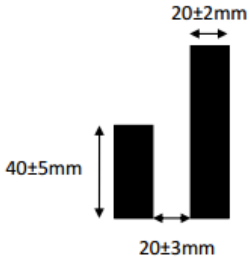


Figura 2-4: Dimensiones de la pista y de las marcas de bifurcación

- El camino a seguir tendrá tantas ramificaciones como la organización crea convenientes.
- El robot deberá seguir siempre una de las líneas de las que conste el camino. En caso de salirse dispondrá de 15 segundos para reincorporarse al circuito a una distancia inferior a 100 mm desde el punto en el que se salió. Si pasado este tiempo no se ha reincorporado o lo ha hecho en otro lugar de la pista, el responsable del equipo tiene la opción de situar el robot en el inicio de la pista sin para el tiempo. En caso contrario quedará eliminado.
- Antes de una bifurcación se marcará el camino que debe tomar el robot con una marca en los laterales del camino.

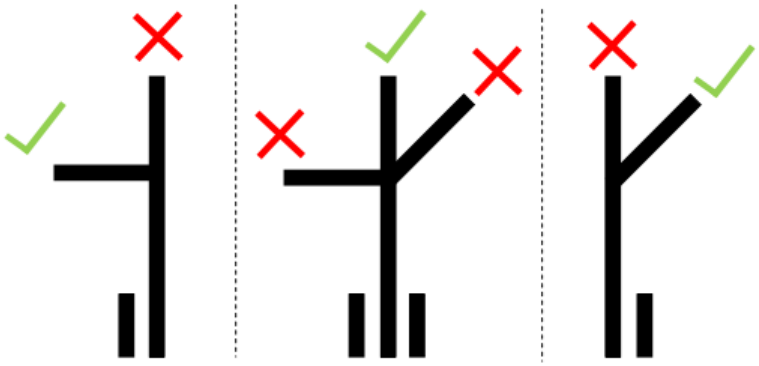


Figura 2-5: Esquema de las bifurcaciones

Partiendo de una situación correcta, en una bifurcación podrán coexistir como máximo 3 posibles caminos, formando entre ellos los siguientes ángulos:

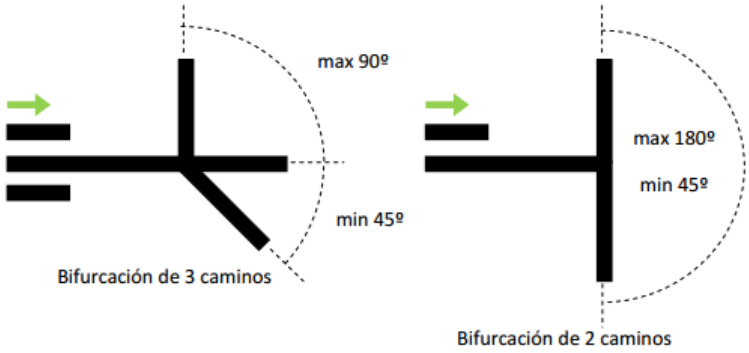


Figura 2-6: Ángulos máximos que pueden aparecer

2.1.3. PRUEBAS DE RASTREADOR

- En el caso de que dos robots compitan en pistas simétricas, cada uno recorrerá una de las dos pistas, luego los robots se cambiarán de pista y se sumarán los tiempos de cada turno. De cada carrera e dos robots se clasificará el más rápido. Quedará a decisión de la organización, según el número de robots inscritos, si el sistema de competición será estilo liguilla o campeonato con eliminación de la competición de los robots no clasificados.
- En caso de que solo se habilite una pista, se permitirá hacer varias rondas (a definir por la organización), ofreciendo así la posibilidad de mejorar los tiempos marcados. Se contabilizará el mejor tiempo de los obtenidos. La clasificación será por tiempo.
- Pasado un tiempo mínimo de tres minutos desde el inicio del asalto los jueces podrán parar y finalizar la carrera cuando lo consideren oportuno. Quedará bajo criterio del juez el robot ganador de la carrera, teniendo en cuenta los siguiente parámetros de decisión:

    1. Ser el único robot que permanezca en movimiento.

    2. Máxima proximidad a la llegada.

    3. No haber perdido la línea en ninguna ocasión durante la carrera.

4. Si todos los robots de la carrera se quedan parados o "colgados", quedará a decisión del juez clasificar a un robot de otra carrera que haya demostrado ser más competente. La decisión del juez será inapelable en ambos casos.

2.2. ANTECENDETES

A continuación se procederá a describir diferentes robots rastreadores que se han ido encontrando en la bibliografía. Con este análisis se pretende recoger ideas para el diseño del robot, así como analizar las fortalezas y debilidades de estos diseños previos.

2.2.1. ROBOT SEGUIDOR DE LÍNEAS DE LA UPNA

Este robot fue el resultado de otro Proyecto Final de Carrera realizado por estudiantes de la Universidad Pública de Navarra[6].

Figura 2-7: Robot seguidor de línea desarrollado en otro proyecto de la UPNA

Como microcontrolador emplea un Arduino Duemilanove y emplea dos motores de corriente continua que son controlados por una tarjeta controladora (Motor Shield) especialmente diseñada para ser usado por placas Arduino. Así que en este proyecto no se aborda la parte del control de potencia de los motores, sino que esa parte del proyecto descansa en una tarjeta comercial Para detectar la línea emplea una matriz de seis sensores. Cuenta con una rueda loca que sirve como tercer punto de apoyo para darle estabilidad al robot. Por último, todo el sistema está alimentado con dos baterías de lipo de 2000 mAh.

En este proyecto se analizan distintas configuraciones del vehículo autónomo, incluso empleando un sensor de distancia, consiguiendo que el robot detecte y esquive obstáculos en su camino. Una de las variantes estudiada es un robot rastreador sigue líneas análogo al que se construirá en este proyecto.

Sin embargo, tras analizar videos y el código de este robot se puede concluir que no aplica ningún tipo de control avanzado sobre los motores. Simplemente cuando los sensores detectan que el robot se encuentra muy desviado de la línea se fuerza un giro del robot alternando la proporción de giro de las ruedas izquierda y derecha. Se trata de un tipo de control Todo/Nada, sin acción de control regulada de forma continua. Esto hace que el error de posición nunca sea nulo, es decir, en ningún momento el robot va completamente recto, sino que avanza oscilando sobre la línea de referencia.

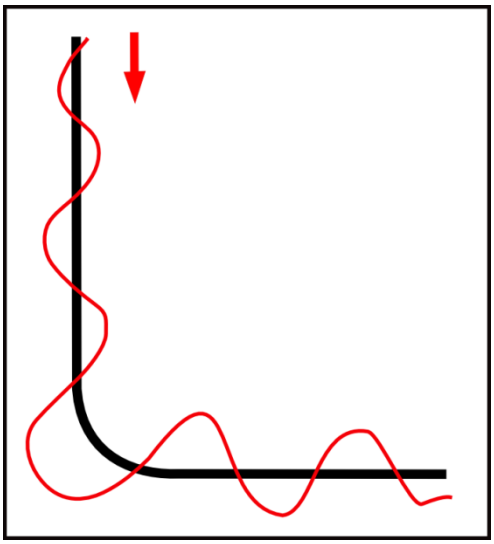


Figura 2-8: Trayectoria que sigue el robot

Por otro lado, al incluir en el diseño del robot una carcasa hace que este resulte mucho más robusto, aunque también lo hace más pesado, lo que tendrá repercusiones en el consumo de energía eléctrica y la autonomía del robot.

2.2.2. POLOLU 3PI

El Pololu 3Pi es un robot comercializado por la empresa estadounidense Pololu diseñado para competiciones de rastreadores y de laberinto[12]. El precio del robot es de 100€ y necesita un programador externo[13].

Figura 2-9: Pololu 3Pi

El microcontrolador de este robot es un ATmega328, el mismo que emplean diversas placas Arduino como la Arduino UNO y la Arduino Pro Mini. Los motores son de corriente continua y emplea cinco sensores infrarrojos situados en diferentes puntos de la periferia del robot para detectar la línea. Dispone de una pantalla LCD que aporta numerosos datos sobre el estado del robot, tales como nivel de batería, velocidad o lectura de los sensores. Por último, la alimentación del todo el sistema se realiza mediante cuatro pilas AAA.

El control de la velocidad y de la desviación de la línea de este robot es muy avanzado. El sistema emplea un control PID para regular la velocidad de los motores en función de las lecturas de los sensores. Además el robot Pololu 3Pi tiene un tamaño muy reducido y poco peso, lo que lo hace extremadamente maniobrable y rápido. De esta forma puede alcanzar velocidades de hasta 1m/s.

Por todas estas características el Pololu 3Pi se va a convertir en una gran referencia a la hora de desarrollar diferentes partes del proyecto, poniendo este robot como ejemplo a superar.

2.2.3. N00B0T

Este robot fue desarrollado en el foro de la Asociación de Robótica y Domótica de España[14] por el usuario dip6, por lo que fue totalmente diseñado y construido por hobby. Como se puede ver en la siguiente imagen, el diseño externo está muy influenciado por el Pololu 3Pi.

Figura 2-10: n00b0t

Al igual que el Pololu 3Pi monta un ATmega328, los motores son de corriente continua, y tiene en diferentes puntos de la periferia del robot 5 sensores infrarrojos para detectar la línea y emplea cuatro pilas AAA para alimentar a todo el sistema. A diferencia del Pololu 3Pi incluye dos sensores infrarrojos extras, uno a cada lado, que tienen como finalidad leer las marcas que aparecen en los circuitos de los rastreadores para indicar qué camino deben tomar.

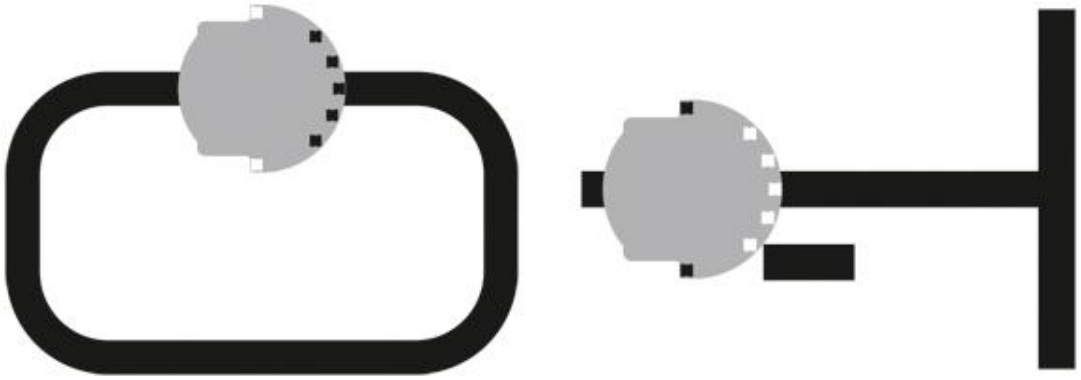


Figura 2-11: Disposición de los sensores de n00b0t

Sin embargo la programación del n00b0t difiere mucho de la programación del Pololu 3Pi. El código, disponible en el blog del creador del n00b0t[15], es muy claro y conciso a diferencia del Pololu 3Pi. Además, para el control de velocidad del robot emplea un control PD que, según los videos que se han podido ver, ofrece grandísimos resultados.

Sin lugar a dudas el n00b0t es un gran robot que aporta bastantes ideas que se podrán emplear en el diseño del robot que se aborda en el presente proyecto.

2.2.4. POCKETBOT 2

El PocketBot 2 es un robot diseñado y construido por Ondřej Staněk, estudiante checo que presentó este robot como Proyecto Final de Carrera[16]. La principal característica de este robot son sus dimensiones, ya que, como el propio Ondřej Staněk dice, es un robot que cabe dentro de una caja de cerillas.

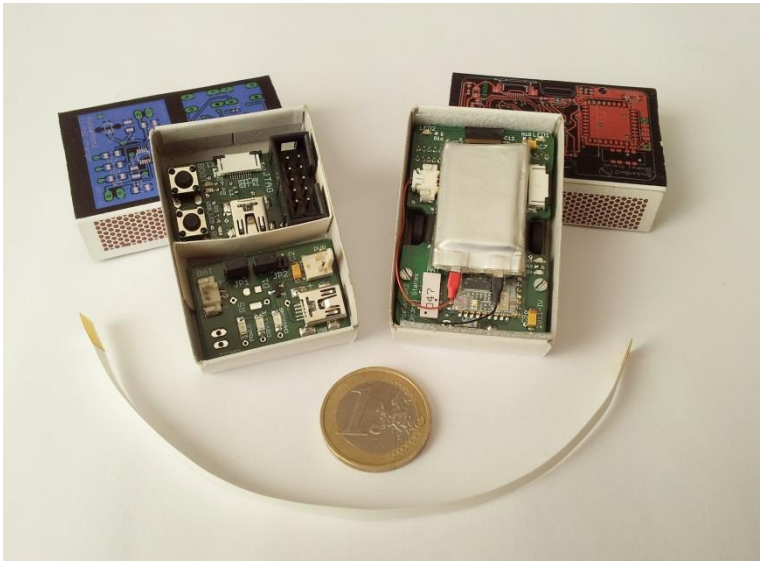


Figura 2-12: Prueba del reducido tamaño del Pocketbot 2

Además de las diminutas dimensiones del robot ya comentadas (48 x 32 x 12 mm) pesa tan solo 20 gramos (incluida la batería) y alcanza velocidades de hasta 70m/s. Emplea un ATxmega128A3 como microcontrolador y motores de corriente continua. Además emplea una matriz de ocho sensores para detectar la línea que debe seguir. Para alimentar todo el sistema el robot cuenta con una batería de lipo de 190mAh.

Otras características de este robot es que emplea un sensor de ángulo de giro de las ruedas, de tipo encoder que le permite calcular la posición del robot en función de la posición inicial de este; dispone de sensores de proximidad para variar la trayectoria del robot en el caso de que haya obstáculos delante de él; cuenta con un sensor RGB con el que en función del color de la línea puede variar la velocidad del robot; dispone de un acelerómetro de tres ejes y puede comunicarse con otros dispositivos mediante tecnología bluetooth.

Además de todas estas grandes características el robot también cuenta con una buena programación. Todo el robot está programado en java, lo que, unido al uso de tecnología bluetooth, le permite una gran interacción con ordenadores y teléfonos móviles. Por último, al igual que el n00b0t, emplea un control PD para controlar la velocidad del robot y al igual que aquel, a la vista de los videos disponibles en su blog, los resultados son excelentes.

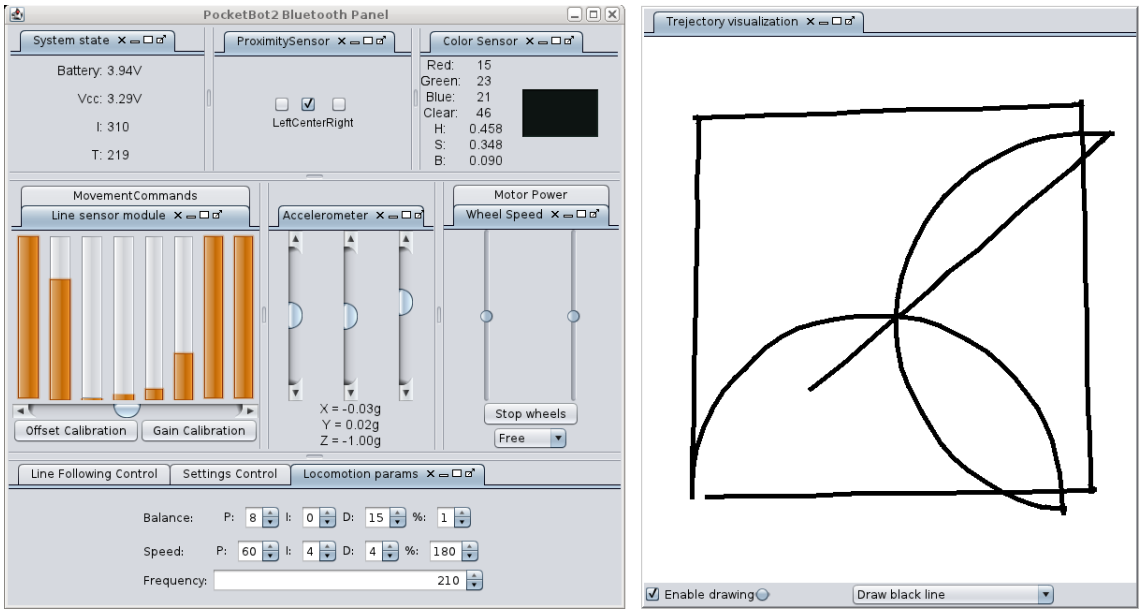


Figura 2-13: Interfaz gráfica de la aplicación de ordenador del Pocketbot 2

El PocketBot 2 es otro gran robot que puede ser de gran utilidad sobre todo en temas de conectividad.

2.2.5. CONCLUSIONES

Todos los robots estudiados incluyen elementos comunes en sus diseños que habrá que tener en cuenta a la hora de elegir los componentes que tenga el robot.

- **Microcontrolador**: la mayoría de los robots estudiados emplean microcontroladores de la familia ATmega. Esto puede ser debido a su bajo coste y sencillez de programación, unido a que esta aplicación en particular no presenta unos requerimientos especiales en cuanto a complejidad de cálculo o de velocidad.
- **Motores**: aunque todos los modelos estudiados empleen motores de corriente continua también existen gran cantidad de modelos que emplean motores paso a paso. Generalmente esta opción permite un mejor control de la posición (ángulo) de las ruedas, pero son bastante más lentos que los motores de corriente continua con reductora.
- **Sensores infrarrojos**: es el sistema que emplean todos los modelos para detectar la línea. Estos sensores lo que hacen es variar la señal que envían al

microcontrolador según si lo que ven es más claro o más oscuro. Existen diferentes variantes, ya que pueden estar en forma de matriz o dispersos por diferentes zonas del robot.

- **Alimentación**: la alimentación en este tipo de robots suele ser a través de pilas o de baterías de LiPO (Li-Polímero).
- **Control velocidad**: los modelos más avanzados emplean controles PID o PD, aunque se han encontrado otras aproximaciones más sencillas que emplean controles de tipo Todo/Nada.

2.3. ROBOTS RASTREADORES PARA USOS DOMÉSTICOS E INDUSTRIALES

Además de los robots sigue líneas explicados con anterioridad cuyo principal criterio de diseño es participar en competiciones de robots, existen diferentes modelos que se emplean en otras aplicaciones. En este apartado se presentarán diferentes robots que siguen el mismo principio que los robots seguidores de línea pero que su utilidad está fuera del de las competiciones de robots.

2.3.1. ROOMBA

Quizás sea el robot rastreador más conocido de todo el mundo. La finalidad de este robot es la de limpiar la casa de forma autónoma.

Figura 2-14: Roomba

Para cumplir con su finalidad dispone de una serie de sensores de proximidad que le permiten delimitar una habitación de la casa. Después con un programa interno crea una ruta que le permite limpiar la habitación de forma óptima. El robot es capaz de cubrir con rutas aparentemente aleatorias toda la superficie de la habitación, y además recordar su posición relativa respecto de la base de carga de baterías. Cuando termina el programa de limpieza, o se va quedando sin batería, Roomba retorna de forma autónoma a su base para recargarse[17].

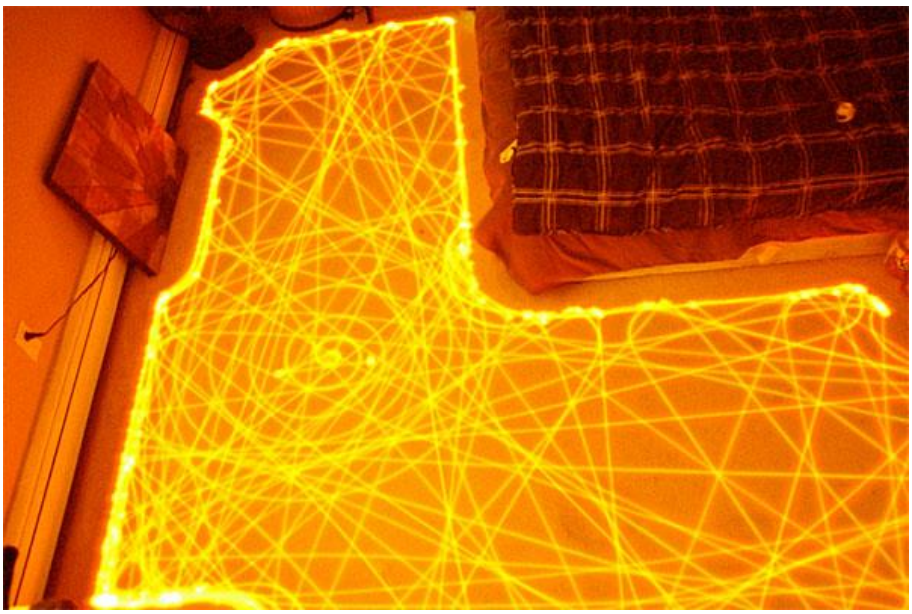


Figura 2-15: Trayectoria seguida por un robot Roomba para limpiar una habitación

2.3.2. KIVA

Este robot es fabricado por Kiva Systems[18] y tiene como finalidad la de organizar almacenes. Multitud de grandes compañías como Amazon, Toys “R” Us, Staples o GAP emplean robots Kiva en sus almacenes.

Figura 2-16: Almacén de Amazon

Como se puede apreciar en la imagen anterior, los robots tienen marcados en el suelo una serie de trayectorias. En realidad se trata de cables conductores enterrados en el suelo, y el sistema de detección y seguimiento en lugar de ser óptico se basa en sensores inductivos. Según a donde tengan que llevar la carga que llevan ese momento, el programa interno decidirá que caminos tomar. Pero estos robots no trabajan solos, sino que puede haber una gran cantidad de ellos para controlar un solo almacén. Es por ello por lo que se necesita una gran sincronización entre todos ellos. Así, estos robots además de incluir sensores para detectar las trayectorias del suelo incluyen diversos sensores para poder comunicarse con el resto de robots. Además, un ordenador central controlado por un operario controla el buen funcionamiento de todo el sistema.

2.3.3. ROBOT TRANSPORTA MEDICAMENTOS

En el hospital Nemocnice Na Homolce[19] de Praga emplean robots seguidores de línea para transportar las medicinas a los pacientes.

Figura 2-17: Hospital Nemocnice Na Homolce de Praga

Como si de un seguidor de líneas normal y corriente se tratara, estos robots tienen marcados a lo largo de todo el hospital marcas en el suelo con las trayectorias que deben seguir. De nuevo, al ser un sistema que implica a varios robots, necesita de un ordenador central que organice a todos los robots.

CAPÍTULO 3. ELECCIÓN DE COMPONENTES

En los siguientes apartados se van a exponer las razones por las que se eligieron los diferentes componentes que forman el robot. En todo momento se tuvieron en cuenta una serie de características que definían al robot. Estas características son:

1. **Pequeño tamaño**: se pensó que el robot fuera más pequeño que el Pololu 3Pi, el cual abarca una superficie de unos 70cm^2.

2. **Ligero**: de nuevo se pensó que el robot fuera más ligero que el Pololu 3Pi, que tiene un peso de 243gr con pilas[20].

3. **Veloz**: se puso como meta superar los 70cm/s del PocketBot 2.

4. **Barato**: se intentó que el coste total del robot fuera inferior a los 100€ del Pololu 3Pi.

3.1. MICROCONTROLADOR

Un microcontrolador es un circuito integrado que está formado por las tres unidades funcionales de un ordenador: microprocesador, memoria y periféricos de entrada y salida[4].

La forma en la que funciona un microcontrolador se determina por el programa almacenado en su memoria. Este programa se puede diseñar y escribir en diferentes lenguajes de programación y tras una compilación, se descarga en la memoria interna del microcontrolador en lenguaje ejecutable. Esto, unido a su alta flexibilidad, hacen que los microcontroladores se empleen en multitud de aplicaciones: automatización, robótica, domótica, medicina, aeronáutica, automoción, telecomunicaciones, etc.

Las principales características de los microcontroladores son:

- Microprocesador: típicamente de 8 bits, pero existen versiones de 4, 32 y hasta 64 bits con arquitectura Harvard, con memoria/bus de datos separada de la

memoria/bus de instrucciones de programa, o arquitectura de von Neumann con memoria/bus de datos y memoria/bus de programa compartida.

- Memoria de Programa: puede ser una memoria ROM (Read Only Memory), EPROM (Electrically Programable ROM), EEPROM (Electrically Erasable/Programable ROM) o Flash. Es la encargada de almacenar el código del programa que ejecutará el microprocesador.
- Memoria de Datos: es una memoria RAM (Random Access Memory) que típicamente puede ser de 1, 2, 4, 8, 16 o 32 kilobytes.
- Generador de Reloj: cristal de cuarzo que produce unos impulsos con una determinada frecuencia y genera una señal oscilante. Esta frecuencia suele ir desde 1 a 40 MHz.
- Interfaz de Entrada/Salida: puertos paralelos, seriales (UARTs, Universal Asynchronous Receiver/Transmitter), I2C (Inter-Integrated Circuit), Interfaces de periféricos seriales (SPIs, Serial Peripheral Interfaces), Red de Área de Controladores (CAN, Controller Area Network), USB (Universal Serial Bus), etc.
- Otras opciones:
  - Conversores Analógicos-Digitales (A/D, analog-to-digital) para convertir un nivel de voltaje en un cierto pin a un valor digital manipulable por el programa del microcontrolador. Estos conversores A/D suelen tener una resolución típica de 10 bits, aunque existen versiones de 12, 16 o 32 bits.
  - Moduladores por Ancho de Pulso (PWM, Pulse Width Modulation) para generar ondas cuadradas de frecuencia fija pero con ancho de pulso variable. Aunque cualquier salida digital del microcontrolador puede ser programada para hacer esta función mediante el uso de interrupciones y temporizadores, muchos microcontroladores incluyen algunas salidas especialmente dedicadas a este efecto, lo cual simplifica su uso.

Para este proyecto desde un primer momento se decidió que el microcontrolador a usar fuera un Arduino ya que se amoldaba perfectamente a las características del proyecto. En los siguientes apartados se realizará una explicación más exhaustiva de las razones por las que se eligió Arduino.

3.1.1. ¿QUÉ ES ARDUINO?

Arduino[21] es una plataforma de hardware y software *open source* que está basado en una placa que permite conectar sensores y actuadores mediante entradas y salidas analógicas y digitales y en un entorno de desarrollo basado en el lenguaje de programación Processing.

Figura 3-1: Logotipo de Arduino

Al ser open source tanto su diseño como su distribución es libre. Es decir, puede utilizarse libremente para desarrollar cualquier tipo de proyecto sin tener que adquirir ningún tipo de licencia. El texto de la referencia de Arduino está publicado bajo la licencia *Creative Commons Reconocimiento-Compartir bajo la misma licencia 3.0*. Los ejemplos de código de la referencia están liberados al domino público.

3.1.2. VENTAJAS

Las razones por las que se eligió Arduino como plataforma sobre la que desarrollar el proyecto fueron cuatro:

- **Barato**: por apenas 20€ se puede conseguir una placa Arduino completamente funcional incluyendo un módulo de interfaz Serie-USB.
- **Popular**: la plataforma Arduino es ampliamente conocida por miles de desarrolladores en todo el mundo, desde profesionales hasta personas que desarrollan aplicaciones por *hobby*. Además existen tutoriales y foros que facilitan notablemente diferentes apartados de los proyectos.
- **Versátil**: una misma placa de Arduino puede servir para proyectos de domótica, robótica, control de sistemas o como programador de otros microcontroladores.

- **Open source**: se encuentran en la misma página web de Arduino los planos y esquemas de las diferentes placas, por lo que se puede modificar cualquier componente como se prefiera.

3.1.3. PARTES DE UNA PLACA ARDUINO

Para explicar las principales partes que componen una placa Arduino se va a utilizar como modelo la Arduino UNO rev3. Las principales partes son:

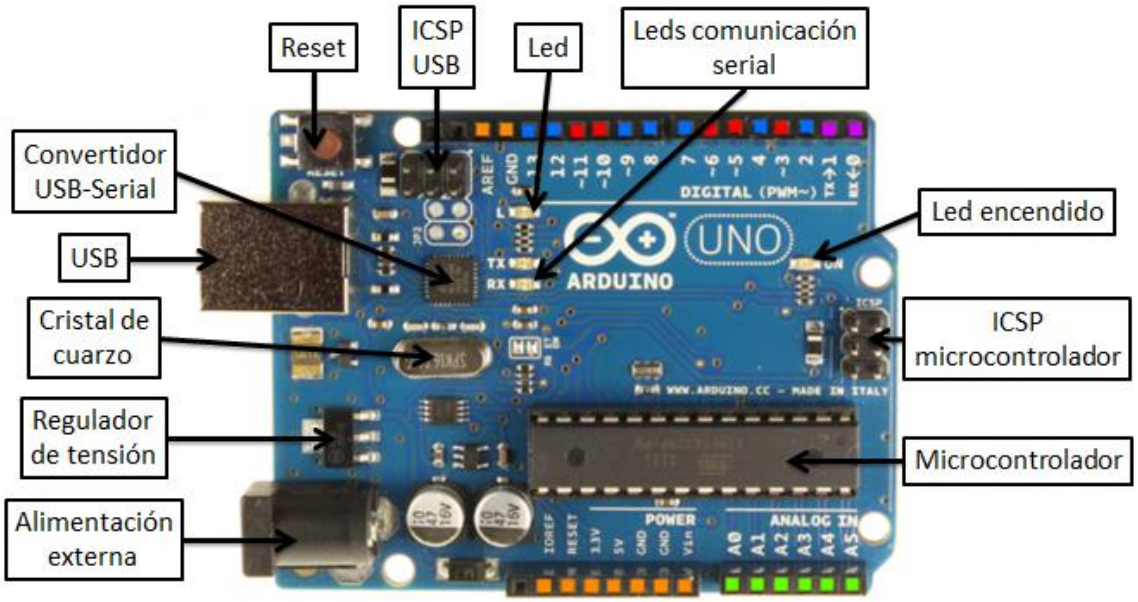


Figura 3-2: Esquema del modelo Arduino UNO rev3

- **Pines**:
  - **Terminales de alimentación y de referencia** (naranja): a través de estos pines se puede alimentar a la placa al mismo tiempo que sirven como referencia de tensión para los circuitos.
  - **Terminales digitales** (azul, rojo, morado y verdes): estos pines tomarán los valores de 0's y 1's. Se pueden configurar como pines de entrada o de salida. Las entradas analógicas también se pueden configurar como digitales.
  - **Terminales PWM** (azul): mediante estos pines se pueden generar señales PWM. El ciclo de trabajo de la señal se puede ajustar con una resolución de 1 byte (desde 0 hasta 255).

  - **Terminales puerto serie** (morado): estos pines permiten enviar recibir datos de otros dispositivos mediante el puerto serie.
  - **Terminales analógicos** (verde): estos terminales cuentan con conversores A/D de 10 bits (desde 0 hasta 1023).

- **Microcontrolador**: las placas Arduino emplean generalmente los microcontroladores ATmega328 y ATmega2560. Son chips sencillos y de bajo coste que permiten el desarrollo de múltiples diseños
- **Terminal ICSP microcontrolador**: permite programar el bootloader del microcontrolador ATmega y poder cargar los programas directamente en el microcontrolador sin tener que necesitar programadores externos.

Figura 3-3: Programación de un Arduino UNO a través de otro dispositivo

El *bootloader* es un conjunto mínimo de instrucciones que permanece almacenado en la memoria Flash del microcontrolador. Le permite interactuar con la *interface* de Arduino, interpretar los programas que se le cargan, recibir y enviar datos por los diferentes puertos o generar señales de control y permite la comunicación USB.

- **LED encendido**: LED que indica si la placa tiene alimentación suficiente como para funcionar.
- **LEDs comunicación serial**: estos LEDs se encienden cuando hay una comunicación por el puerto serie de la placa. Si recibe un dato se encenderá el LED RX (*receive*) y si transmite un dato se encenderá el LED TX (*transmit*).
- **LED**: este LED está unido mediante una resistencia interna (resistencia *pull-up*) al terminal 13. Permite comprobar el correcto funcionamiento de la salida digital

13 sin necesidad de conectar ningún elemento externo a esta para limitar la corriente proporcionada por esta salida.

- **ICSP USB**: permiten emplear la placa de Arduino como un programador de otros microcontroladores.
- **Reset**: sirve para resetear el microcontrolador.
- **Convertidor Serie – USB**: este dispositivo permite la conversión de los datos que llegan por el USB a datos entendibles por el microcontrolador, es decir, transforma los datos a serie. Permite la programación directa del Arduino desde el ordenador.
- **Terminal USB**: permite tanto alimentar la placa como programarla.
- **Cristal de cuarzo**: dispositivo que permite que los microcontroladores operen a una cierta frecuencia. En Arduino esta frecuencia es de 8 o 16 MHz.
- **Regulador de tensión**: sirve para independientemente de la tensión de alimentación de la placa cada elemento interno de la placa obtenga o bien 3’3 V ó 5 V.
- **Alimentación externa**: permite alimentar la placa desde un dispositivo externo sin emplear un cable USB.

3.1.4. LENGUAJE DE PROGRAMACIÓN

Las placas Arduino se programan mediante un lenguaje propio basado en el lenguaje de alto nivel **Processing**, aunque también es posible emplear otros lenguajes de programación y aplicaciones como C++, Java, Matlab o Python, y luego programarse mediante un compilador **AVR** (**A**lf (Egil Bogen) and **V**egard (Wollans)’s **R**ISC processor), el cual corresponde con la familia de microcontroladores de Atmel que incluyen las placas de Arduino.

Sin embargo, uno de los principales atractivos de Arduino es su interfaz de programación. El equipo de Arduino ha desarrollado una aplicación en el lenguaje Processing (que a su vez está basado en Java), que permite una programación muy sencilla a través de un lenguaje en pseudocódigo. Esto hace que el diseñador pueda desentenderse de aspectos engorrosos de la programación del microcontrolador y concentrarse en otros aspectos del proyecto.

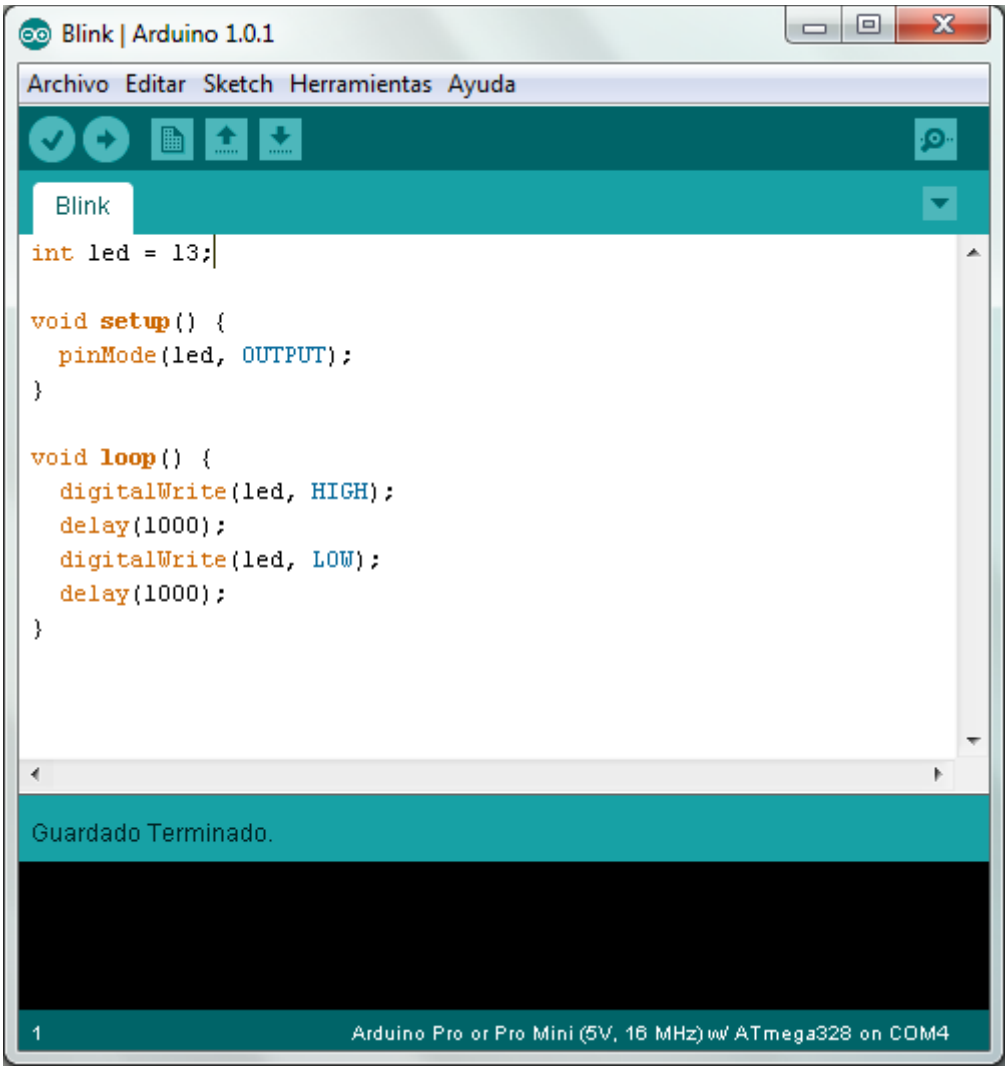


Figura 3-4: Interfaz del IDE de Arduino

La imagen anterior es la pantalla del entorno de desarrollo de Arduino. En la parte de más arriba se encuentran las típicas opciones de todos los programas. En la siguiente fila se tienen elementos propios de Arduino como son verificar, cargar, nuevo, abrir, guardar y monitor serial. Este último sirve para ver qué datos se escriben en el puerto serie.

Después se encuentra el nombre del *sketch* (en este caso Blink) y el *sketch* propiamente dicho. Un *sketch* es como se denomina en Arduino a un programa. En la parte inferior hay una consola que informa de los errores producidos a la hora de compilar el código. Por último se muestra el modelo de placa sobre el que se va a volcar el *sketch* y el puerto en el que está conectada al ordenador.

Todo *sketch* está formado por tres partes. Para explicar estas partes se va a utilizar como ejemplo un sencillo *sketch* que permite encender y apagar el LED que incorpora la placa[3].

- **Declaración de variables**

Lo primero que se debe hacer al empezar a escribir un *sketch* es definir las variables y constantes que formarán el *sketch*.

```
int led = 13;
```

En este caso se ha definido la variable LED como un entero y de valor 13.

- **Configuración de la placa**

Tras declarar las variables y constantes que va a emplear el *sketch* se procede a configurar la placa. Para ello se emplea la sintaxis *void setup().* Las funciones más empleadas aquí son dos. Por un lado está la función *pinMode* que permite definir los terminales como entradas o como salidas. Y por otro lado está la función *Serial.Begin* que establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie.

```
void setup() {
  pinMode(led, OUTPUT);
}
```

En este caso se ha definido el pin 13 (ya que se había declarado la variable LED con el valor de 13) como un pin de salida.

- **Bucle del programa principal**

Una vez configurada la placa se llega al bucle del *sketch*, es decir, lo que va a estar continuamente ejecutándose. Se emplea la sintaxis *void loop().* Aquí las funciones más utilizadas son las funciones de lectura y de escritura de pines: *digitalWrite*, *digitalRead*, *analogRead*, *analogWrite*.

```
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

En este caso en primer lugar se escribe un 1 lógico en el pin 13, después se espera 1 segundo, después se escribe un 0 lógico en el pin 13 y se vuelve a esperar 1 segundo. Este bucle se estaría repitiendo indefinidamente mientras no reseteemos la placa o le quitemos la alimentación.

De esta forma en apenas 10 líneas ha sido posible controlar el LED interno de la placa Arduino. Una vez que se ha comprendido como se enciende y como se apaga un LED es posible poder manipular otros dispositivos como por ejemplo sensores o motores.

3.1.5. MODELOS DE ARDUINO

En la actualidad existen infinidad de modelos Arduino: UNO, Leonardo, Mega, LilyPad, Micro, Pro Mini…

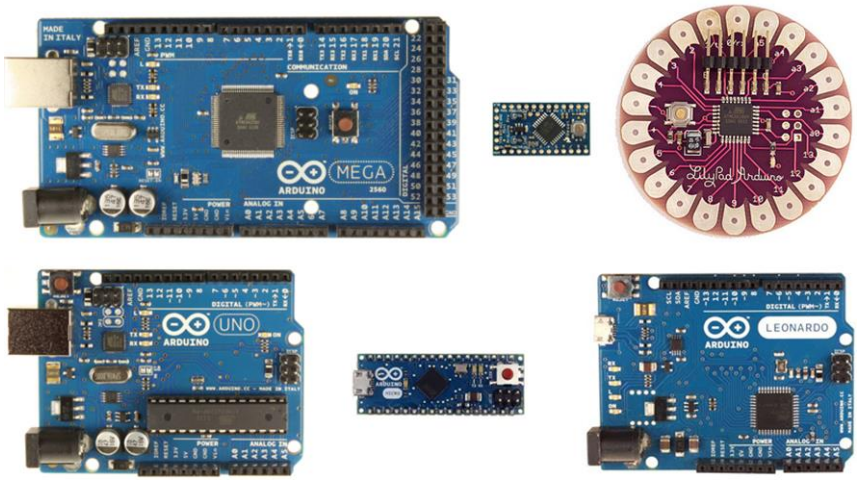


Figura 3-5: Diferentes modelos de Arduino a escala. Fila superior de izquierda a derecha: Mega, Pro Mini y LilyPad. Fila inferior de izquierda a derecha: UNO, Micro y Leonardo

Tras una primera criba se decidió se iba a estudiar a fondo tres modelos de ellos: Mega, UNO y Pro Mini.

- **Arduino Mega**

El Arduino Mega es un microcontrolador basado en el ATmega2560. Dispone de 54 terminales digitales (14 de ellos se pueden emplear como salidas PWM), 16 terminales analógicos y 4 puertos serie.

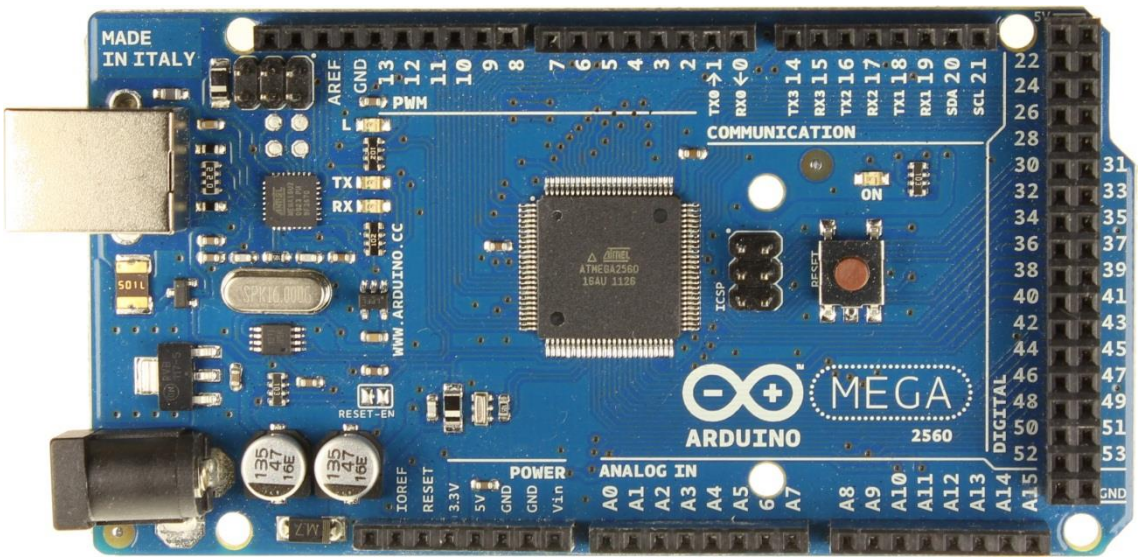


Figura 3-6: Arduino Mega

Posee una memoria interna de tipo flash de 256 KB que permite guardar códigos realmente extensos (8 KB son empleados por el bootloader). Además cuenta con 8 KB de SRAM y 4 KB de EEPROM, al cual se puede acceder desde la librería EEPROM.

Todo esto hace que este modelo sea el más apropiado para los proyectos más complejos en los que se necesiten multitud de entradas y salidas o más memoria. Por otro lado se trata del Arduino de mayor tamaño.

- Arduino UNO

El Arduino UNO es un microcontrolador basado en el ATmega328. Dispone de 14 terminales digitales (6 de ellos se pueden emplear como salidas PWM), 6 terminales analógicos y un puerto serie.

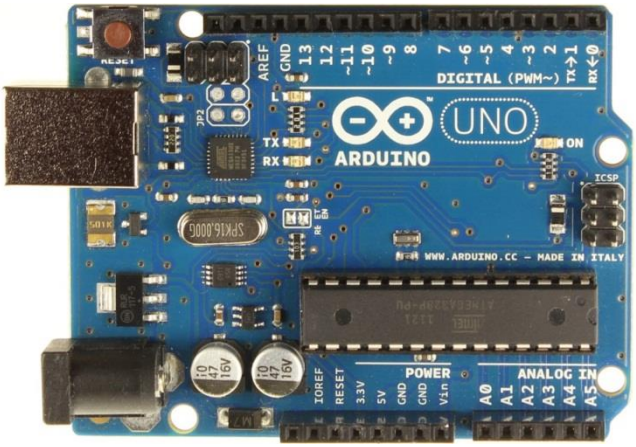


Figura 3-7: Arduino UNO

Posee una memoria interna de 32 KB (0’5 kB son empleados por el bootloader). Además cuenta con 2 KB de SRAM y 1 KB de EEPROM, al que también se puede acceder desde la librería EEPROM.

Este modelo es el más apropiado para proyectos en los que no se necesiten muchas entradas y salidas y el tamaño no sea un problema ya que es el modelo más económico.

- **Arduino Pro Mini**

El Arduino Pro Mini es un microcontrolador que cuenta con las mismas características que el Arduino UNO: mismo microcontrolador, mismo número de entradas y salidas digitales y analógicas y misma memoria.

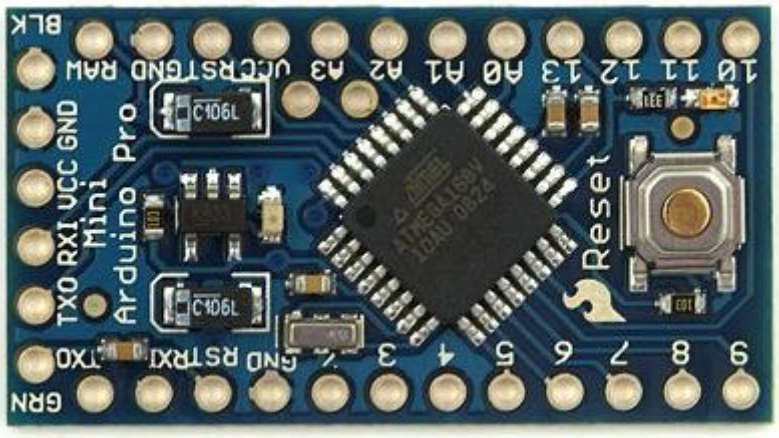


Figura 3-8: Arduino Pro Mini

La gran diferencia del Arduino Pro Mini con respecto al Arduino UNO es su tamaño, ya que en muy poco espacio se dispone de prácticamente todas las opciones que ofrece el Arduino UNO. Esto hace que sea el modelo más apropiado a la hora de realizar proyectos en los que el tamaño sea un factor clave. Es necesario destacar que por razones de coste esta placa no incorpora el circuito integrado de interfaz Serie-USB, con lo cual será necesario disponer de hardware adicional para programarlo.

3.1.6. ELECCIÓN DEL MICROCONTROLADOR

Las características de los tres modelos estudiados se resumen en la siguiente tabla:

Modelo	Arduino Mega	Arduino UNO	Arduino Pro Mini
Micro	ATmega2560	ATmega328	ATmega328
Tensión de funcionamiento	5 V	5 V	5 V
Tensión recomendada	7 - 12 V	7 - 12 V	7 - 12 V
Tensiones límite	6 - 20 V	6 - 20 V	6 - 20 V
Corriente pines entrada/salida	40 mA	40 mA	40 mA
Pines digitales	54 (14 PWM)	16 (6 PWM)	16 (6 PWM)
Pines analógicos	16	6	6
Puertos serie	4	1	1
Memoria interna	256 KB (8 KB bootloader)	32 KB (0'5 KB bootloader)	32 KB (0'5 KB bootloader)
SRAM	8 KB	2 KB	2 KB
EEPROM	4 KB	1 KB	1 KB
Frecuencia	16 MHz	16 MHz	16 MHz
Ventajas	Gran cantidad de entradas y salidas y mucha memoria	Económico	Reducido tamaño

Desventajas	Gran tamaño	-	Hay que soldar los pines y no dispone de conversor USB - SERIE

Tabla 3.1: Comparativa entre diferentes modelos

Tras este análisis se desechó el modelo Arduino Mega. Para este proyecto no se van a emplear multitud de sensores, por lo que se desaprovecharían gran cantidad de entradas y salidas. Por el contrario, se deseó desde un principio que el robot fuera lo más pequeño posible, por lo que los 55 x 110 mm del Arduino Mega hicieron que finalmente se desechara la opción de este modelo.

Figura 3-9: Comparativa a escala del tamaño

Finalmente se decidió hacer el proyecto empleando el Arduino Pro Mini debido a su reducido tamaño. Las desventajas que suponen tener que soldar los pines a la placa y

tener que comprar un dispositivo externo para poder programarlo son compensadas con la diferencia de tamaño que tiene este modelo con respecto a los otros dos.

Como ya se ha explicado, el Arduino Pro Mini carece de conversor Serie – USB, por lo que para poder programarlo se adquirió el conversor Serie - USB FTDI232.

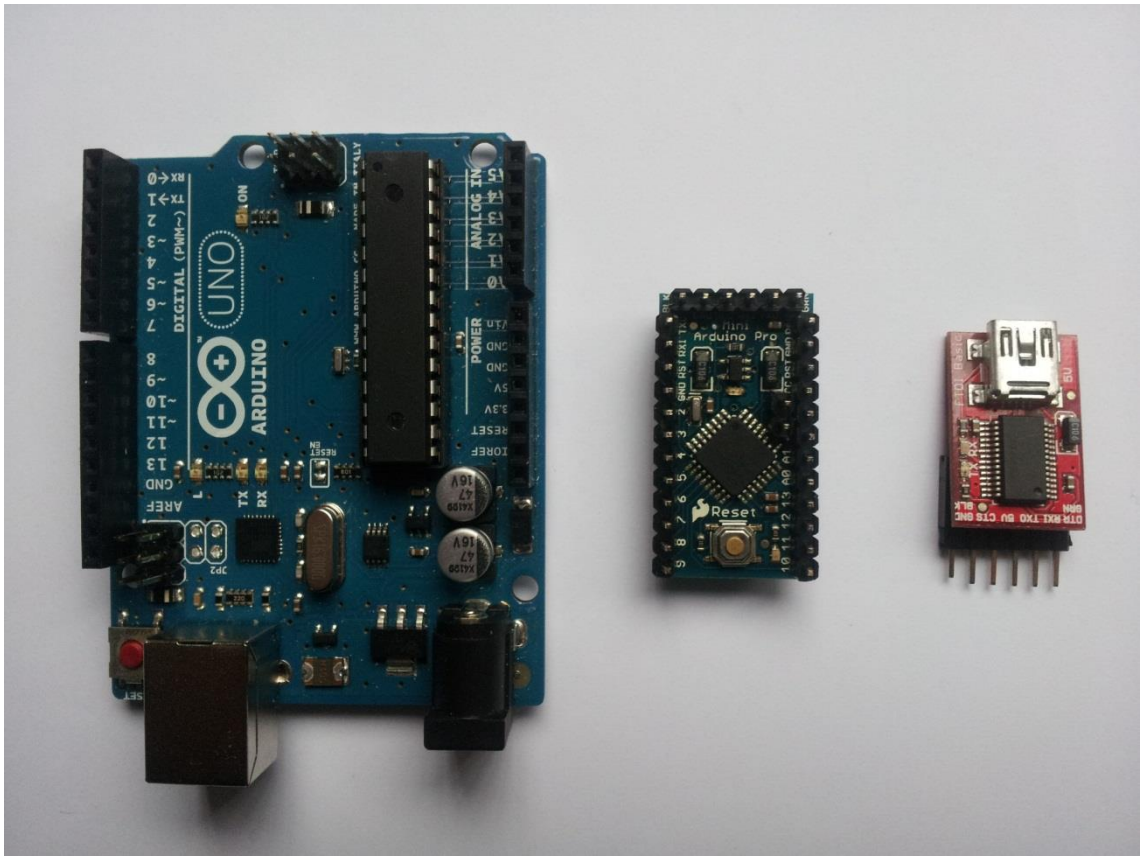


Figura 3-10: De izquierda a derecha: Arduino UNO, Arduino Pro Mini y conversor Serie – USB FTDI232

3.2. RUEDAS

3.2.1. CONFIGURACIÓN DE RUEDAS

A continuación se va a realizar un estudio sobre las diferentes configuraciones posibles que puede presentar el robot en cuanto a disposición de sus ruedas.

- **Configuración Diferencial**

Consta de dos ruedas paralelas entre sí con tracción independiente. En teoría esta es la mecánica más fácil de construir ya que únicamente se necesitan ruedas de tracción, ya que los giros se consiguen con la diferencia de velocidades y sentidos de ambas ruedas.

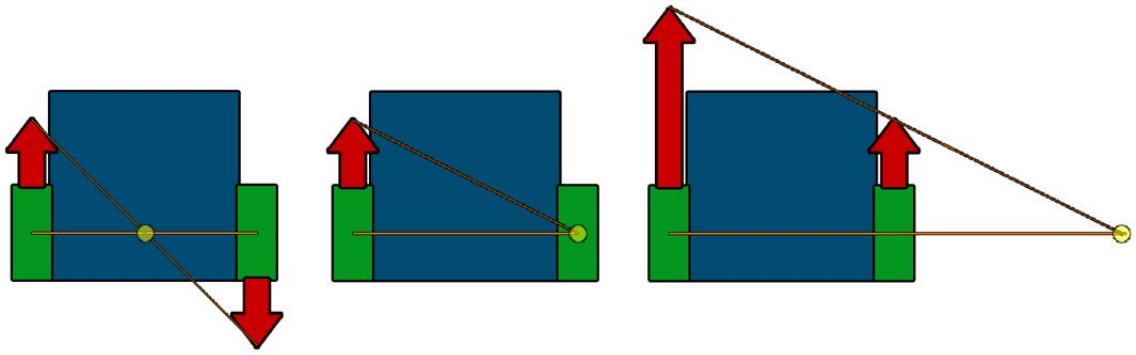


Figura 3-11: Giros con una configuración diferencial

Para darle estabilidad al conjunto se suelen usar una o dos ruedas locas que aguantarán el peso del robot impidiendo que este se incline y entre en contacto con la pista. Sin embargo pueden dar problemas en pistas irregulares ya que se puede perder tracción.

De esta forma su principal ventaja es que es muy fácil de construir. Manipulando las velocidades de ambas ruedas se puede conseguir que el robot se desplace y además gire sin tener que manipular otro elemento. Además permite que el robot pueda girar sobre punto medio del eje formado por ambas ruedas. Esto permite que el robot responda mejor en ruedas cerradas.

Sin embargo esta disposición presenta varios inconvenientes. La principal de ellas es que las ruedas de tracción no pueden ir a la máxima velocidad siempre. Por ejemplo en las curvas la rueda interior deberá frenar o incluso invertir el sentido de giro para que el robot pueda girar. Además, para asegurar el movimiento rectilíneo del robot se deberá comprobar que ambas ruedas giran a la misma velocidad.

- **Configuración en Triciclo**

Esta configuración está compuesta de tres ruedas que forman un triángulo. La delantera proporciona la dirección y las dos traseras, paralelas entre sí, son las de tracción.

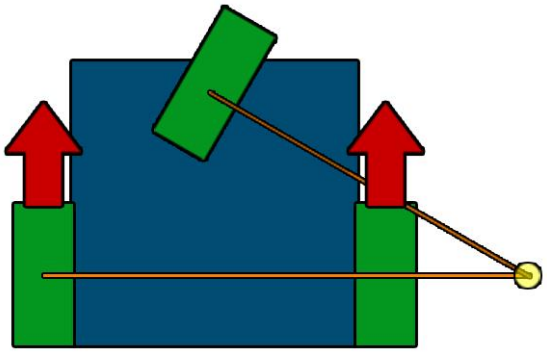


Figura 3-12: Giros con una configuración triciclo

Su principal ventaja es que las ruedas traseras pueden ir siempre a la máxima velocidad siempre que el radio de la curva sea lo suficientemente grande. Además, debido a que ambas ruedas no giran d forma independiente, con esta configuración es más fácil que el robot vaya recto.

El principal problema del triciclo son los giros. El radio mínimo que pueda girar el robot sin problemas dependerá de la distancia que exista entre las ruedas de tracción y la rueda de dirección.

- **Configuración Ackerman**

Es la configuración que tienen los coches, es decir, cuatro ruedas siendo dos de ellas de dirección y dos de tracción. Pueden existir diferentes configuraciones según donde se encuentren las ruedas de tracción.

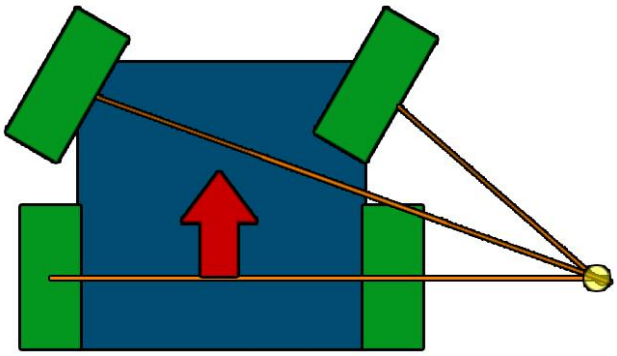


Figura 3-13: Giros con una configuración Ackerman

En un coche las ruedas de dirección generalmente son las delanteras ya que son las ruedas más cercanas al conductor. Sin embargo, en un robot estas pueden sin ningún tipo de problema las traseras. Las ruedas de tracción también suelen ser las delanteras. Sin embargo en robots las ruedas de tracción suelen ser distintas a las de dirección para simplificar el diseño.

La ventaja de la tracción delantera es que se aprovecha mucho mejor la energía en curva porque la fuerza se transmite en la dirección de esta por lo que serán más fáciles de controlar. Sin embargo, si la potencia es elevada se hace difícil de manejar el vehículo de esta manera ya que tienden a subvirar.

Las principales ventajas que ofrece la configuración Ackerman son que las ruedas de tracción pueden ir a máxima velocidad siempre que el radio de giro sea lo suficientemente grande y que ofrece una gran estabilidad.

Por otro lado, el radio de giro en esta configuración no es muy pequeño, por lo que se ha de frenar antes de entrar en una curva.

3.2.2. MODELOS DE RUEDAS

Al mismo tiempo que se analizaban las diferentes configuraciones que podía tomar el robot en cuanto al número y disposición de las ruedas se compararon las diferentes ruedas que estaban disponibles en el mercado:

Imagen	Modelo	Diámetro	Precio
	Pololu Wheel 32x7mm Pair - White	32 mm	5'95 €
	Pololu Wheel 42x19mm Pair	42 mm	5'90 €
	Pololu Wheel 60x8mm Pair - Black	60 mm	6'90 €

Tabla 3.2: Ruedas motrices

Imagen	Modelo	Altura máx	Precio
	Pololu Ball Caster 3/8" Metal Ball	1 cm	3'15 €
	TAMIYA BALL CASTER KIT (2 UND.)	3 cm	5'10 €

Tabla 3.3: Ruedas locas

3.2.3. ELECCIÓN

Tras este análisis se eligió el sistema diferencial. Es el sistema más sencillo de implementar además de que al estar formado por únicamente dos ruedas es el que permite que el diseño del robot sea lo más pequeño posible.

Ya que se seleccionó el sistema diferencial se necesitaba una rueda loca que diera estabilidad al sistema. De las estudiadas se seleccionó la TAMIYA que además de ofrecer una altura máxima mayor, esta se puede ajustar con diferentes añadidos que trae el kit.

Figura 3-14: Tamiya Ball Caster Kit

En cuanto a las ruedas de tracción se escogieron las Pololu de 32 mm ya que eran las ruedas de menor tamaño y se adaptaban mejor al objetivo de que el robot fuera lo más pequeño posible.

Figura 3-15: Rueda de goma 32x7mm (2 und)

3.3. MOTORES

Un motor eléctrico es una máquina que transforma la energía eléctrica en movimiento, y generalmente en movimiento rotativo. Hay muchos tipos diferentes, pero los más empleados en robótica son los motores paso a paso y los motores de corriente continua pudiendo estos últimos encontrarse en su variante con escobillas (la tradicional) o sin escobillas (*brushless*).

Los parámetros principales que describen las características de un motor son la velocidad, medida en rpm, y el par, medido en mN·m o g·cm. La unidad del par muestra la dependencia que hay entre la fuerza y la distancia a la que se aplica. Por ejemplo, un motor con 2 mN·m de par puede producir una fuerza de 2 mN con una palanca de 1 metro conectada a su eje, 1 mN con una palanca de 2 metros, 0'5 mN con una palanca de 4 metros, etc.

Todo motor tiene una velocidad máxima (cuando no se aplica fuerza sobre su eje) y un par máximo (cuando se aplica una fuerza en su eje y esta para el motor). Estos parámetros se denominan *free-running speed* y *stall torque (Ts).*

El motor consume menos corriente cuando no se aplica fuerza sobre su eje y la corriente aumenta a medida que la fuerza aplicada sobre su eje aumenta, hasta llegar al punto de máximo par (Ts)[2]. Esta relación es directa, pero puede no ser lineal dependiendo del tipo de motor que se emplee.

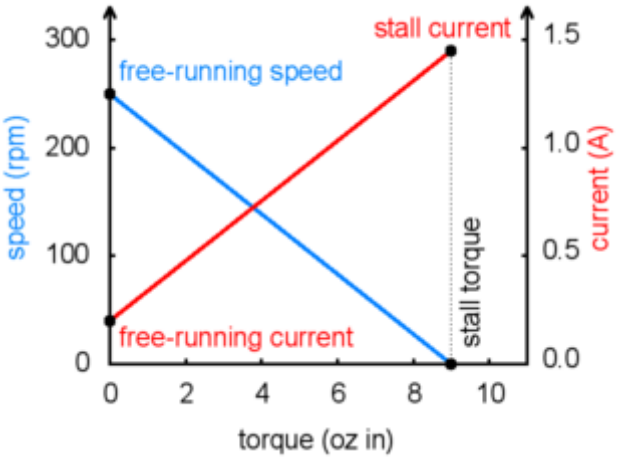


Figura 3-16: Relación velocidad-torque-corriente

En los siguientes apartados se describirán los diferentes tipos de motores y se determinará la velocidad que deben dar estos motores.

3.3.1. MOTORES PASO A PASO

Un motor paso a paso (o *stepper* del inglés) es un dispositivo que produce una rotación en ángulos iguales, denominados pasos, por cada impulso digital que llega a su

entrada. Por ejemplo, si en el caso de un motor un pulso produce un giro de 3°, entonces 120 pulsos producirán una rotación de 360°. Además poseen la peculiaridad de que se pueden enclavar en una cierta posición. Estos motores son los más apropiados para aplicaciones en las que la precisión en la posición angular del eje sea un requisito esencial de diseño[22].

Existen diversos tipos de motores paso a paso, entre los que destacan los de reluctancia variable, los de imán permanente y el híbrido.

Básicamente un motor paso a paso está formado por una parte móvil, el rotor, y una parte fija, el estator. El rotor suele ser un imán permanente o un inducido ferromagnético. En el estator se ubican las bobinas por las que se hará circular la corriente que inducirán un campo magnético. La corriente se va alternando por las diferentes bobinas, lo que provoca que se cree un campo magnético giratorio, el cual seguirá el rotor, produciéndose de este modo el giro del rotor.

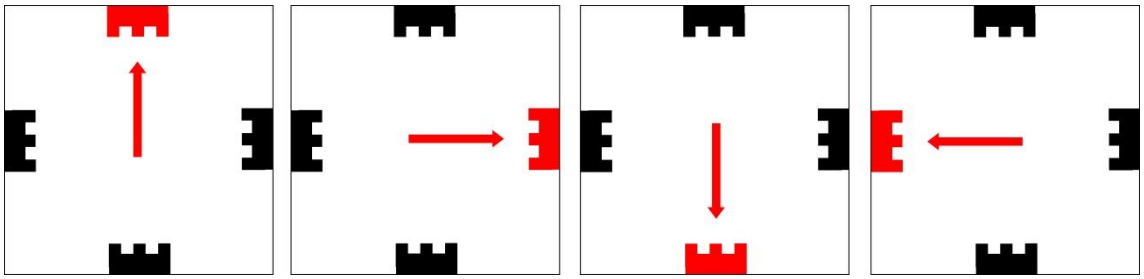


Figura 3-17: Funcionamiento motor paso a paso

De esta forma, si se consigue que la corriente circule por las bobinas en el orden y con la frecuencia adecuada se podrá conseguir que el motor avance un paso en uno u otro sentido.

La principal ventaja que presenta este tipo de motores se la precisión en el giro, por lo que le movimiento del robot será más preciso. Cuanto menor sea el paso del motor mayor será la precisión. Además un motor paso a paso responde a pulsos de entrada digitales, lo que permite un control en lazo abierto, haciendo un control más simple.

3.3.2. MOTORES DE CORRIENTE CONTINUA

Un motor de corriente continua proporciona un par de giro proporcional a la corriente inyectada en su armadura Por este motivo su control de par requiere de un lazo cerrado de control de corriente, sin embargo, para aplicaciones sencillas donde la carga del motor permanece casi constante, se puede asumir que esta corriente controlará la velocidad de giro del motor. Tienen velocidades de giro muy altas, por lo que es necesario utilizar cajas reductoras para adaptar la velocidad a un rango utilizable.

Un motor de este tipo está formado por un estator, el cual da soporte mecánico al sistema y posee una cavidad en el centro de este, y un rotor al que le llega la corriente procedente del estator a través de escobillas.

El principio de funcionamiento de este tipo de motores está basado en la Ley de Lorentz, por la que al introducir un conductor por el que circula una corriente eléctrica en un campo magnético este sufrirá una fuerza perpendicular al plano formado por el campo magnético y la corriente.

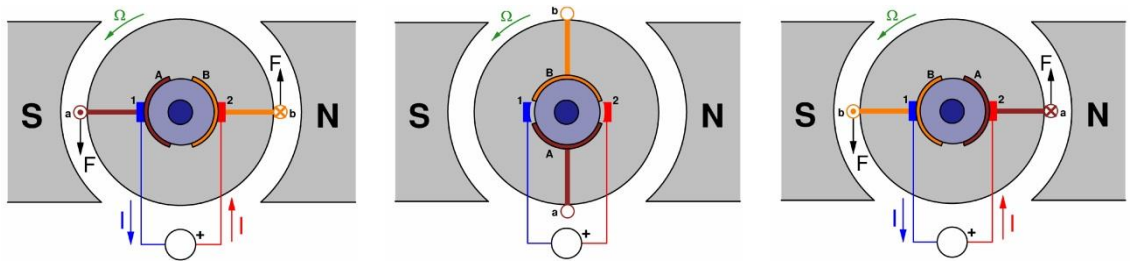


Figura 3-18: Funcionamiento motor de corriente continua

$$F = B \cdot l \cdot I$$

Donde F es la fuerza en newtons, B es la densidad de flujo en teslas, l es la longitud del conductor en metros e I es la corriente que circula por el conductor en amperios.

El rotor tiene varios devanados repartidos por la periferia. A medida que gira, la corriente se activa en el conductor apropiado. De esta forma, el sentido de giro de un motor de corriente continua dependerá del sentido relativo de las corrientes circulantes por los devanados inductor e inducido. Si se desea invertir el sentido de giro del motor se deberá invertir el sentido del campo magnético o de la corriente del inducido.

La ventaja principal de este tipo de motores reside en su fácil adaptación a las diferentes situaciones que puede presentar el sistema, permitiendo ajustar o variar la velocidad, el par y el sentido de rotación.

El problema principal es el mantenimiento de las escobillas. Debido a que es una superficie que está en continuo rozamiento con otra superficie estas se desgastan a largo plazo y es necesario realizar operaciones de mantenimiento periódicas. Esto es de especial importancia en aplicaciones industriales donde el ambiente puede ser especialmente agresivo. El contacto entre estas dos superficies es clave para el correcto funcionamiento del motor, por lo que habrá que asegurarse que no existe ningún problema. En aplicaciones de robótica pequeñas o de juguetería, no suele haber ningún problema con estos componentes durante el periodo de vida del motor.

3.3.3. CÁLCULO DE LA VELOCIDAD

Como objetivo se va a fijar que el motor se mueva aproximadamente a 1 m/s. Para determinar las rpm que debe tener el motor se aplica la siguiente fórmula:

$$V = 2\pi RN/60$$

Tal que V viene en m/s, R en m y N en rpm. Teniendo en cuenta que el radio de las ruedas escogidas es de 16 mm se tiene:

$$1 = \frac{2\pi \cdot 0{,}016 \cdot N}{60} \rightarrow N = 596{,}83\ rpm$$

3.3.4. ELECCIÓN MOTOR

Tras analizar cada uno de los motores se optó por emplear motores de corriente continua. Este tipo de motor se adapta mejor a las necesidades del robot, ya que se necesita que el motor sea capaz de cambiar su velocidad cuanto más rápido mejor, y en ese aspecto los motores de corriente continua son más apropiados que los de paso a paso. Además el control de la velocidad de los motores de corriente continua ofrece más posibilidades que el control de motores paso a paso.

Los motores que más se aproximaban a la velocidad necesaria eran dos:

Imagen	Modelo	Reductora	Torque	Velocidad sin carga	Consumo sin carga	Precio
	10:1 Micro Metal Gearmotor	10:1	0,2 kg·cm	1250 rpm	40 mA (Max: 360 mA)	13'20 €
	30:1 Micro Metal Gearmotor	30:1	0,3 kg·cm	440 rpm	40 mA (Max: 360 mA)	13'20 €

Tabla 3.4: Motores

Según las ruedas escogidas de 16 mm de radio los motores ofrecerían una velocidad de 2 m/s el modelo con reductora de 10:1 y de 0’74 m/s el modelo con reductora de 30:1. Debido a que el robot no va a ser un velocista sino un rastreador, por lo que al fin y al cabo la velocidad no es el punto más importante en el diseño del robot, se escogió el modelo con una reductora de 30:1 ya que ofrecía un par mejor.

Figura 3-19: Motor Micro Metal DC con reductora 30:1

Por último, para poder fijar los motores se adquirieron un par de fundas protectoras para este tipo de motores.

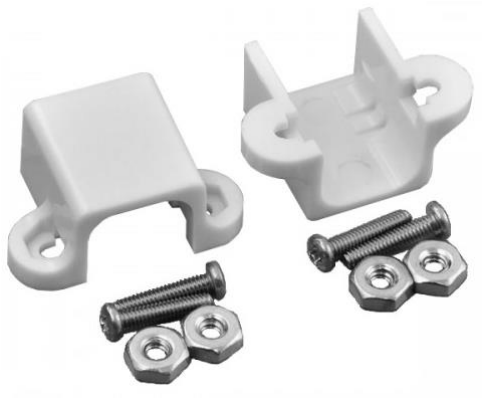


Figura 3-20: Funda protectora para motor micro metal (2 und)

3.4. ACTUADOR ELECTRÓNICO

Como se ha especificado en el apartado anterior, los motores seleccionados necesitan de al menos 40 mA para funcionar. Si se recuerdan las especificaciones del Arduino Pro Mini cualquier línea de salida de este puede proporcionar hasta un máximo de 40 mA, por lo que se necesita de otro componente que pueda alimentar a los motores.

Además, si se desea que la velocidad de giro de los motores sea variable y además que se pueda producir en ambos sentidos se hace más evidente la necesidad de otro componente.

De entre las diferentes tipologías de actuadores electrónicos de potencia, en este proyecto se empleará la conocido como etapa conmutada en puente en H. Este componente se denomina puente en H. Es una de las partes críticas del robot, ya que un mal funcionamiento o un mal montaje de este provocarán que el robot no funcione como se espera de él. En los siguientes apartados se va a proceder a explicar los motivos de su elección, su funcionamiento y a analizar los diferentes modelos de circuitos integrados que se estudiaron para su implementación en este proyecto.

3.4.1. ¿QUÉ ES UN PUENTE EN H?

Un puente en H es un circuito electrónico de potencia que está formado por una serie de transistores, los cuales permitirán pasar o cortar la corriente en un determinado

sentido[1]. Como se acaba de comentar, los puntos de trabajo de los transistores serán el de corte (OFF) o saturación (ON), y estos puntos de trabajo serán controlados mediante sus terminales de Base (en el caso de transistores BJT) o de Puerta (en el caso de transistores MOSFET).

Para el caso unipolar, la etapa se compone de cuatro transistores que actúan como interruptores, los cuales pueden estar en dos estados, abierto (ON) y cerrado (OFF). Cuando un interruptor está abierto no permite el paso de corriente a través de él, en cambio cuando esté cerrado sí lo permitirá.

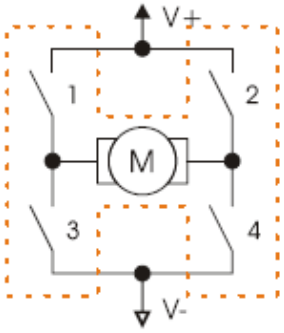


Figura 3-21: Esquema de un puente en H

Variando los puntos de trabajo de los interruptores se puede conseguir que el motor gire en un sentido u otro, o que se quede parado al fijar los dos terminales del motor a una misma tensión.

Si el puente en H tiene los interruptores 1 y 4 cerrados mientras que los interruptores 2 y 3 permanecen abiertos, se permite el paso de la corriente de izquierda a derecha. De esta forma el motor gira en un sentido.

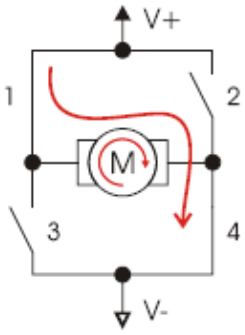


Figura 3-22: Giro en un sentido del motor

Si se invierte el estado de los interruptores la corriente circula en sentido contrario. De esta forma variando entre estas dos posiciones se consigue que el motor gire en uno o en otro sentido.

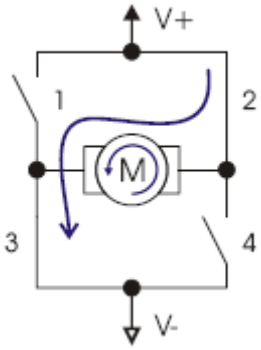


Figura 3-23: Giro en el otro sentido del motor

También hay que tener en cuenta que no todas las combinaciones son correctas, ya que algunas posiciones crean cortocircuitos. Es decir, los interruptores de una misma rama (1-3 y 2-4) no podrán estar cerrados al mismo tiempo.

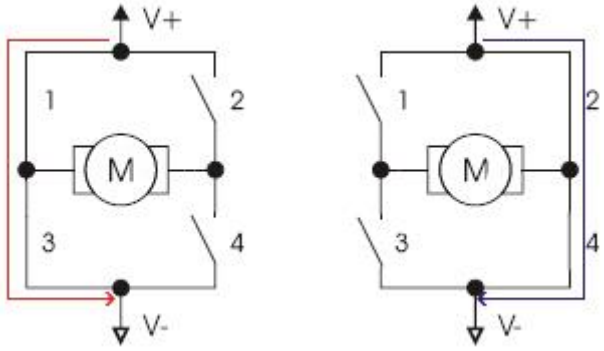


Figura 3-24: Situaciones no posibles

Por último, si los interruptores 1 y 2 estuvieran cerrados y los interruptores 3 y 4 abiertos tendrían ambos terminales del motor el mismo voltaje, con lo que el motor se pararía rápidamente, más incluso que si se cortase la alimentación. A este modo se le llama *Fast stop*.

En la siguiente tabla se resumen todas las combinaciones posibles de los interruptores y el resultado que tienen sobre el motor:

Estado	I1	I2	I3	I4	Resultado
1	Abierto	Abierto	Abierto	Abierto	El motor se detiene bajo su inercia
2	Cerrado	Abierto	Abierto	Cerrado	El motor gira en avance
3	Abierto	Cerrado	Cerrado	Abierto	El motor gira en retroceso
4	Cerrado	Cerrado	Abierto	Abierto	*Fast stop*

Tabla 3.5: Estados de los motores en función de los estados de los interruptores

Una de las principales ventajas del actuador en puente en H es que permite controlar de forma precisa el valor promedio de tensión que se aplica en su carga. Es decir, que no es un actuador Todo/Nada como pudiera parecer debido al funcionamiento en forma de interruptor de sus transistores, sino que es posible obtener valores intermedios controlables de tensión en la carga. Esto permite controlar la corriente que se inyecta en los motores y, en última instancia, la velocidad a la que gira.

Esto se consigue mediante la técnica de Modulación de Ancho de Pulsos o PWM de sus siglas en inglés (Pulse Width Modulation). Esta técnica consiste en un sencillo principio que consiste en alternar el estado de conducción del puente a una frecuencia alta, mucho mayor que la constante de tiempos del motor alimentado. De esta manera, el motor no percibe las bruscas variaciones de tensión, sino que las ve como un valor promedio. Como el voltaje que ve el motor es una serie de pulsos de ancho variable, a este método se denomina Modulación de Anchos de Pulsos.

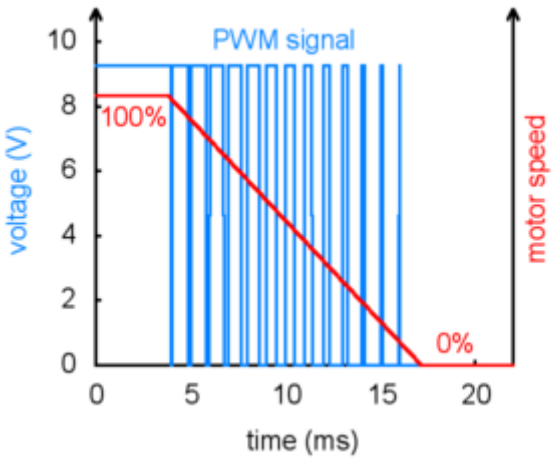


Figura 3-25: Relación entre el voltaje y la velocidad del motor en función del tiempo

El principal concepto importante de entender es el conocido como ciclo de trabajo. Este se define como la fracción del periodo de conmutación en que la señal se encuentra en estado activo:

$$D = \frac{\tau}{T}$$

Donde D es el ciclo de trabajo, τ es la duración en la que la señal se encuentra en estado activo y T el periodo de la señal.

En la gráfica se puede ver un ejemplo de PWM. En un principio el ciclo de trabajo es del 100 %, por lo que el motor gira al 100 % de su velocidad nominal. Sin embargo, cuanto más bajo es el ciclo de trabajo menor es la velocidad a la que gira el motor, hasta llegar al 0 %.

A continuación se van a estudiar tres componentes que se plantearon para usarlos como puentes en H del robot.

3.4.2. ULN2803

El ULN2803 es un circuito integrado que contiene 8 transistores en configuración Darlington con sus respectivos diodos de retroceso.

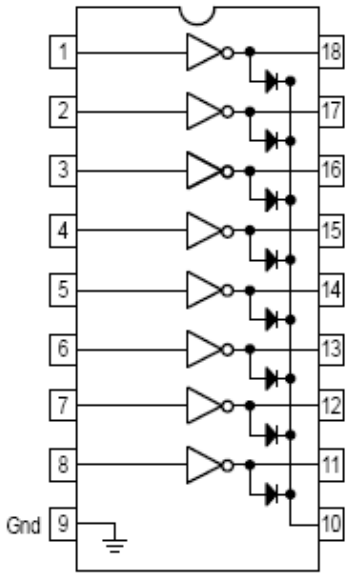


Figura 3-26: Esquema del ULN2803

Se pensó en utilizar este circuito integrado ya que lo recomendaba el fabricante de los motores. Las etapas de tipo Darlington son muy adecuadas ya que suelen ser capaces de manejar valores de corriente bastante elevados.

Cada una de sus salidas puede ofrecer hasta 500 mA, pero además se pueden conectar en paralelo para ofrecer una corriente mayor. Está optimizado para trabajar con voltajes comprendidos entre 6 y 15 V, por lo que en principio podrían ser perfectos para alimentar los motores.

Se pensó en usar cada uno de los inversores de los que está formado el ULN2803 como interruptores para la construcción del puente en H. Sin embargo, al analizar más en profundidad el ULN2803 se comprobó que este solo podría funcionar en medio puente, es decir, solo permitiría girar a los motores en un sentido. Esto es debido a que es un circuito integrado especialmente diseñado para el control de displays y de LEDs, por lo que todos sus canales están internamente conectados con la configuración de cátodo común. Como ya se especificó en el 3.2, al disponer de una configuración diferencial se necesita que ambas ruedas puedan girar en ambos sentidos y por ello se deben proporcionar corrientes en los dos sentidos de la carga, por lo que en principio este integrado no nos es útil.

Se pensó una solución para poder usar este integrado como un puente completo. Esta consistió en añadir un par de transistores PNP, los cuales permitirían construir un puente completo con el ULN2803[23].

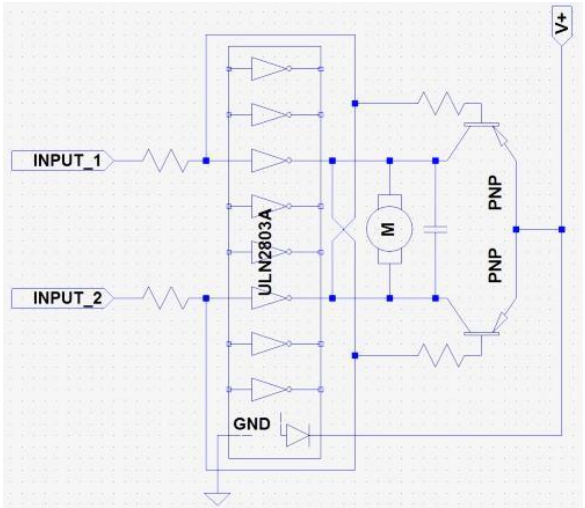


Figura 3-27: Posible solución para el ULN2803

Finalmente se desechó esta opción debido a que existen opciones comerciales en el que el circuito integrado ya integra todo el sistema de control dentro de él.

3.4.3. L298N

El L298N es un circuito integrado formado por dos puentes en H separados. Cada uno de estos puentes permite obtener hasta 2 A (o 4 A si se conectan en paralelo ambos puentes), por lo que se pueden utilizar para alimentar a los motores.

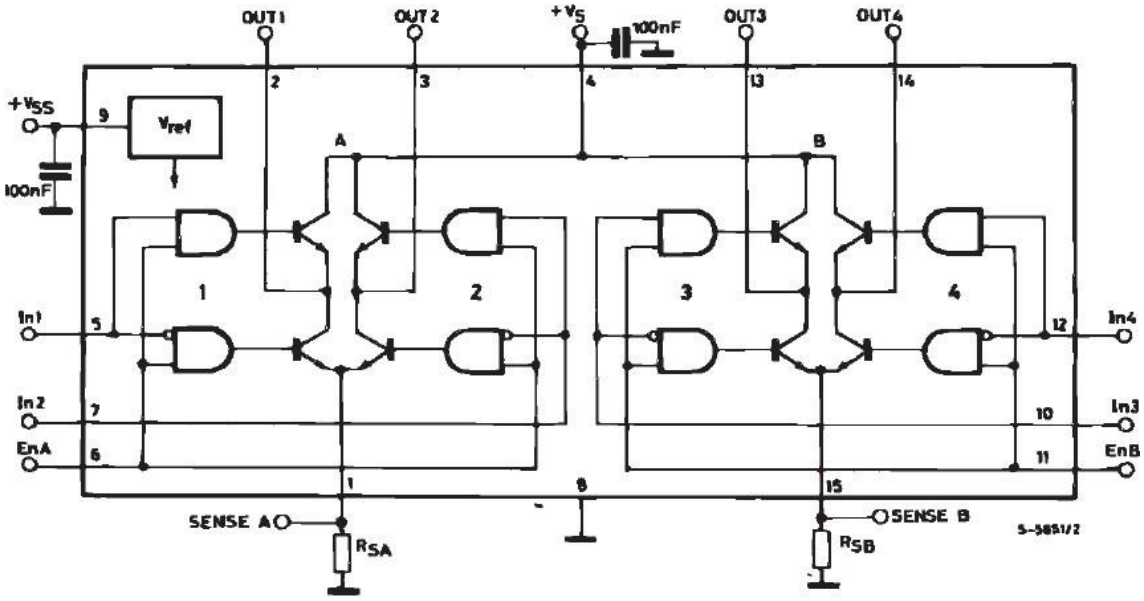


Figura 3-28: Esquema del L298N

Se empleó este circuito integrado ya que se disponía de él en el laboratorio y cumplía los requerimientos de los motores.

Sin embargo, tras diversas pruebas con él, se desechó esta opción por tres motivos. El primero de ellos es su tamaño. De nuevo el requerimiento de construir un robot lo más pequeño posible influyó en la elección de un componente. Este circuito integrado tiene unas medidas exageradas en comparación a otros circuitos integrados que ofrecen las mismas posibilidades.

Figura 3-29: L298N

La segunda razón es que se presentó un excesivo calentamiento. Aun haciéndole pasar apenas un poco más de 100mA el disipador que incorpora aumentaba mucho su temperatura, tanto que había que esperar varios minutos para volver a manipularlo una vez terminada una prueba Esto podría solucionarse mediante el uso de un radiador, pero aumentaría aún más el tamaño del actuador. Se pensó también en utilizar el plano de masa de la tarjeta de circuito integrado como radiador.

La tercera razón, relacionada con la anterior, es que el L298N no incluye ningún elemento de protección contra sobretensiones inversas (problema muy frecuente cuando se alimentan cargas de componente inductiva), por lo que había que añadirle un total de ocho diodos (4 para cada motor) para su perfecto funcionamiento y evitar que algún pico de tensión inversa pudiera dañar algún transistor. Estos diodos suelen conocerse como diodos de libre circulación. Esto hacía de nuevo que el tamaño ocupado por este circuito integrado fuera demasiado grande en comparación con otras opciones.

3.4.4. L6205N

El L6205N es un circuito integrado formado por 2 puentes en H completos. En el mismo empaquetado se incluyen también los diodos de libre circulación[24].

Sus principales características son que funciona con tensiones de alimentación desde los 8 V hasta los 52 V y puede ofrecer corrientes de salida de hasta 2’8 A. Por lo

que puede ser utilizado para alimentar los motores. Este integrado emplea transistores de tipo MOSFET en lugar de los BJT del L298N

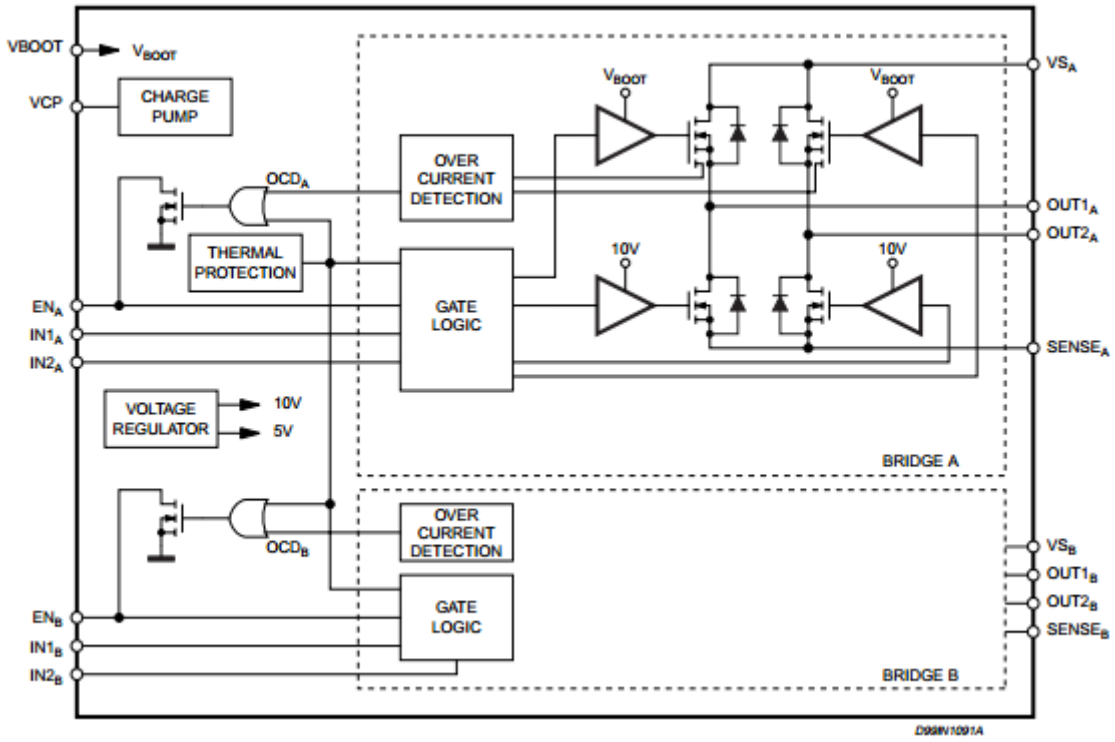


Figura 3-30: Esquema del L6205N

A continuación se va a estudiar cada conexión del circuito integrado:

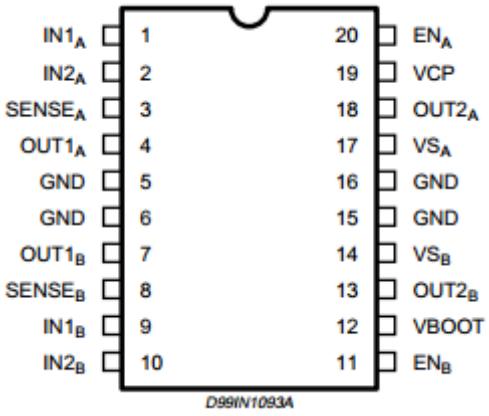


Figura 3-31: Relación de pines del L6205N

1. IN1A: Entrada lógica 1 del Puente A.

2. IN2A: Entrada lógica 2 del Puente A.

3. SENSEA: Pin de medida del Puente A. Se deberá conectar a tierra si no se desea usar este pin.

4. OUT1A: Salida 1 del Puente A.

5. GND: Tierra.

6. GND: Tierra.

7. OUT1B: Salida 1 del Puente B.

8. SENSEB: Pin de medida del Puente B. Se deberá conectar a tierra si no se desea usar este pin.

9. IN1B: Entrada lógica 1 del Puente B.

10. IN2B: Entrada lógica 2 del Puente B.

11. ENB: Enable del Puente B. Un nivel bajo en este pin apagará todos los transistores que forman el Puente B. Si no se va a emplear se deberá conectar a +5 V a través de una resistencia.

12. VBOOT: Tensión para que empiecen a funcionar los transistores de ambos puentes.

13. OUT2B: Salida 2 del Puente B.

14. VSB: Alimentación del Puente B. Debe estar conectada a la fuente de alimentación junto a VSA.

15. GND: Tierra.

16. GND: Tierra.

17. VSA: Alimentación del Puente A. Debe estar conectada a la fuente de alimentación junto a VSB.

18. OUT2A: Salida 2 del Puente A.

19. VCP: Voltaje del multiplicador del oscilador.

20. ENA: Enable del Puente A. Un nivel bajo en este pin apagará todos los transistores que forman el Puente A. Si no se va a emplear se deberá conectar a +5 V a través de una resistencia.

3.4.5. PRUEBAS L6205N

Antes de probar el L6205N se estudiaron sus hojas de características. Estas recomendaban emplear el siguiente esquema para hacer funcionar el circuito integrado:

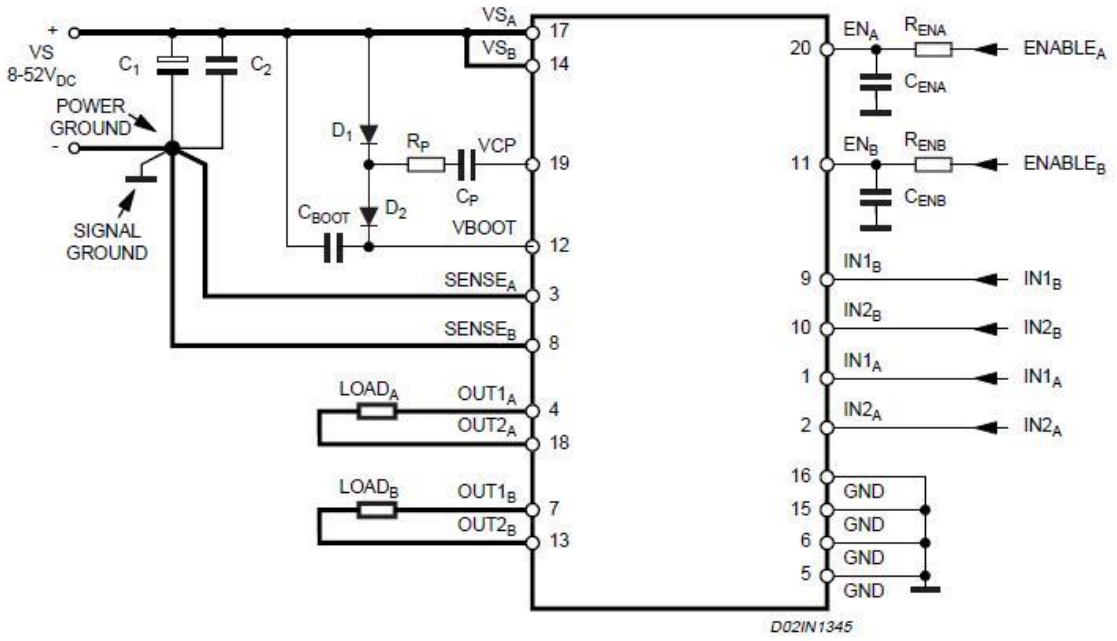


Figura 3-32: Esquema de conexión del L6205N

En este esquema se pueden apreciar diferentes componentes que se van a proceder a explicar a continuación:

- C1 y C2: la función de estos condensadores es doble. Por un lado filtra altas frecuencias en la alimentación del L6205N; es decir, impedir que llegue ruido al circuito integrado y de esta forma asegurar que siempre que se desee el L6205N se encuentre alimentado de forma estable. Por otro lado también reduce los transitorios generados al encender y apagar los transistores.
- RENA, CENA, RENB y CENB: la función de estos componentes es la de crear un filtro de paso de bajo. De esta forma se consigue evitar que los puentes se desactiven por errores debido a interferencias.

- CBOOT, CP, RP, D1 y D2: estos componentes tienen como función la de preparar la onda que requiere el oscilador interno del L6205N. Esta debe de ser una señal cuadrada de 600 kHz de frecuencia y 10 V de amplitud.

Siguiendo las recomendaciones del fabricante y ajustando los valores, se escogieron los siguientes componentes:

Componente	Valor
C_1	120nF
C_2	120nF
C_{BOOT}	220nF
C_P	15nF
C_{ENA}	6'8nF
C_{ENB}	6'8nF
D_1	1N4148
D_2	1N4148
R_{ENA}	100kΩ
R_{ENB}	100kΩ
R_P	120Ω

Tabla 3.6: Componentes empleados

De esta forma para realizar las pruebas se emplearon:

- Un L6205N.
- Un Arduino UNO para generar los PWM y activar los puentes.
- Dos motores.
- Placa de prototipos.
- Osciloscopio.
- Cables y componentes.

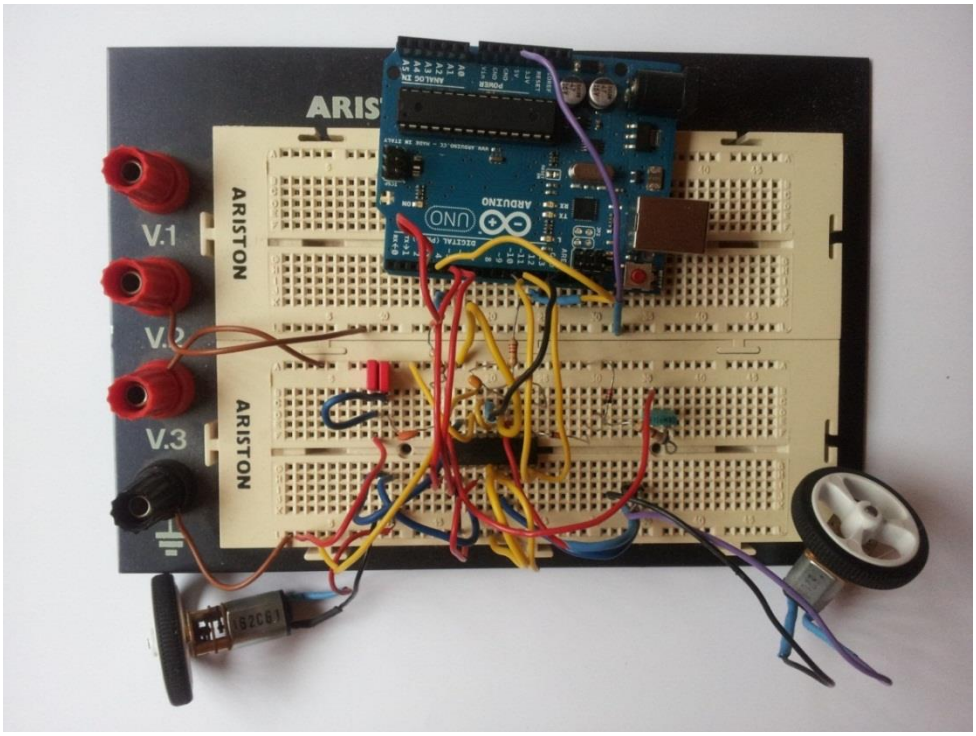


Figura 3-33: Montaje para la prueba del L6205N

A la hora de programar el Arduino se pensaron diferentes técnicas. En primer lugar se pensó en mandar los pulsos PWM a través de los *enables* (uno para cada puente) para que de esta forma solo se empleasen dos pines PWM de la placa de Arduino. Sin embargo, al poner en LOW el puente no se conseguían 0 V a la salida, sino 3 V. Esto también repercutía en el PWM que veían las salidas. Se llegaba al máximo con 165 (sobre 255 es un 64%, por lo que la precisión dejaba mucho que desear). De esta forma se optó por controlar de forma individual cada una de las entradas. De esta forma se necesitarían cuatro pines PWM de la placa Arduino.

TRUTH TABLE

INPUTS			OUTPUTS	
EN	**IN1**	**IN2**	**OUT1**	**OUT2**
L	X	X	High Z	High Z
H	L	L	GND	GND
H	H	L	Vs	GND
H	L	H	GND	Vs
H	H	H	Vs	Vs

X = Don't care

High Z = High Impedance Output

Tabla 3.7: Tabla de la verdad del L6205N

En cuanto a la programación, se pensó que sería mejor dejar ambos *enables* siempre en H, de esta forma desde Arduino se mandarían las señales PWM a cada *input*. Así, si tras los cálculos el valor es positivo se manda el PWM al IN1 y se manda un 0 al IN2. SI el valor fuera negativo se manda el PWM al IN2 y se manda un 0 al IN1.

Finalmente se comprobó el perfecto funcionamiento de ambos puentes:

Figura 3-34: Resultado obtenido en la prueba

El programa que se ejecutó en Arduino fue el siguiente (ANEXO A.1):

```
/*************************************************************
 *               Prueba funcionamiento L6205N                *
 *************************************************************/
int enA = 3;                // enable del Puente A
int enB = 4;                // enable del Puente B
int in1A = 5;               // entrada 1 del Puente A
int in2A = 6;               // entrada 2 del Puente A
int in1B = 10;              // entrada 1 del Puente B
int in2B = 11;              // entrada 2 del Puente B
int pwmA = 50;              // valor PWM del Puente A
int pwmB = 150;             // valor PWM del Puente B
void setup()  {
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1A, OUTPUT);
  pinMode(in2A, OUTPUT);
  pinMode(in1B, OUTPUT);
  pinMode(in2B, OUTPUT);
}
void loop()  {
  digitalWrite(enA, HIGH);
  digitalWrite(enB, HIGH);
  analogWrite(pwmA, in1A);
  analogWrite(0, in2A);
  analogWrite(pwmB, in1B);
  analogWrite(0, in2B);
}
```

3.5. SENSORES INFRARROJOS

Los sensores infrarrojos van a ser el único medio a través del cual el robot reciba información de su entorno cuando se encuentre funcionando en modo rastreador. Primero se tiene que determinar qué información necesita obtener el robot sobre su entorno o sobre sí mismo.

Lo único que necesita conocer el robot sobre su entorno es la situación de la línea negra que tiene que seguir. Para ello se emplean los sensores infrarrojos. Estos pueden leer el nivel de oscuridad del suelo en puntos concretos, por lo que es importante a la hora de diseñar el robot elegir bien el número y la distribución de estos.

Lo más común es situar estos sensores en la parte delantera del robot para que de esta manera se pueda anticipar antes la llegada de una curva. Ahora bien, existen diferentes configuraciones posibles. La primera de ellas consiste en adquirir una matriz de sensores infrarrojos. Esta matriz está compuesta por una serie de sensores infrarrojos dentro de un mismo circuito integrado. El número de estos sensores puede variar. La principal ventaja que tienen es su reducido tamaño mientras que su desventaja está en que no se puede colocar cada sensor donde quieras.

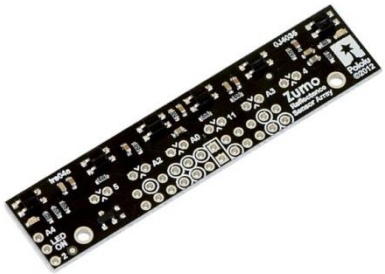


Figura 3-35: Matriz de sensores Pololu Zumo

También es posible adquirir cada sensor infrarrojo por separado y colocar cada uno de ellos donde más le convenga al robot. El sensor por excelencia es el CNY70, del cual se puede encontrar multitud de información por internet. La principal ventaja de esta configuración es que puedes ser más creativo con la distribución de los sensores y colocarlos de la forma más eficiente posible. Sin embargo, estas configuraciones requieren una cantidad de espacio generosa, ya que se necesita montar circuitería

adicional como resistencias y transistores para hacer funcionar correctamente a los sensores.

Figura 3-36: Disposición de los sensores de n00b0t

Otro factor importante a tener en cuenta a la hora de elegir los sensores infrarrojos es su distancia al suelo. Esta viene determinada por el tipo de sensor y por el tipo de lectura que se realiza. En la mayoría de las ocasiones la lectura es digital, es decir, se leen 0's y 1's. En general, cuanto más pegado esté el sensor al suelo mejores lecturas realizará.

Por último es importante determinar la distancia que hay entre los sensores y el eje de los motores. Cuanto más alejados estén los sensores del eje de los motores antes se anticipará la llegada de una curva, pero, sin embargo, los motores deberán de hacer más fuerza para mover o corregir el robot, por lo que en principio responderá peor. De esta forma hay que encontrar un equilibrio entre la distancia a la que se encuentran los motores entre sí y la distancia que existe entre los sensores y el eje de los motores.

Finalmente se eligió como sensor infrarrojo el QTR-8RC Reflectance Sensor Array. Las razones por las que se eligió este sensor son las ya explicadas con respecto a una matriz de sensores infrarrojos.

Figura 3-37: QTR-8RC

3.5.1. QTR-8RC

La matriz de sensores infrarrojos QTR-8RC está diseñada específicamente para la utilidad que se le va a dar en este robot, es decir, para detectar líneas, aunque se puede emplear para otros propósitos como por ejemplo sensor de proximidad.

Está formada por ocho parejas de emisores infrarrojos y de receptores (fototransistores) espaciados entre sí 9'5 mm. Cada fototransistor emplea un circuito de descarga de un condensador, el cual permite que un pin digital del microcontrolador analice la cantidad de luz que le llega al sensor según el tiempo que tarde en descargarse el condensador. Cuanta más rápida sea la descarga del condensador mayor será la reflectividad del objeto al que mira el sensor. Mayor reflectividad implica mayor cantidad de luz en la base del fototransistor, lo que inmediatamente provoca un incremento en la corriente de colector, que descarga más rápidamente el condensador[25].

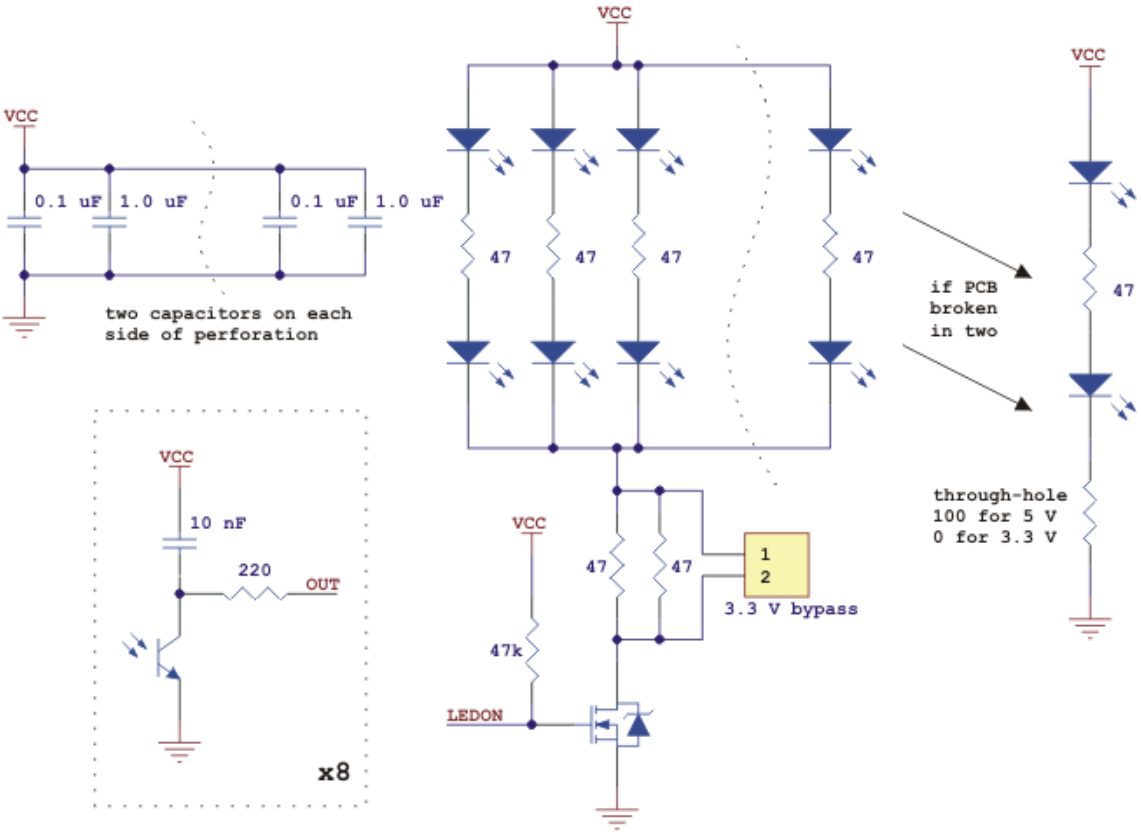


Figura 3-38: Esquema de funcionamiento del QTR-8RC

Todas las salidas son independientes, pero los LEDs están dispuestos en parejas para reducir a la mitad el consumo. Los LEDs se controlan mediante un transistor MOSFET, cuya puerta normalmente está en alto, permitiendo apagar los LEDs al poner en bajo la puerta del MOSFET. Se puede reducir el consumo del sensor apagando los LEDs cuando no se está usando el sensor o ajustando el brillo de los LEDs mediante un PWM.

Las resistencias que componen la matriz hacen que esta funcione a 5V, con un consumo de cada LED de 20 – 25mA, haciendo un consumo total de la matriz de 100mA. Añadir que la matriz se puede dividir en dos submatrices, dejando en una seis sensores y en la otra dos.

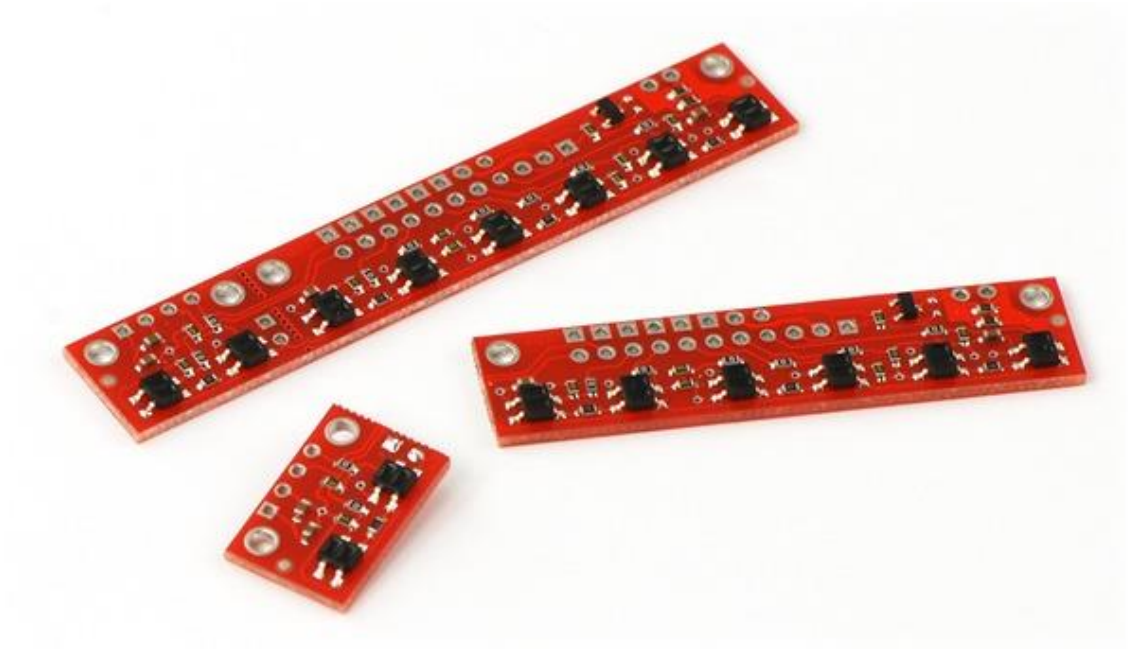


Figura 3-39: División de la matriz QTR-8RC en dos

- Dimensiones: 7’59 cm x 1’27 cm x 0’32 cm
- Voltaje de funcionamiento: 3’3 – 5 V
- Consumo de corriente: 100 mA
- Formato de las salidas: digital
- Distancia óptima: 3 mm
- Máxima distancia recomendad: 9’5 mm
- Peso: 3’09 g

3.5.2. PRUEBAS QTR-8RC

Al igual que el L6205N, la matriz de sensores es una de las piezas clave del robot, por lo que se le realizaron una serie de pruebas antes de su montaje en el robot. En primer lugar hay que entender cómo funciona el QTR-8RC antes de probarlo. La secuencia que debe seguir cada sensor es la siguiente:

1. Encender el LED infrarrojo.

2. Configurar la vía de comunicación del sensor como salida.

3. Dejar tiempo suficiente para que el condensador se cargue. Según el fabricante, con 10 microsegundos es suficiente.

4. Configurar la vía de comunicación del sensor como entrada.

5. Medir el tiempo que tarda el condensador en descargarse.

6. Apagar el LED infrarrojo.

Cuando haya una alta reflectividad, como por ejemplo cuando el sensor esté sobre una superficie blanca, el tiempo de descarga será de unos pocos microsegundos; con una baja reflectividad, como por ejemplo cuando el sensor esté sobre una superficie negra, el tiempo de descarga será de unos pocos milisegundos.

Tras haber entendido el funcionamiento del sensor se llevó a cabo su montaje y se comprobó que los sensores funcionaban:

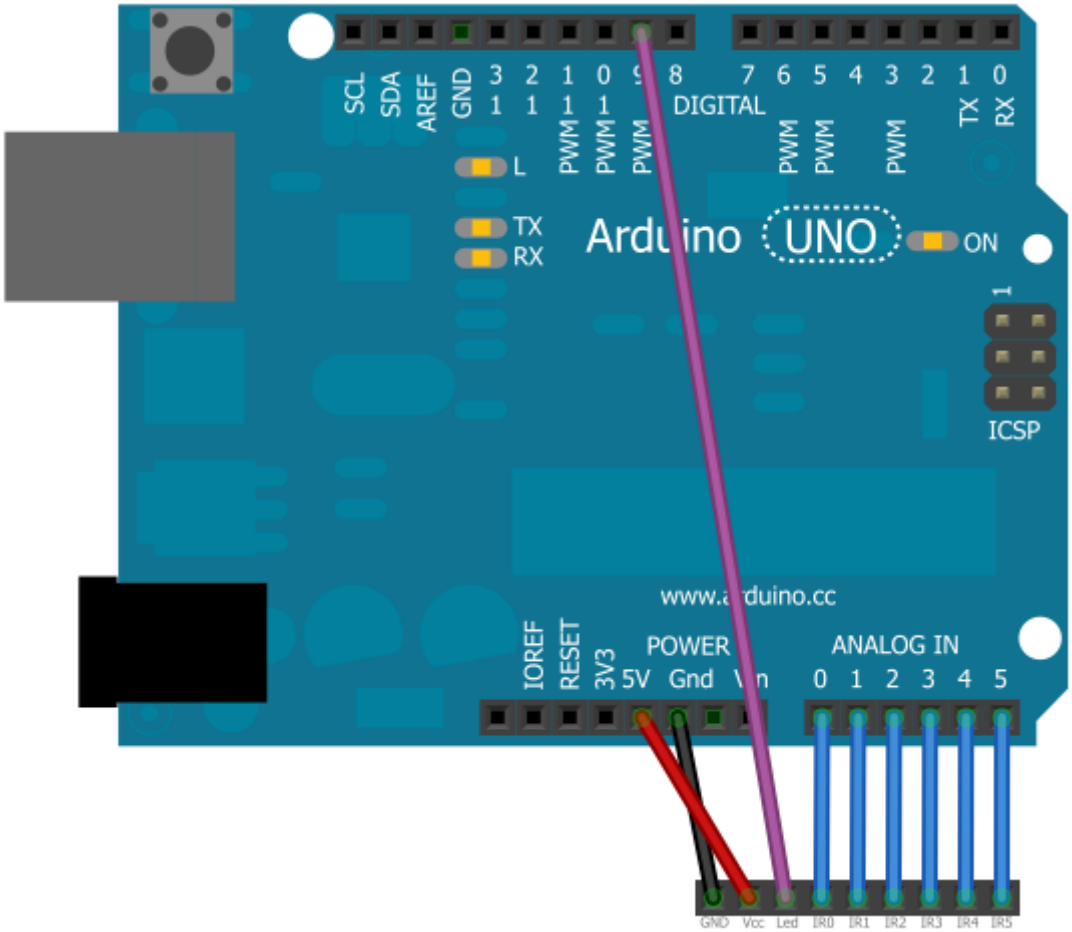


Figura 3-40: Conexión para la prueba de funcionamiento del QTR-8RC

Figura 3-41: Comprobación de que los sensores infrarrojos se encuentran funcionando

Después se realizó el *sketch* para comprobar el funcionamiento de los sensores. Como se ha indicado, hay que realizar una serie de pasos para que los sensores tomen medidas de forma correcta. Aprovechando que este sensor incluye una librería, se empleó la función *qtrrc.read* de la librería *QTRSensors*. El diagrama del *sketch* que se empleó (ANEXO A.2) es el siguiente:

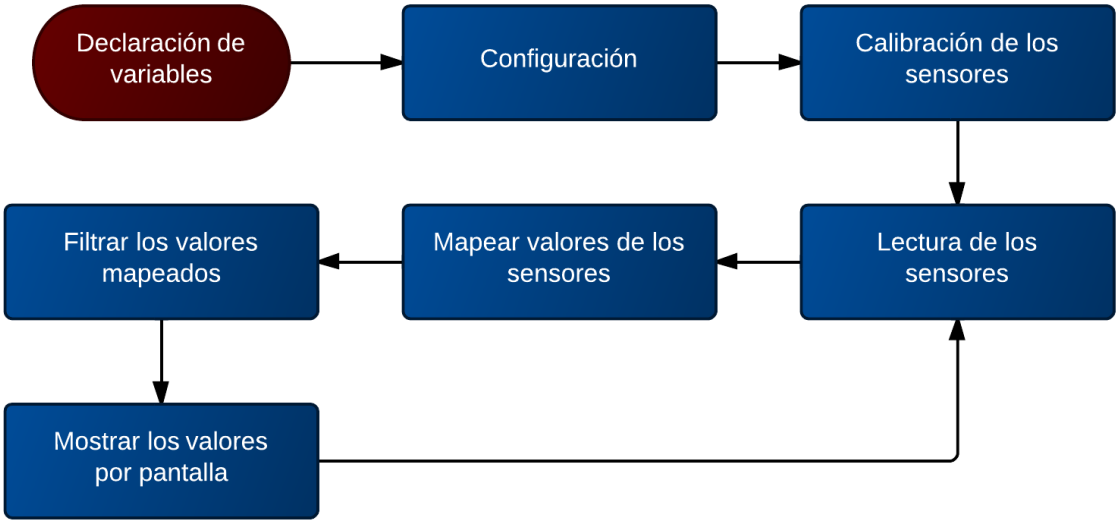


Figura 3-42: Esquema del *sketch* empleado para la prueba del QTR-8RC

Es necesario hacer notar que es necesario calibrar los sensores. Cada sensor tiene diferentes valores máximos y mínimos, por lo que a menos de que se guarden los valores en la memoria EEPROM, cada vez que se acceda a los sensores habrá que calibrarlos.

A continuación se muestra la pantalla del puerto serie. En vez de realizarlo sobre una superficie blanca y con una pista negra, se ha empleado el dedo para ir tapando los sensores:

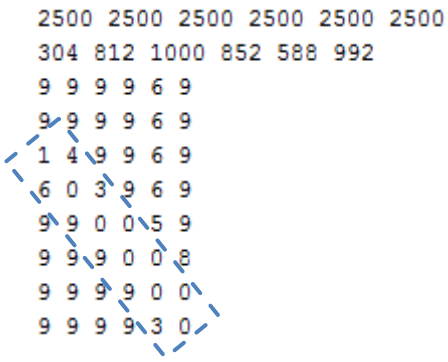


Figura 3-43: Resultado de la prueba del QTR-8RC

En la primera línea se muestran los valores máximos que ha visto cada sensor. En la segunda línea están los valores mínimos. Después, al haber mapeado los valores entre 0 y 9, se muestra el camino que ha ido siguiendo el dedo (allí por donde pasaba el dedo se

muestra un 0). Como los resultados de la prueba resultaron satisfactorios, se concluyó que el sensor funcionaba de forma correcta y estaba listo para su utilización en el robot.

3.6. BLUETOOTH

Desde un primer momento uno de los objetivos más ambiciosos del proyecto fue el tema de la comunicación del robot con un ordenador. Un robot sigue líneas no es como por ejemplo un brazo robótico que tiene su base siempre fija en el mismo sitio, si no que este se encuentra en movimiento, por lo que la conexión con el ordenador a través de cables se desechó desde un primer momento.

Primero se pensó en guardar los datos dentro de una tarjeta SD y después volcar los datos a un ordenador ya que existen *shields*[26] para Arduino los cuales permiten grabar y leer datos desde una tarjeta SD. Se desechó esta opción por dos razones. La primera de ellas era que se buscaba un sistema más directo en el que se pudiera comprobar el funcionamiento del robot en vivo. La segunda de ellas, una vez más, fue el tamaño de estos *shields* ya que son bastante grandes.

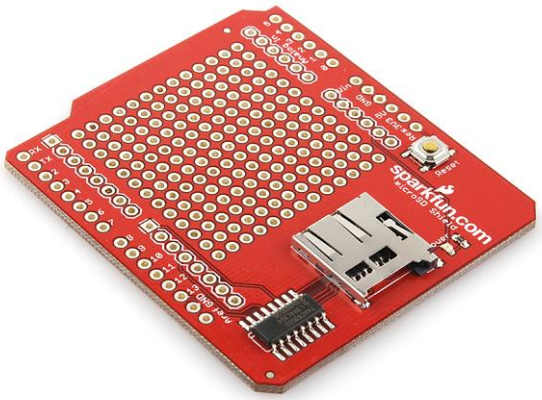


Figura 3-44: microSD Shield

Tras desechar la opción de emplear un microSD Shield se pensó en utilizar tecnología bluetooth para comunicar el robot con diferentes periféricos como por ejemplo podía ser un ordenador o un teléfono móvil.

Primero se pensó en poner directamente un *dongle* bluetooth USB mediante un puerto USB comunicado con el Arduino. Pero no se pudo llevar a cabo esta idea ya que la placa Arduino Pro Mini carece de conversor Serie-USB, por lo que no se podría llevar a cabo dicha comunicación.

Figura 3-45: dongle bluetooth USB

Como alternativa se pensó en utilizar un *shield* que transforma la señal USB a Serial, pero su elevado coste, además de que suponía utilizar muchos pines de la placa Arduino, se desechó esta opción.

Por lo tanto, si se quería que el sistema se comunicara mediante bluetooth se necesitaría un bluetooth específico. Se buscó en las principales páginas de proveedores y todos tenían un precio que oscilaban entre los 35 - 50 $, precio bastante elevado, por lo que se consideró la opción de utilizar un bluetooth genérico chino. Estos tienen precios que oscilan entre 8 – 15 $. Finalmente se optó por uno de estos y que tuviera el envío más rápido.

El modelo elegido es el JY – MCU V1.06[27]. Sus principales características son que su tamaño es muy reducido (4’4 cm x 1’6 cm x 0’7 cm), está formado por cuatro pines (Vcc, Tierra, TXD y RXD) y posee internamente un regulador de tensión, por lo que se puede alimentar a los 5 V que aporta el Arduino.

Figura 3-46: Módulo bluetooth

3.6.1. CONFIGURACIÓN

El módulo bluetooth venía sin configurar, por lo que lo primero que se hizo fue configurarlo. Para poder configurarlo se deben emplear comandos AT[28]. Los pasos para configurar el bluetooth son los siguientes:

1. Asegurarse de que el módulo no está vinculado a ningún dispositivo, esto es, que la luz roja del módulo esté parpadeando.

2. Dejar libres los pines de la comunicación Serial de la placa Arduino, pines TX y RX.

3. Cargar el programa.

4. Cuando se indique conectar el TX del módulo bluetooth con el RX de la placa Arduino y el RX del módulo bluetooth con el TX de la placa Arduino.

5. Empezar la comunicación con el módulo (AT).

6. Cambiar el nombre del módulo (AT+NAME).

7. Ajustar la velocidad de comunicación (AT+BAUD):

AT+BAUD	Velocidad Transmisión (Baudios)
1	1200
2	2400
3	4800
4	9600
5	19200
6	38400
7	57600
8	115200

Tabla 3.8: Tabla de selección de velocidad de transmisión

8. Configurar el pin de asociación (AT+PIN)

A continuación se ejecutó el siguiente programa (ANEXO A.3):

```
/*********************************************************************************
*                 Cambio de la configuración del módulo bluetooth                *
*                              mediante comandos AT                              *
*********************************************************************************/

//Variable para control de configuración
boolean conf = false;
int espera = 1000;

void setup(){
   pinMode(13,OUTPUT);      // Pin13 para control del proceso
   Serial.begin(57600);     // Velocidad del módulo bluetooth
   digitalWrite(13, LOW);   // Apagamos el LED 13
}

void loop(){
  if (conf){
    // Parpadeo para verificar su funcionamiento
    digitalWrite(13, HIGH);   // Enciende el LED
    delay(espera);            // Espera
    digitalWrite(13, LOW);    // Apaga el LED
    delay(espera);            // Espera
  }
  else{
    digitalWrite(13, HIGH); // Empieza el tiempo de espera para reconectar
    Serial.println("Configuración del módulo bluetooth");
    Serial.println("Conecte el TX del módulo BT al RX del Arduino y el RX del
módulo BT al TX del Arduino");
    delay(15000);           // Espera de 15 segundos (se puede cambiar)
    digitalWrite(13, LOW);  // Tiempo de espera agotado para reconectar

    Serial.print("AT");     // Empieza la comunicación por el módulo BT
    delay(espera);          // Espera para el envío de comandos

    Serial.print("AT+NAMEBTarduino"); // Nombre del dispositivo
    delay(espera);                    // Espera para el envío de comandos

    Serial.print("AT+BAUD7");   // Establecemos la velocidad en 57600
    delay(espera);              // Espera para el envío de comandos

    Serial.print("AT+PIN4242");  // Establecemos el pin de vinculación
    delay(espera);               // Espera para el envío de comandos

    //En este punto debe estar configurado
    digitalWrite(13, HIGH); // Todo ha funcionado según lo esperado
    conf = true;     // Para no volver a configurarlo, salvo que se resetee
  }
}
```

Con esta configuración el nombre del módulo bluetooth será BTarduino, funcionará a 57600 baudios y el pin de vinculación será 4242.

3.6.2. PRUEBAS

Tras la configuración del módulo bluetooth se pasó a probar el módulo. Para ello se ideó un *sketch* a través del cual se manejara el LED de una placa Arduino UNO. Según el valor que se recibiera se encendería el LED, se apagaría o parpadearía. Para poder realizar la comunicación con la placa Arduino se empleó la aplicación *BlueTerm*[29] para Android que está disponible en su *Play Store*. Esta aplicación lo que hace es enviar

y recibir datos entre el móvil y el dispositivo con el que esté vinculado. Así se empleó el siguiente montaje y se volcó el siguiente código en la placa de Arduino (ANEXO A.4).

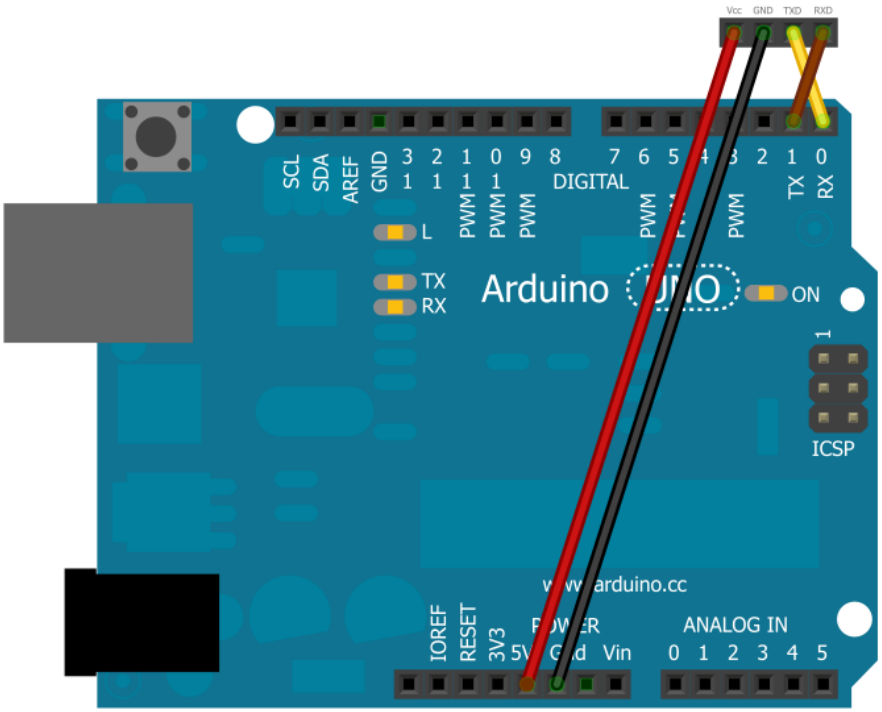


Figura 3-47: Conexión para la prueba de funcionamiento del módulo bluetooth

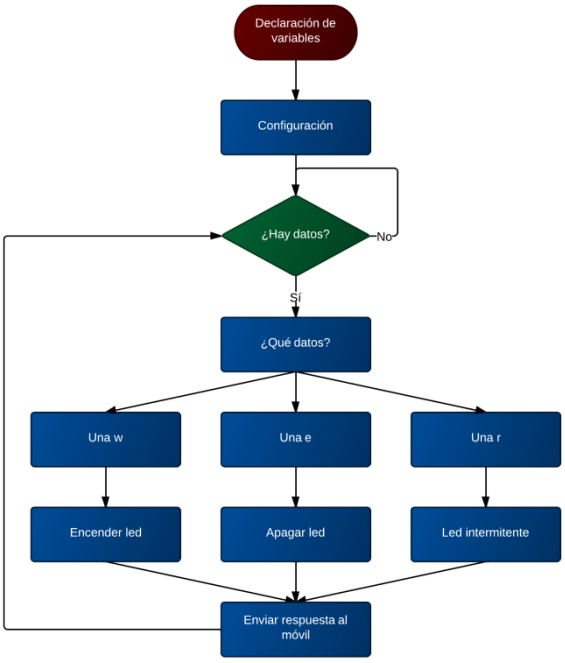


Figura 3-48: Diagrama del *sketch* empleado para la prueba de funcionamiento del módulo bluetooth

Como el resultado resultó ser satisfactorio, se consideró que el módulo bluetooth ya estaba listo para ser utilizado en el robot.

Figura 3-49: Pantalla obtenida en el dispositivo Android

3.7. ALIMENTACIÓN

3.7.1. ESTUDIO DE LOS COMPONENTES

A continuación se va a representar en una tabla las tensiones que requieren cada uno de los componentes que van a conformar el robot:

Componente	Voltaje mínimo	Voltaje máximo	Voltaje recomendado	Consumo (mA)
Arduino Pro Mini	6	20	De 7 a 12	-
Motor	3	9	6	360
L6205N	8	52	De 8 a 52	10
QTR-8RC	3'3	5	5	100
Bluetooth	3'3	5	5	-

Tabla 3.9: Estudio de voltajes de cada componente

Tras un breve análisis se puede llegar a la conclusión de que tanto al bluetooth como al QTR-8RC los puede alimentar las salidas Vcc del Arduino, ya que esta ofrece 5 V. Del resto el voltaje mínimo más alto es el del L6205N (8 V) mientras que el voltaje máximo más bajo es el del motor (9 V). Por lo tanto, la fuente que alimente a todo el sistema debe de estar comprendida entre 8 y 9 V.

3.7.2. ELECCIÓN: PILAS FRENTE A BATERÍAS LIPO

En primer lugar se pensó en utilizar baterías de tipo LiPo ya que ofrecen una gran autonomía y una elevada capacidad de descarga en relación con su peso. Pero todas las que se encontraron tenían un tamaño y un peso excesivo para este proyecto, además de que el pack batería + cargador resultaba caro. Incluso, según el modelo, llegaban a ser más grandes y pesadas que todo el resto del robot. Por lo que finalmente se desechó el uso de baterías.

Figura 3-50: Batería de 11V

Tras rechazar el uso de baterías inmediatamente se pensó en emplear pilas de 9 V. Estas tienen un tamaño y un peso reducido y su coste es relativamente pequeño, además de aportar el voltaje que se necesita para el proyecto. El gran inconveniente de estas

pilas es su duración. Así que para solventar este problema, finalmente se emplearon pilas recargables de 9 V de Ni-MH (Niquel-Metal Hidruro).

Figura 3-51: Pilas recargables de 9V

3.8. RESUMEN

En la siguiente tabla se detalla la lista de componentes empleados en el sistema:

Cantidad	Componente	Descripción
1	Arduino Pro Mini	Microcontrolador
1	Conversor SERIE-USB FTDI232	Programador
2	Pololu Wheel 32x7 mm White	Ruedas de tracción
1	Tamiya Ball Caster Kit	Rueda de apoyo
2	30:1 Micro Metal Gear motor	Motores
2	Funda protectora para Metal Gear motor	Soporte motores
1	L6205N	Controlador motores
1	QTR-8RC	Matriz de sensores infrarrojos
1	JY - MCU V1.06	Módulo bluetooth
1	Pila 9V recargable	Alimentación del sistema
2	Condensador 120 nF	Condensadores de poliéster
1	Condensador 220 nF	Condensador de poliéster
1	Condensador 15 nF	Condensador cerámico
2	Condensador 6'8 nF	Condensadores cerámicos
2	1N4148	Diodos
2	Resistencia 100 kΩ	Resistencias
1	Resistencia 120 Ω	Resistencia

Tabla 3.10: Elementos que compondrán el robot

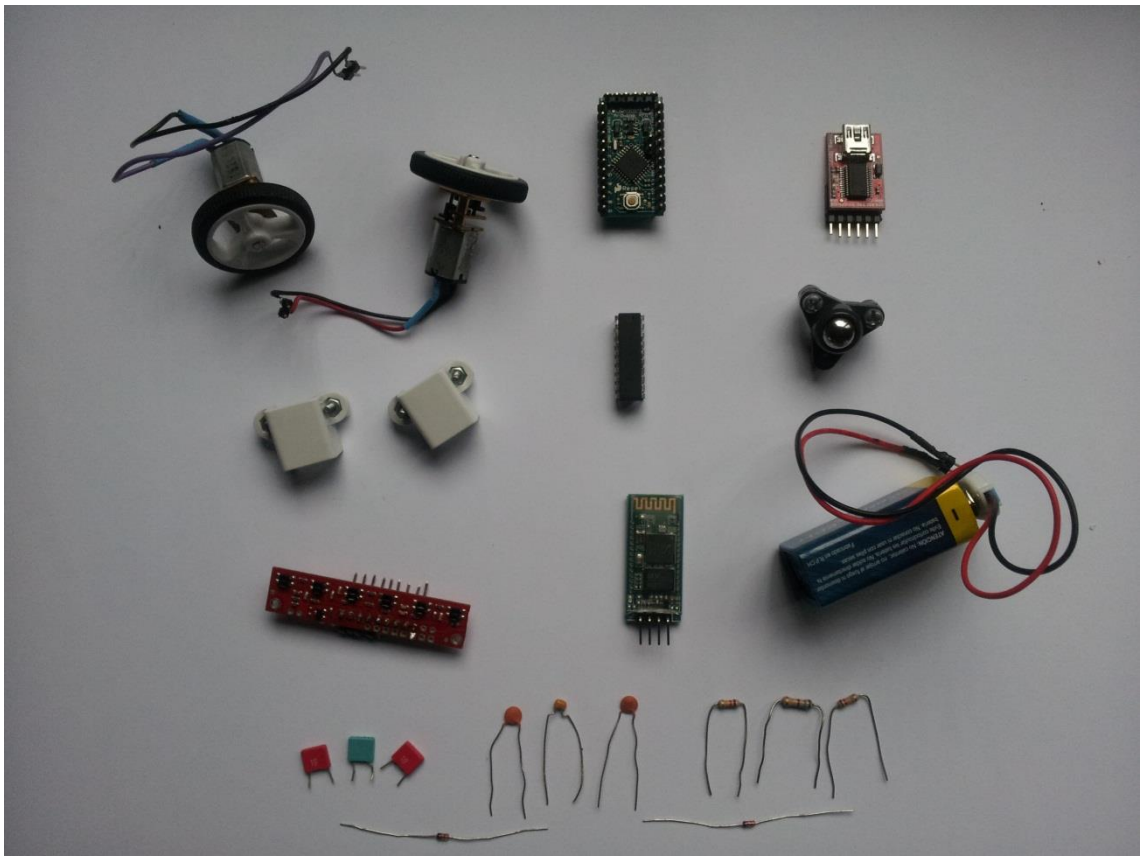


Figura 3-52: Elementos que compondrán el robot

CAPÍTULO 4. DISEÑO

Tras seleccionar todos los componentes que formarían el robot se pasó a diseñar propiamente el robot. Para unir todos los componentes se pensaron dos métodos. El primero de ellos, el más sencillo, consistía en utilizar una placa de topos para montar en ella la mayoría de los elementos y después emplear una base para colocar las diferentes ruedas que componen el robot. El segundo método consistía en diseñar una tarjeta de circuito impreso (PBC, Printed Circuit Board) que diera sustento tanto a los elementos eléctricos como a los mecánicos.

La principal ventaja que ofrece el primer método sobre el segundo es que le da al robot más versatilidad. Si se desea construir un robot para que realice diferentes tareas es más interesante esta opción ya que cambiando la base sobre la que va el motor se puede conseguir que este funcione de una forma o de otra. Por otro lado, la principal ventaja que ofrece el segundo método sobre el primero es que se pueden conseguir diseños realmente pequeños. Al eliminar conexiones entre placa y base del robot se gana en espacio, por lo que el diseño del robot resulta más compacto.

Finalmente se escogió la opción de diseñar un PBC en el que se incluyeran todos los componentes del robot. Se estaba intentando realizar el robot más pequeño posible, por lo que esta opción era la más idónea, además de que le daba al robot un aspecto mucho más profesional.

En los siguientes apartados se desarrollará cómo se llevó a cabo el diseño del robot. Destacar que se fue realizando en paralelo tanto el diseño eléctrico como el diseño en sí del PBC. Esto se debió a que hasta no ver físicamente como quedaban los componentes en relación a los demás no se eligieron las conexiones.

Como programas de diseño se empleó para el diseño eléctrico Eagle[30] y para el diseño del PBC DesignSpark[31]. El primero de ellos tiene unas librerías con multitud de componentes, además de ser más llamativo visualmente hablando, por lo que se pensó en él como software para diseño eléctrico. El gran inconveniente de este programa es que es se necesita adquirir una licencia para poder exprimir al máximo el programa. De todas formas, la versión Freeware permite realizar esquemas eléctricos,

por lo que en este aspecto no supone ninguna restricción. El segundo nos permite de una forma muy intuitiva diseñar PBCs, además de ser un software libre. Por lo que se decidió que fuera este el encargado del diseño del PBC. La conexión entre ellos permite convertir archivos de Eagle a archivos de DesignSpark, pero no a la inversa.

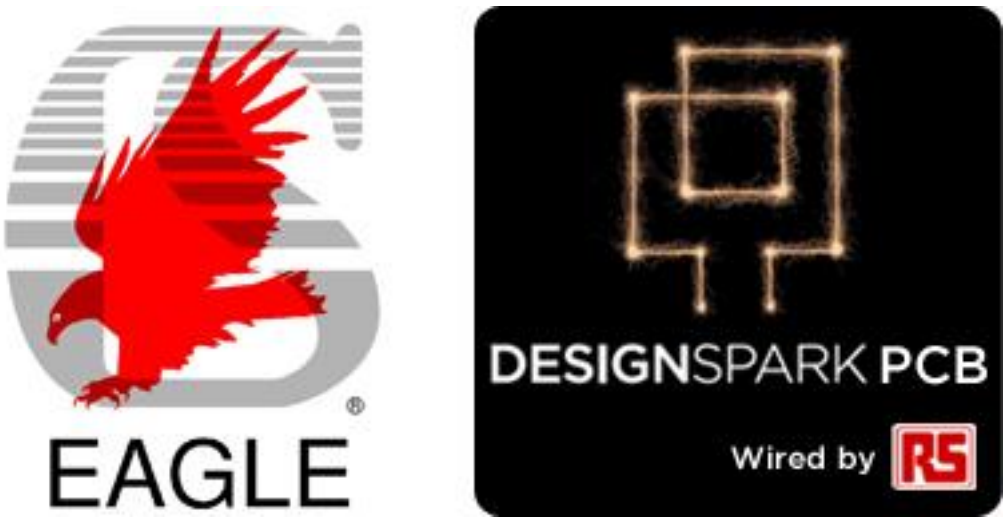


Figura 4-1: Logotipos de Eagle y DesignSpark

4.1. ASPECTOS GENERALES

Lo primero que se pensó al iniciar el diseño del PBC fue la forma de este. Se barajó la opción de que fuera circular ya que era más atractivo visualmente. Sin embargo, a la hora de encajar los componentes, sería más eficiente que el PBC fuera de forma rectangular. De esta forma se decidió que el diseño del PBC fuera rectangular.

Después se analizaron las conexiones que debían de existir entre las diferentes partes que iban a formar el robot.

- **Programador**: este viene diseñado especialmente para aprovechar los pines de la parte superior del Arduino Pro Mini, por lo que emplearían los seis pines superiores del Arduino Pro Mini. Para realizar la conexión de forma correcta solo hay que asegurarse que ambos pines de los extremos se unen con sus pines del mismo nombre (cables amarillos).

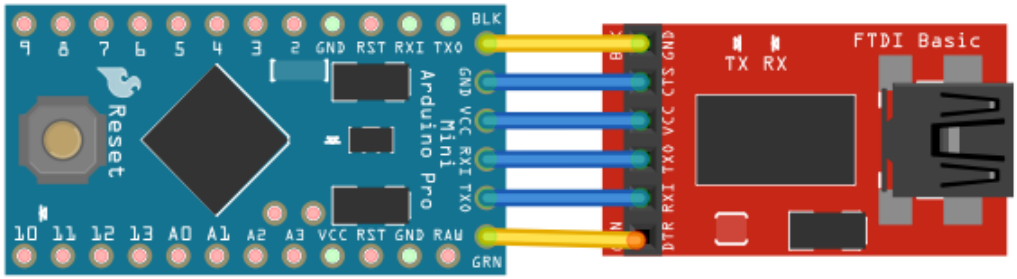


Figura 4-2: Conexión entre programador y Arduino Pro Mini

- **Módulo bluetooth**: comparte los cuatro pines centrales del programador, por lo que se pensó en utilizar los mismos pines.
- **L6205N y sus componentes**: tanto el circuito integrado como los componentes auxiliares que necesita para funcionar necesitan diversas conexiones a la alimentación. Además, se deben conectar cuatro pines PWM y dos pines digitales del Arduino Pro Mini. Por último, se deberá conectar a cada motor mediante dos pines.
- **QTR-8RC**: además de la alimentación, necesita siete pines digitales del Arduino Pro Mini, sin importar si pueden ofrecer PWM.

Tras este análisis, se observó que sobraba algún pin del Arduino Pro Mini, por lo que se pensó en aprovechar dos ellos añadiendo al sistema dos diodos LED que servirán para ofrecer información en forma de luz, como por ejemplo que haya terminado cierto proceso, o que el robot debe girar. Para evitar que el diodo se queme se incluirán junto a ellos dos resistencias de 220 Ω para limitar así la corriente a unos 20 mA.

Además, al necesitarse un total de 13 pines digitales, y al estar siendo utilizados los pines digitales 0 y 1 (TX0 y RX1 respectivamente) se necesitarían configurar al menos dos pines analógicos como digitales.

Después se hizo un primer boceto en el que se colocaron todos los componentes y que sirvió para hacerse una primera idea del diseño final que se buscaba.

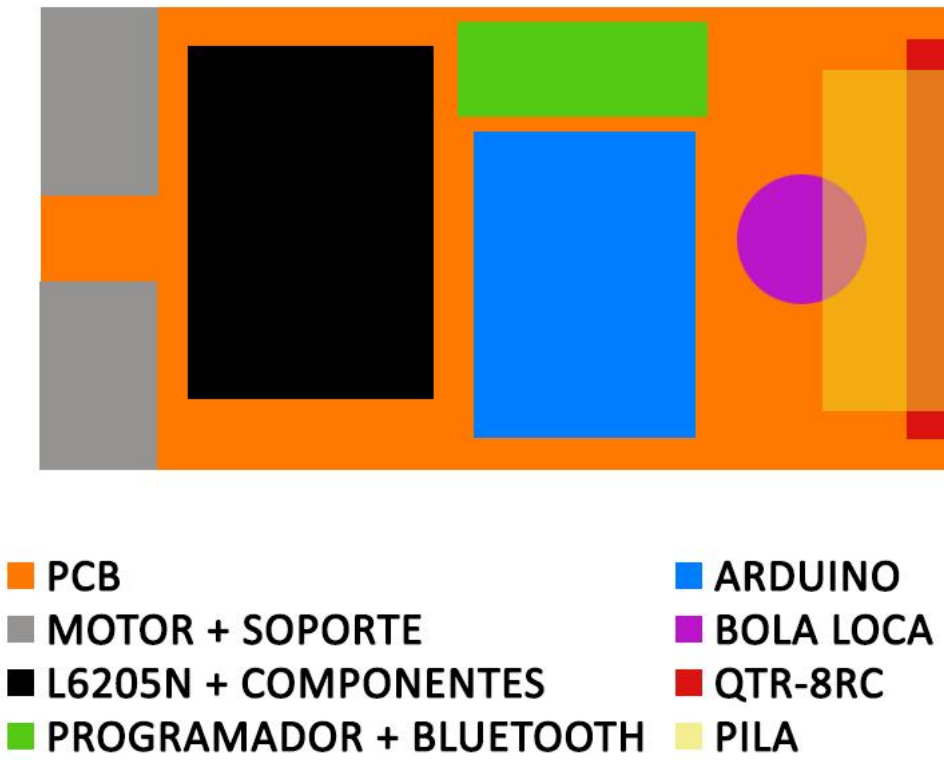


Figura 4-3: Boceto del diseño a realizar

Lo primero que se situó fue el Arduino Pro Mini. Es el componente que más conexiones tiene, por lo que se pensó situarlo en la parte central del PBC para que fuera más accesible al resto de componentes.

Como ya se ha explicado, tanto el programador como el bluetooth comparten las conexiones de la parte superior del Arduino Pro Mini, por lo que se pensó en que estuvieran situados justo en ese lateral del Arduino Pro Mini.

Para anticipar los movimientos que deberá hacer el robot se situó el QTR-8RC en la parte delantera del robot para que detecte cuanto antes un cambio en la línea.

Se situaron las ruedas de tracción en la parte posterior del PBC mientras que la rueda loca se situó en la parte delantera del mismo. De esta forma, al tener tres puntos de apoyo lo suficientemente lejanos se conseguiría dar al robot toda la estabilidad que necesitase.

El L6205N tiene diversas conexiones tanto con el Arduino como con los motores, por lo que se pensó situarlo en una zona intermedia de ambos componentes. Los componentes que necesita este circuito integrado se colocaran en los alrededores de él.

Se pensó en colocar la pila en la parte superior del PBC que dejasen libre tanto la bola loca como el QTR-8RC y de esta forma ahorrar espacio. Estos dos elementos irán por debajo del PBC mientras que la pila irá en la parte superior.

Por último se pensó que todo el plano del PBC formara un plano de masa, de esta forma daría más estabilidad eléctrica al conjunto.

4.2. ARDUINO PRO MINI

Ya se ha indicado que lo primero que se iba a fijar iba a ser el Arduino Pro Mini. Se fijará su posición y después se colocarán el resto de elementos en función de él. Es el cerebro del robot por lo que es su parte más importante, por lo que habrá que tomar especial cuidado en su colocación.

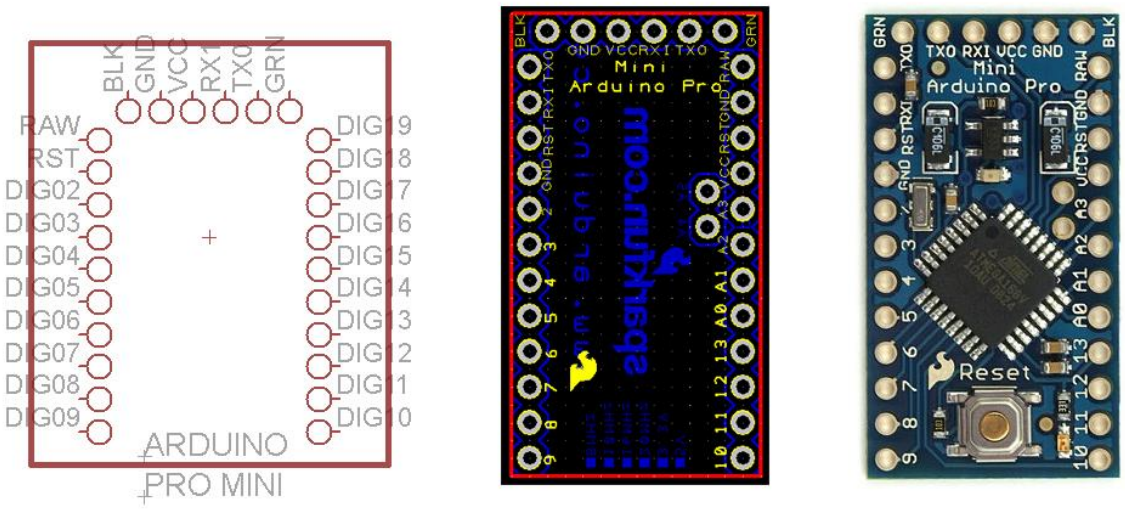


Figura 4-4: Diferentes representaciones del Arduino Pro mini. De izquierda a derecha: Eagle, DesignSpark, realidad

Para poder realizar los diferentes diseños se han empleados los esquemas que hay disponibles en la página de Arduino. El esquema eléctrico era demasiado detallado, por

lo que se ha modificado y se ha simplificado. Para el diseño del PBC se ha empleado tal cual y no se ha hecho ninguna modificación.

Según el boceto realizado se situará entre el QTR-8RC y el L6205N. Este último necesita cuatro conexiones PWM. Las salidas PWM del Arduino Pro Mini son las salidas digitales 3, 5, 6, 9, 10 y 11; por lo que el lado izquierdo del Arduino Pro Mini tiene más salidas PWM. De esta forma se decidió que este lado fuera el que estuviera en el lado del L6205N.

Al otro lado del Arduino Pro Mini se encuentra el QTR-8RC. La posición de este también estaba fijada si se quería que los sensores estuvieran lo más adelante posible. Pero tal y como estaba situado el Arduino con respecto a él, las líneas de Vcc y de GND tenían que dar mucha vuelta, es por ello por lo que se decidió girar el Arduino sobre sí mismo para que los pines Vcc y GND quedaran más accesibles. De esta forma, visualmente el Arduino Pro Mini quedaría boca abajo.

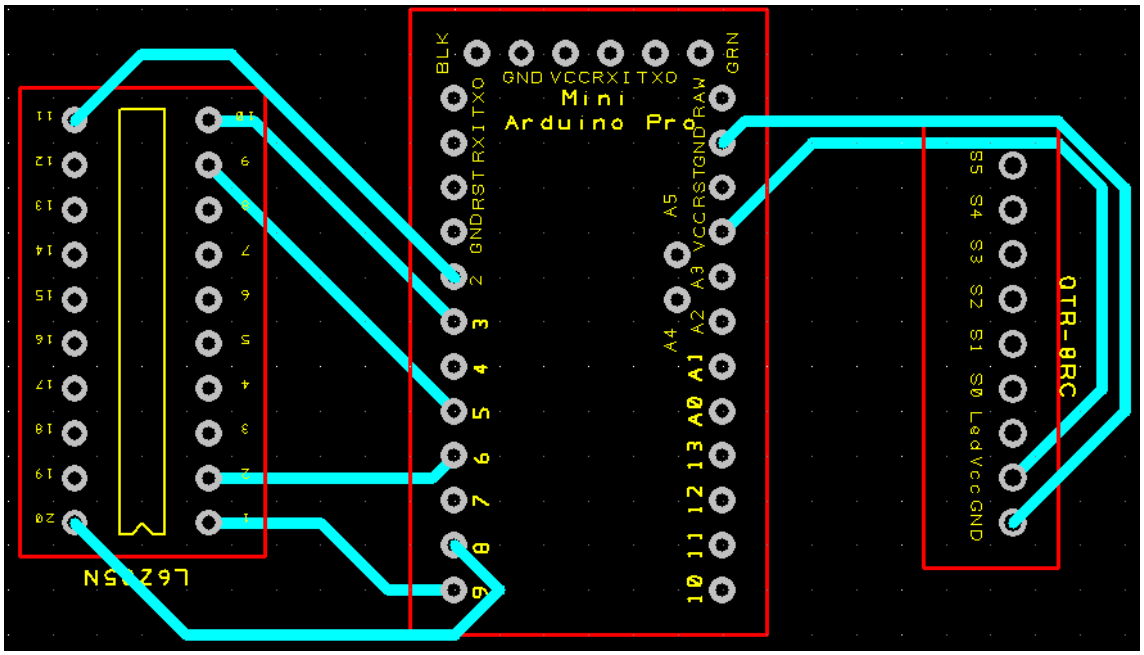


Figura 4-5: Problema de alimentación del QTR-8RC

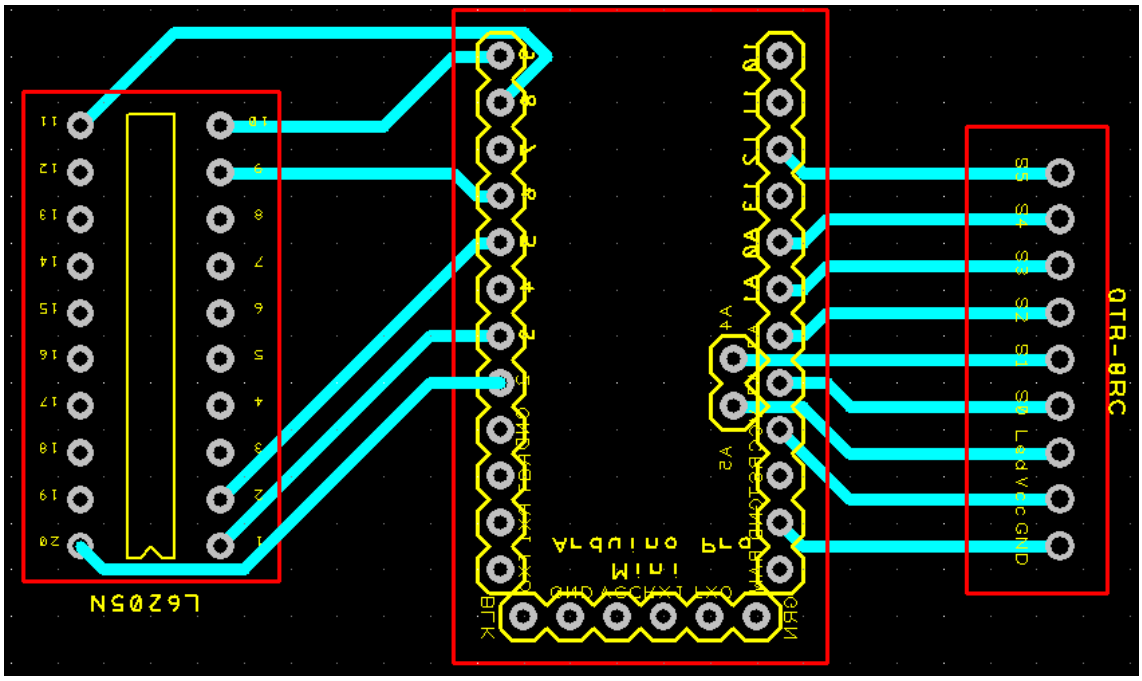


Figura 4-6: Conexionado entre el Arduino Pro Mini y el L6205N y el QTR-8RC

De esta forma se fijó la posición relativa del Arduino Pro Mini con respecto al L6205N y al QTR-8RC.

4.3. PROGRAMADOR Y BLUETOOTH

Tanto el programador como el módulo bluetooth comparten la disposición de los pines con la parte superior del Arduino Pro Mini, así que determinar su posición una vez habiendo fijado la del Arduino fue muy sencillo.

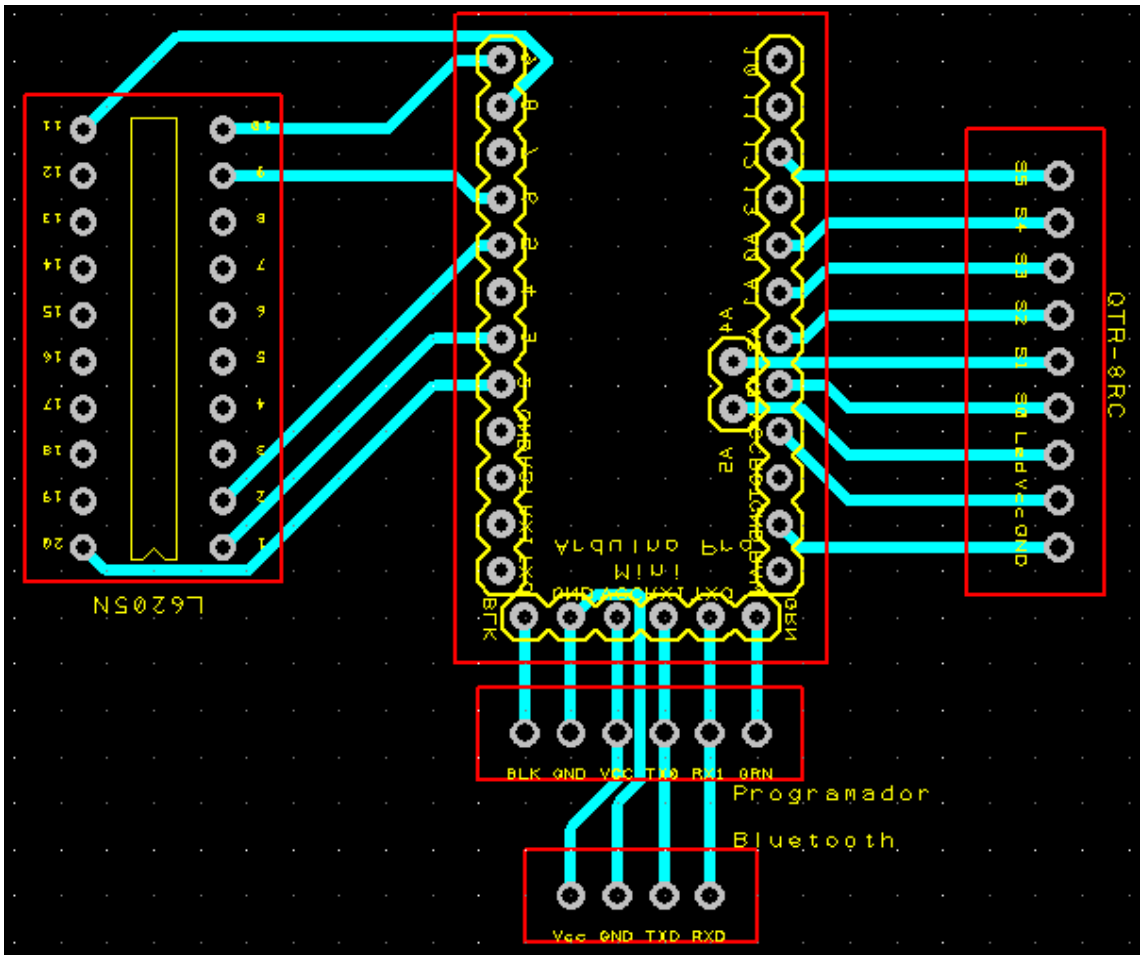


Figura 4-7: Conexión entre Arduino Pro Mini y programador y módulo bluetooth

Solo habría que tener especial cuidado en que en el módulo bluetooth los terminales Vcc y GND están al revés.

4.4. L6205N Y MOTORES

Tanto el L6205N, como sus componentes y los motores fueron las partes del robot que más se tardó en encajar. Al ser un gran número de elementos, resultó difícil encontrar una combinación de disposiciones para que ninguna línea se cruzara. Junto a esta parte también se incluyó el interruptor que gobierna el sistema.

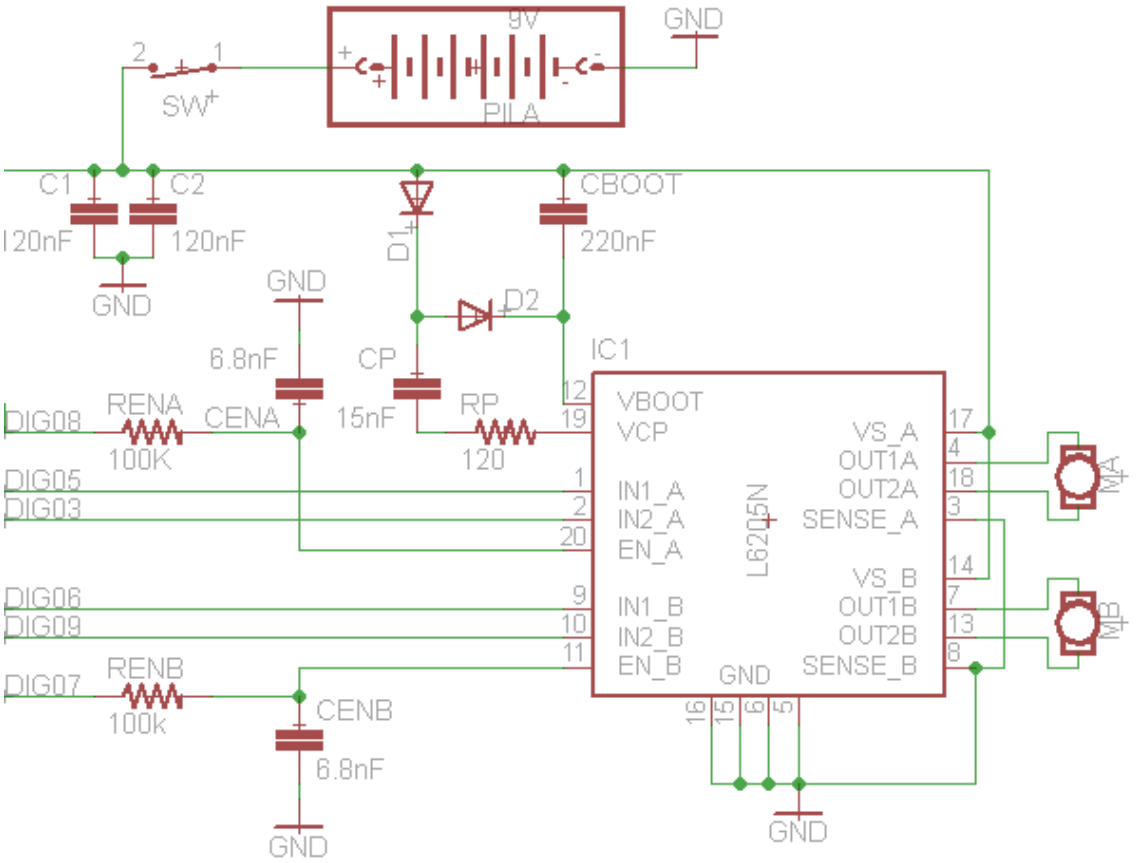


Figura 4-8: Esquema en Eagle del L6205N y los componentes que necesita

En primer se situaron los motores. Se determinó que la anchura de ambos motores puestos en línea era mayor que la anchura de la matriz de sensores, por lo que estos determinarían el ancho del robot. También se determinó que estuvieran en la parte superior del PBC. Su parte inferior era de metal, por lo que si se colocaban en la parte inferior del PBC podrían causar un cortocircuito con el plano de masa. De esta forma se midieron los motores y sus fundas y se creó un bloque para poder tener sus dimensiones en el programa. Una vez determinada la posición de los motores quedaba un hueco libre entre ellos, el cual se empleó para situar el interruptor. Después se fueron añadiendo los diferentes componentes del L6205N de tal forma que se fuera ocupando el menor espacio posible.

Tras multitud de pruebas, se llegó a la siguiente configuración.

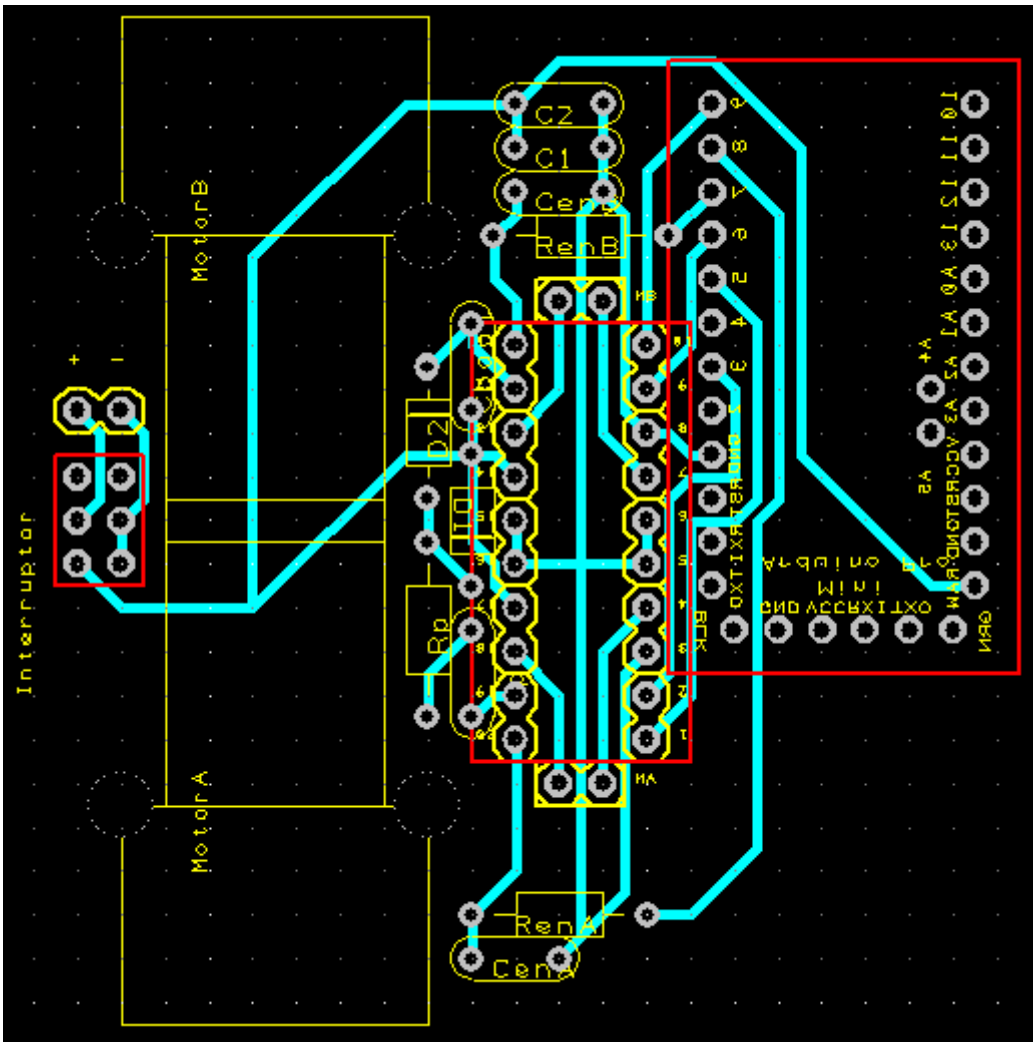


Figura 4-9: Conexión entre el L6205N y sus componentes

4.5. BOLA LOCA, QTR-8RC Y PILA

Después se añadió la bola loca. Esta se une al PBC a través de tres tornillos, pero se hicieron pruebas y la bola quedaba lo suficientemente ajustada con solo dos tornillos. De esta forma se consiguió ahorrar en espacio al suprimir uno de los tornillos.

Pero al incluir estos tornillos había que añadir unos taladros en el PBC que interferían con la conexión entre el Arduino Pro Mini y el QTR-8RC, por lo que hubo que rediseñar la conexión. Además se pensó en ajustar al máximo posible las distancias de la bola loca y del QTR-8RC, por lo que, como inicialmente se había pensado, no entraría la pila en el hueco dejado en la parte superior del PBC. Así que se pensó que la

pila iría en una plataforma superior. Tras varias pruebas, se llegó a la siguiente configuración:

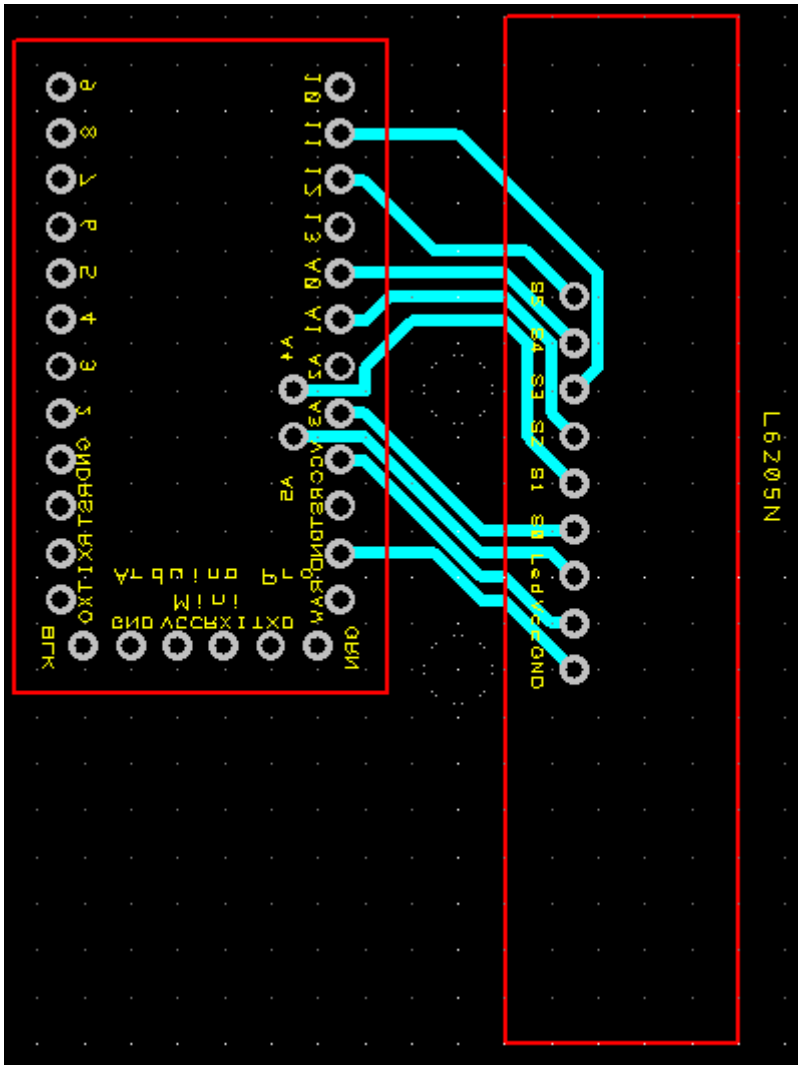


Figura 4-10: Disposición de los taladros de la rueda loca y conexión entre Arduino Pro Mini y L6205N

4.6. RESULTADO FINAL

Por último se añadieron cuatro taladros para poder incluir la plataforma para la pila, además de añadir los LEDs.

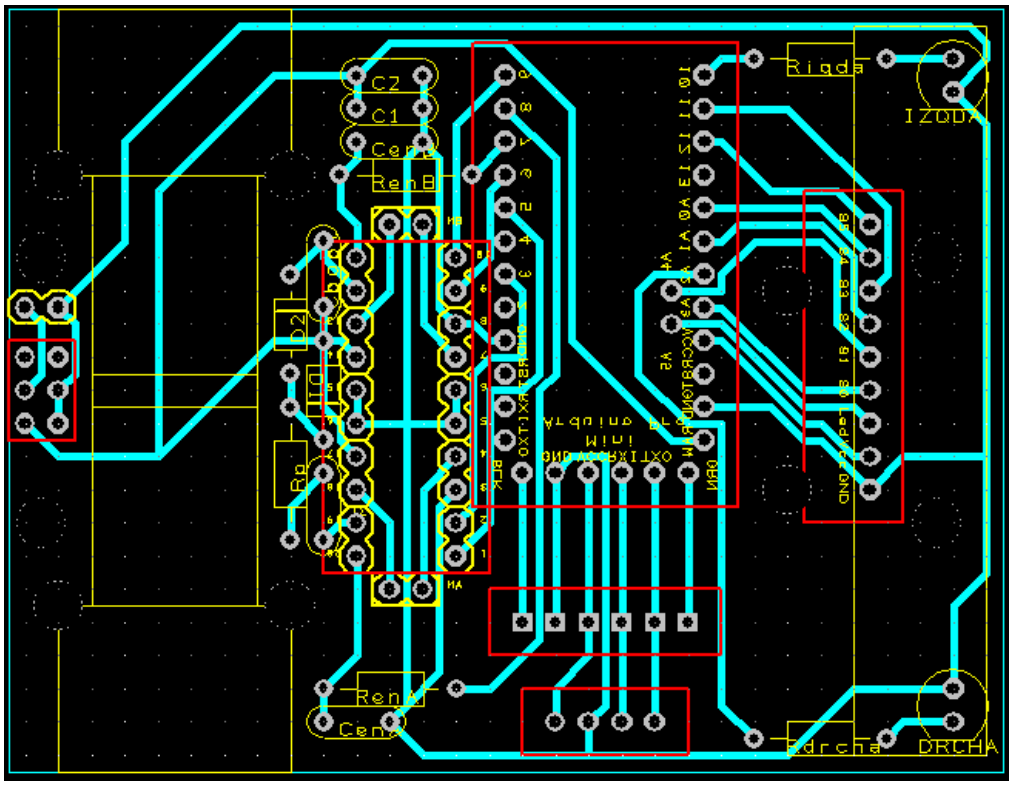


Figura 4-11: Resultado final del PBC

Y se añadió el plano de masa.

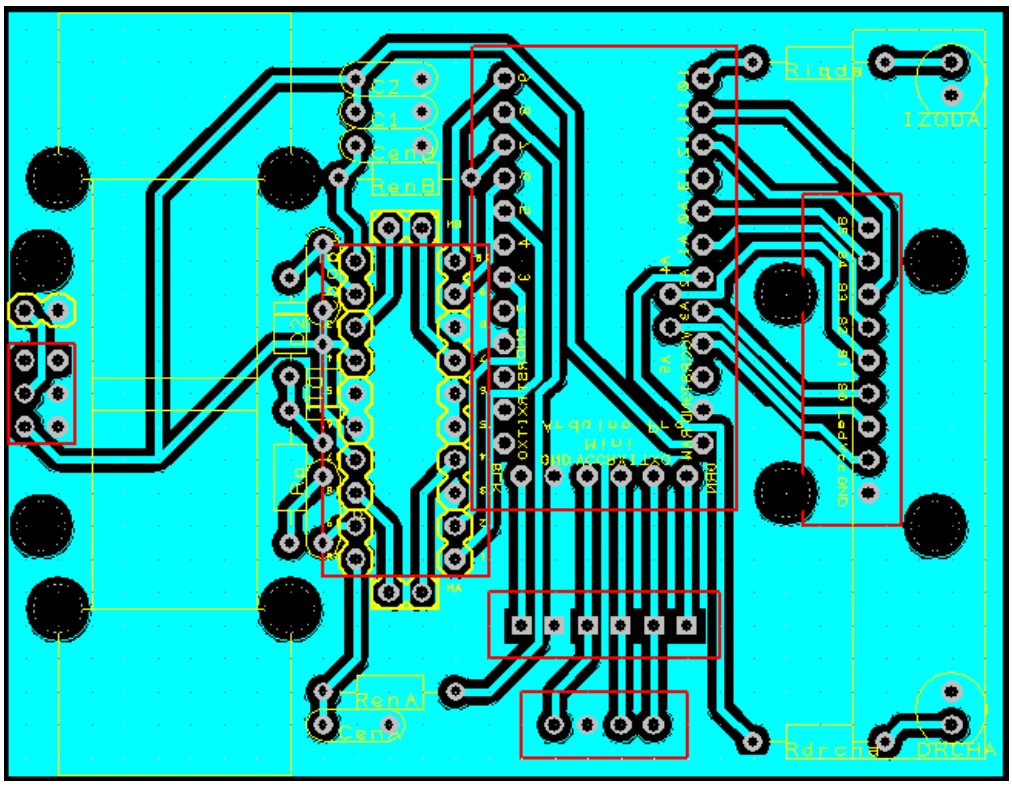


Figura 4-12: Resultado final con plano de masa

De esta forma se consiguió que no hiciera falta añadir ningún cable externo además de los que conectan los motores con el L6205N. Además se realizó un 3D del diseño:

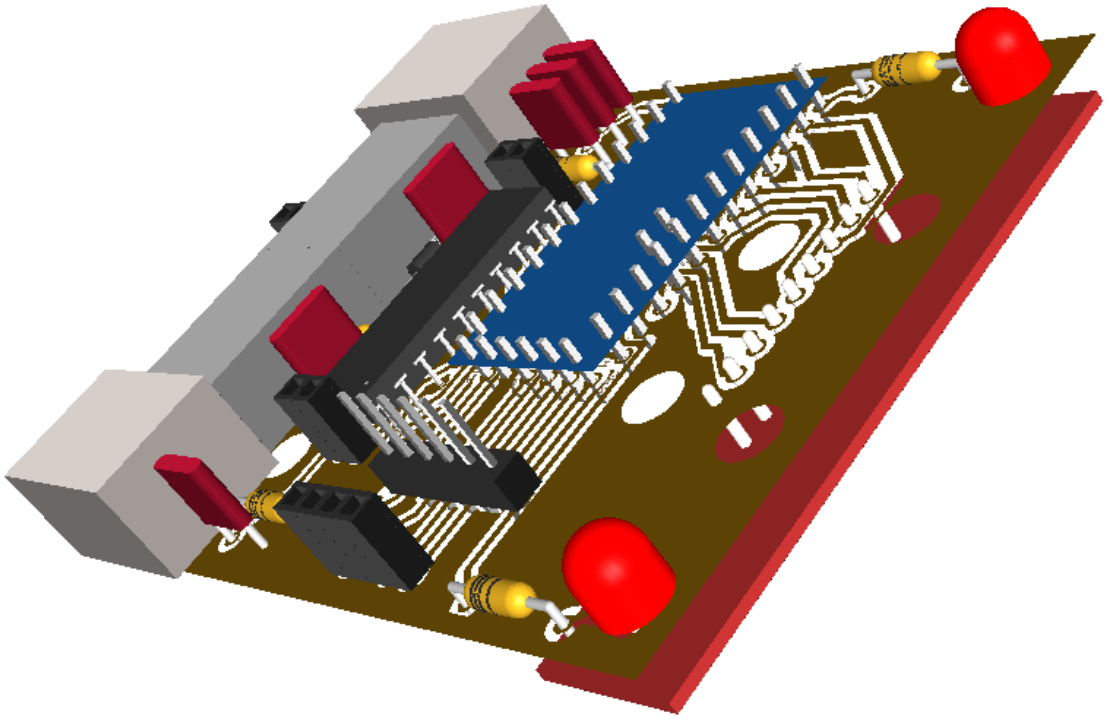


Figura 4-13: Representación 3D del diseño. Vista isométrica 1

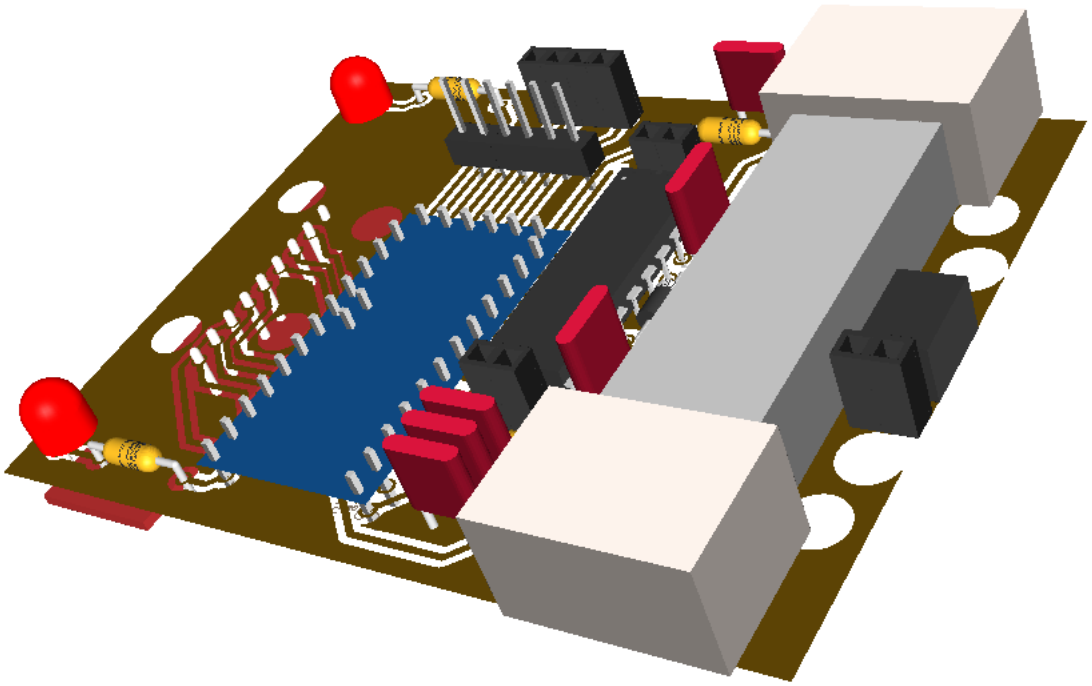


Figura 4-14: Representación 3D del diseño. Vista isométrica 2

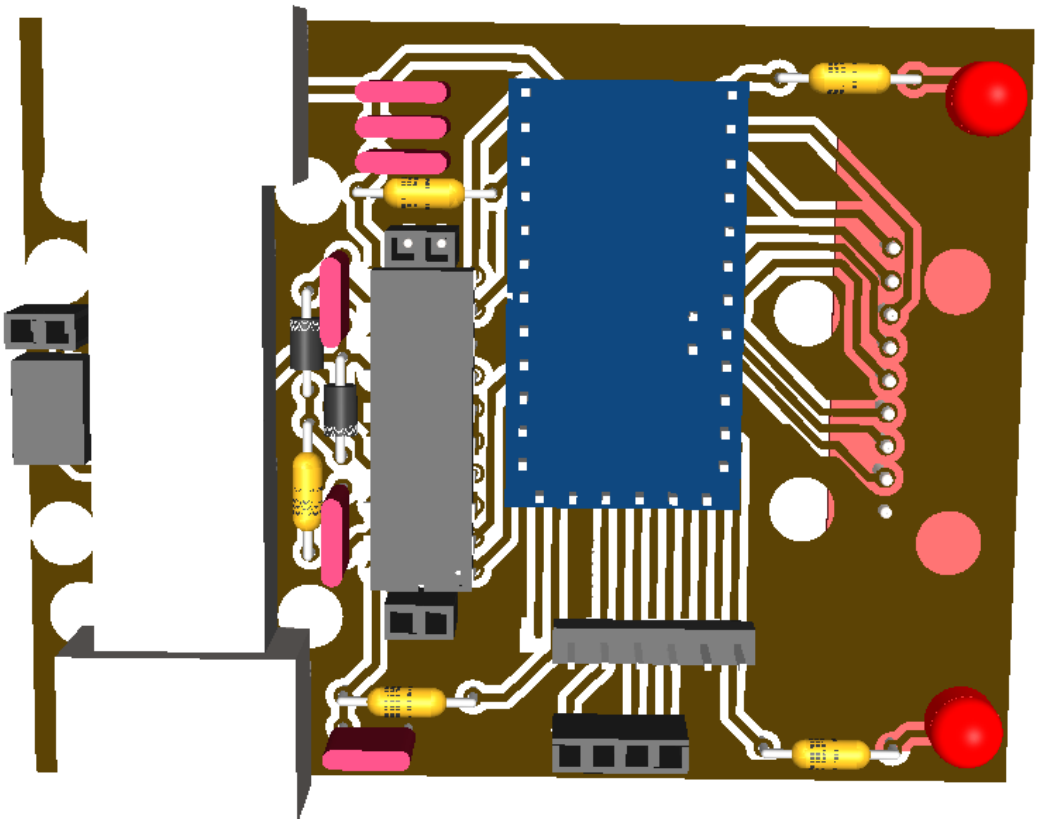


Figura 4-15: Representación 3D del diseño vista de alzado

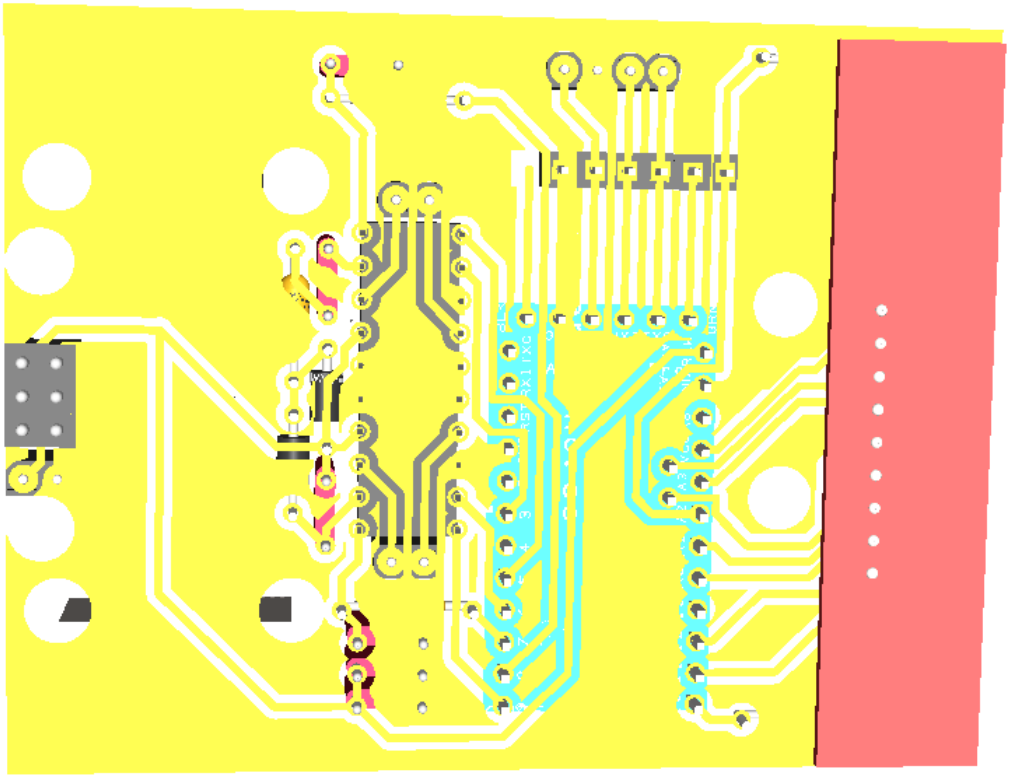


Figura 4-16: Representación 3D del diseño. Vista posterior

Tras esto se montó un prototipo para comprobar que todos los componentes encajaban:

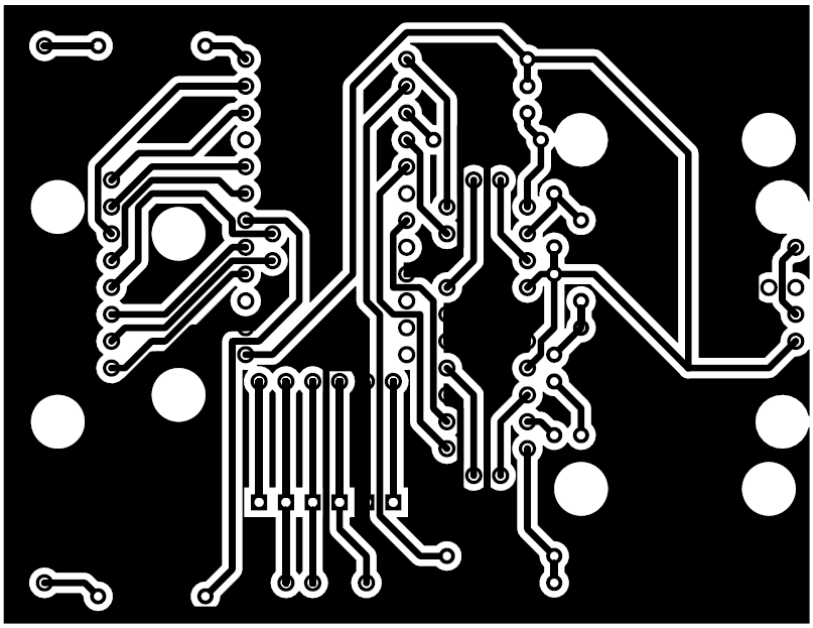


Figura 4-17: Impresión del PBC

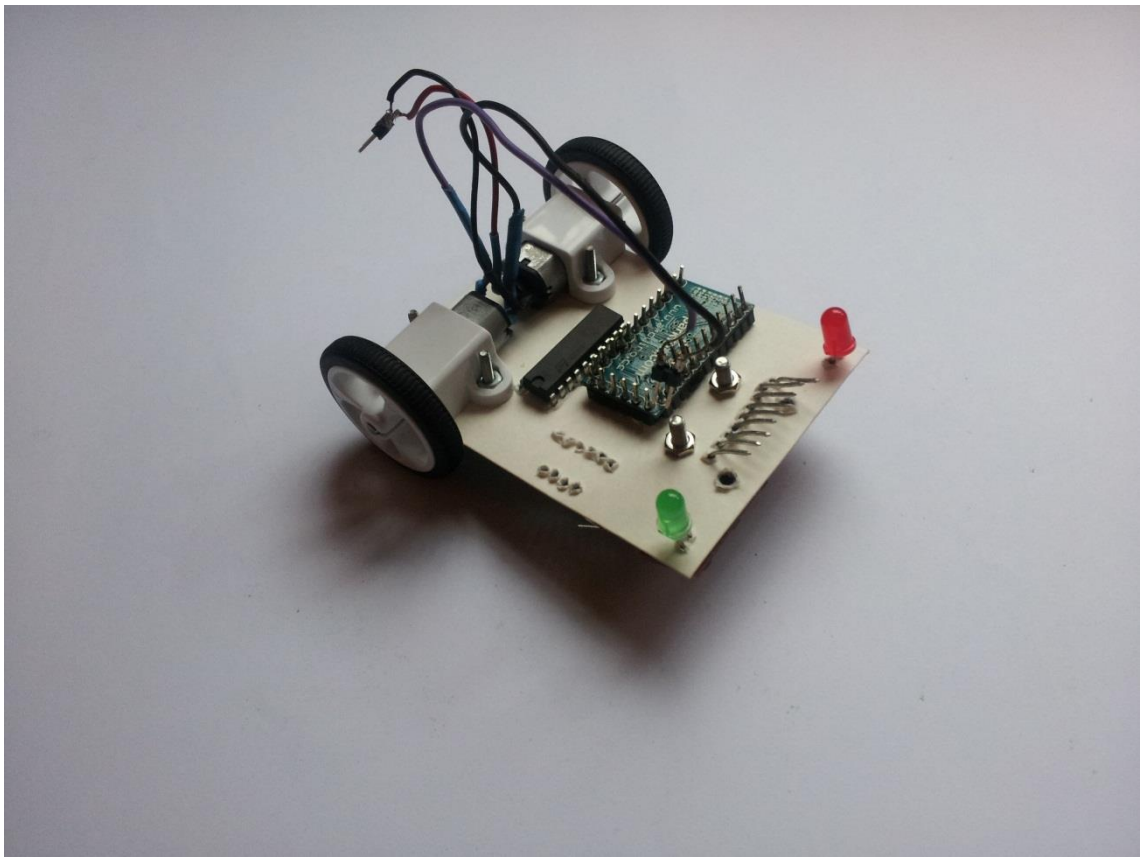


Figura 4-18: Prototipo. Vista isómetrica

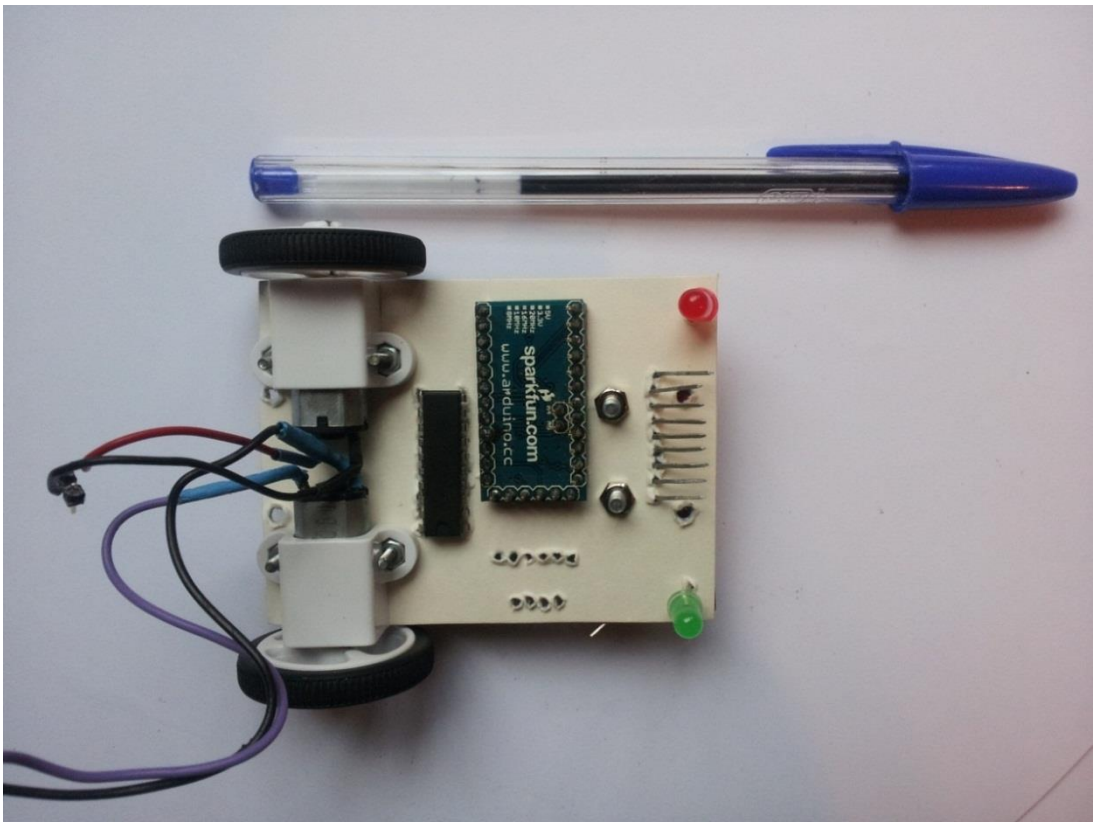


Figura 4-19: Prototipo. Vista de alzado. Comparación con un boli Bic

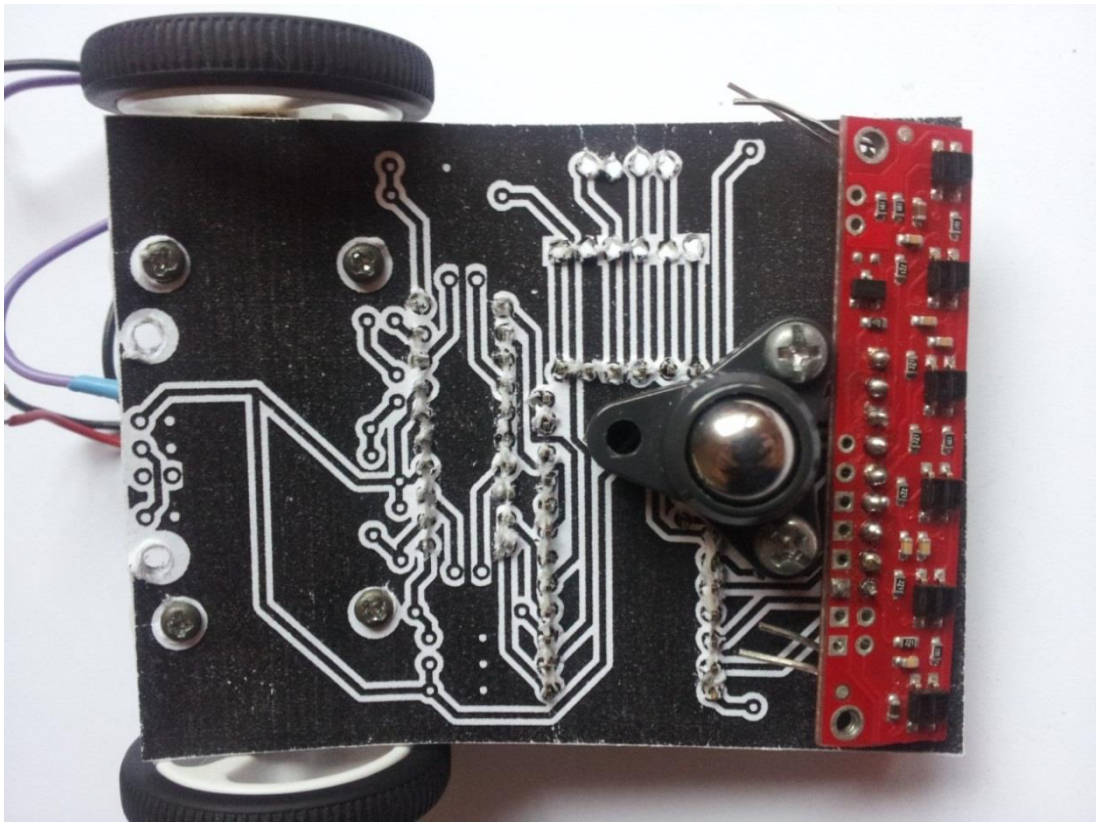


Figura 4-20: Prototipo. Vista posterior

Y finalmente, el PBC construido:

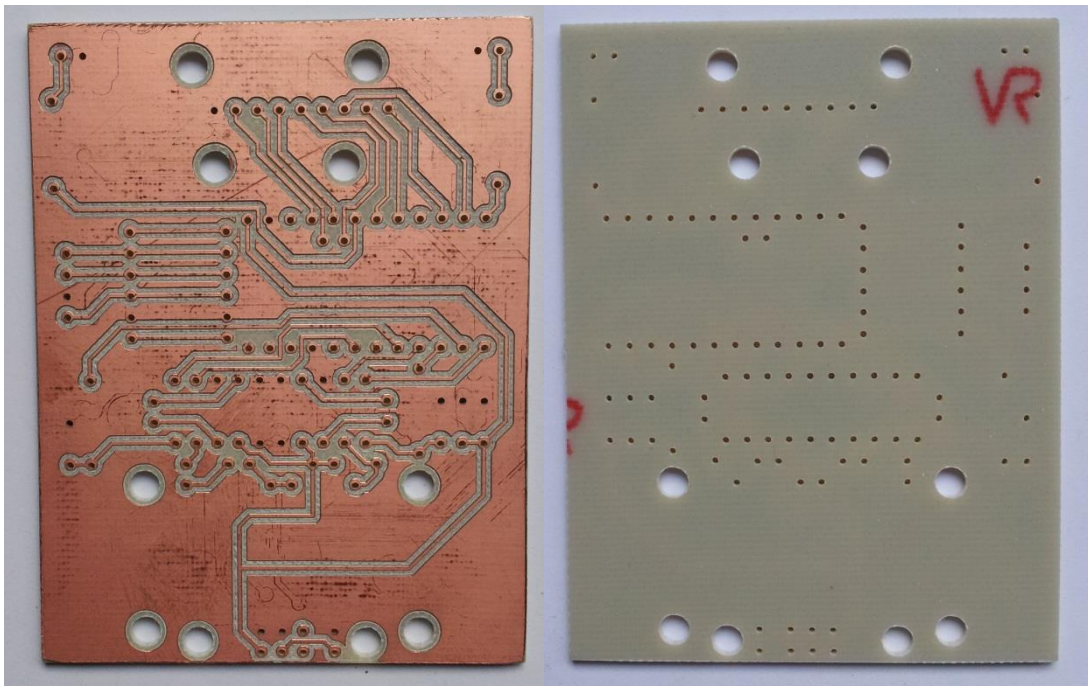


Figura 4-21: PBC. Izquierda vista posterior. Derecha vista de alzado

Por último se completó el esquema eléctrico:

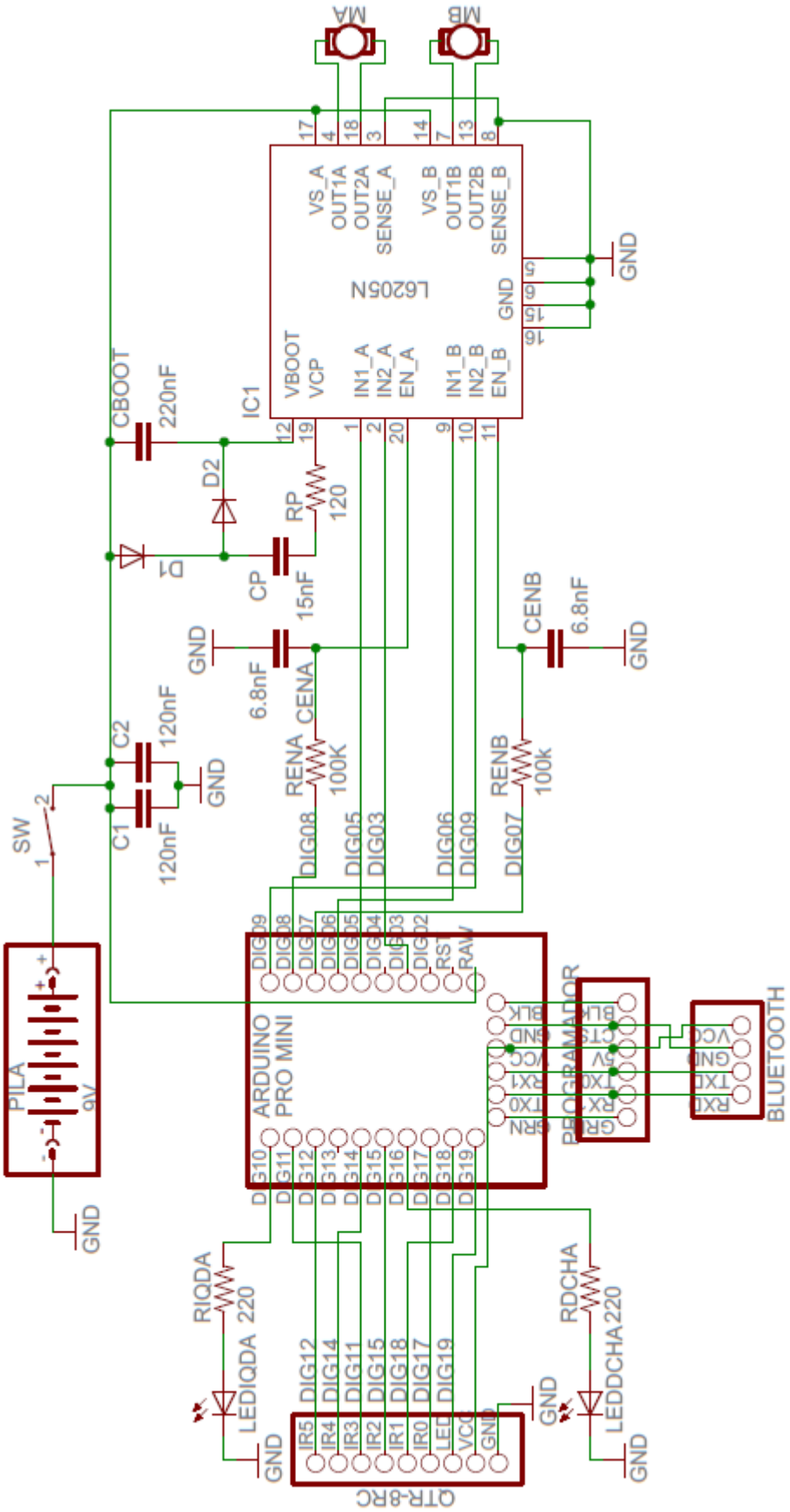


Figura 4-22: Esquema unifilar final del robot

De esta forma la asignación de pines del Arduino Pro Mini quedó como sigue:

Función	Pin Arduino
Comunicación con el programador	pines digitales 0, 1
Comunicación módulo bluetooth	pines digitales 0, 1
Leds delanteros	pines digitales 10, 16
Entradas digitales del QTR-8RC	entradas digitales 11, 12, 14, 15, 17, 18
Control led QTR-8RC	pin digital 19
Enables de los puentes del L6205N	pines digitales 7, 8
PWM de los puentes del L6205N	pines digitales PWM 3, 5, 6, 9

Tabla 4.1: Asignación de pines por función

Pin Arduino	Función
digital 0	Recepción de datos (RX) del programador y del bluetooth
digital 1	Envío de datos (TX) al programador y al bluetooth
digital 2	-
digital 3 (PWM)	Entrada 2 del puente A del L6205N
digital 4	-
digital 5 (PWM)	Entrada 1 del puente A del L6205N
digital 6 (PWM)	Entrada 1 del puente B del L6205N
digital 7	Enable del puente B del L6205N
digital 8	Enable del puente A del L6205N
digital 9 (PWM)	Entrada 2 del puente B del L6205N
digital 10	Led izquierdo
digital 11	QTR-8RC sensor 3
digital 12	QTR-8RC sensor 5
digital 13	Control led interno Arduino Pro Mini
digital 14	QTR-8RC sensor 4
digital 15	QTR-8RC sensor 2
digital 16	Led derecho
digital 17	QTR-8RC sensor 0
digital 18	QTR-8RC sensor 1
digital 19	Control led QTR-8RC

Tabla 4.2: Asignación de pines por pin

CAPÍTULO 5. CONSTRUCCIÓN DEL ROBOT

Después de disponer de todos los componentes que iban a formar el robot se procedió a su construcción. Para ello se hizo primero un análisis de los elementos por separado.

En primer lugar se colocaron los motores. Es de suma importancia que ambos motores se encuentren posicionados de forma correcta, por lo que se pensó que la mejor manera de trabajar con el PBC y los motores era cuando aquel no tuviera todavía sobre él ningún componente

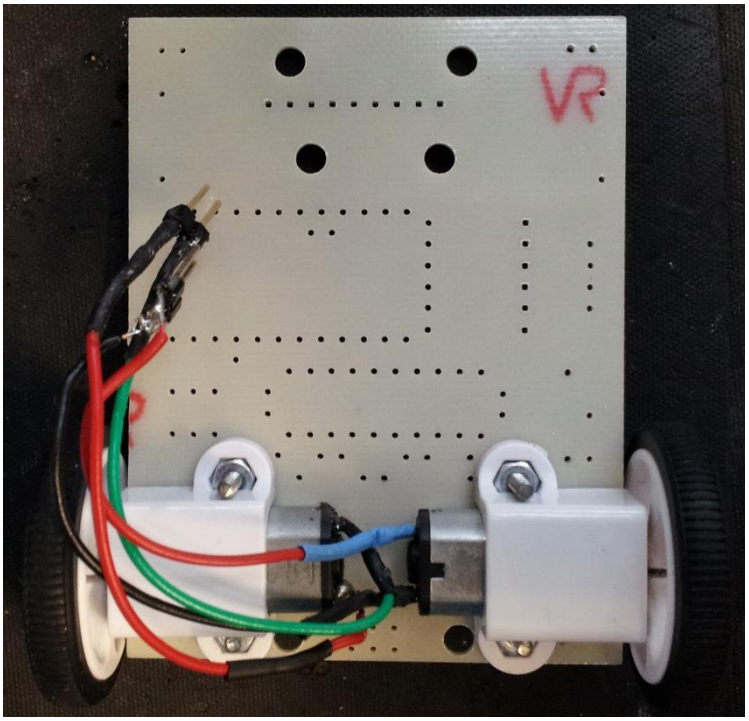


Figura 5-1: Colocación de los motores

Después se pusieron los diferentes conectores. Desde un primer momento se desechó la opción de soldar los componentes más importantes a la placa, tales como el Arduino Pro Mini, el L6205N, el programador, el módulo bluetooth y los pines de alimentación. Además, por temas de diseño, fue imposible incluir en el diseño del PBC las pistas que unen los motores con el L6205N, por lo que también se tuvieron que incluir dichos conectores.

Los conectores se colocaron antes que el resto de componentes ya que requieren que se coloquen lo más vertical posibles, por lo que, al igual que con los motores, cuando mejor se iba a poder operar con el PBC era cuando este contara con el menor número de componentes sobre él.

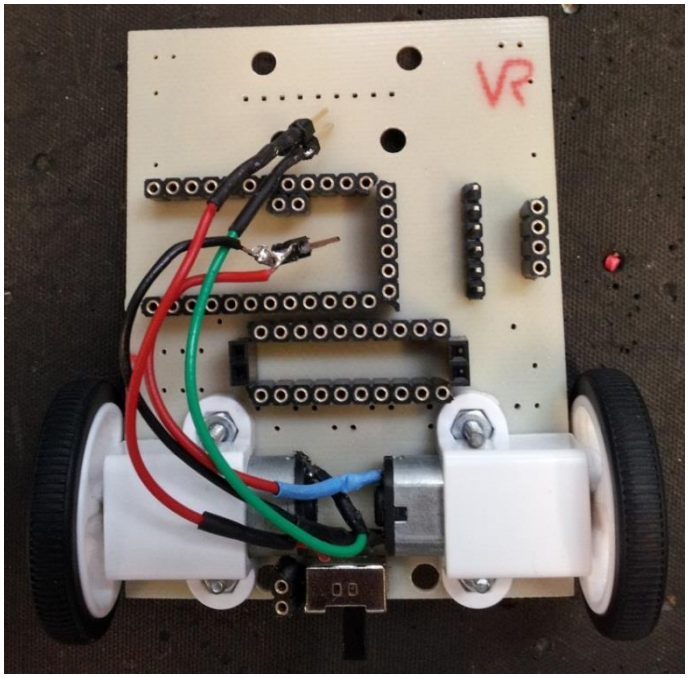


Figura 5-2: Colocación de los conectores hembras y machos

Una vez colocados todos los conectores se soldaron a la placa los diferentes componentes, tales como los LEDs delanteros, los componentes que conforman el circuito del L6205N y el interruptor que controla la alimentación.

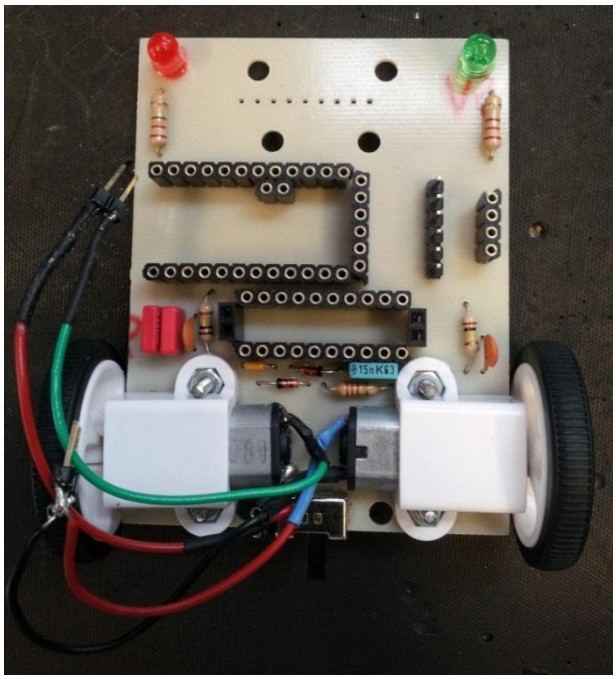


Figura 5-3: Colocación de los LEDs delanteros y los componentes del L6205N

Después se atornilló al PBC la rueda loca. Esta se encuentra muy próxima al QTR-8RC, por lo que se prefirió montar este componente antes que la matriz de sensores infrarrojos.

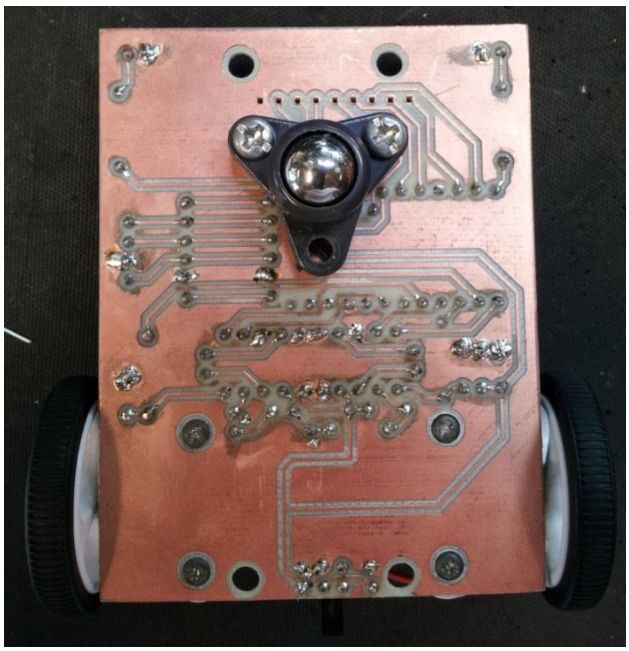


Figura 5-4: Colocación de la bola loca

Tras esto se soldaron unos pines al QTR-8RC y se soldaron al PBC. La distancia al suelo a la que se decidió poner los sensores fue de 4mm, ya que el fabricante recomienda que estos se sitúen a una distancia del suelo de entre 3 y 9mm.

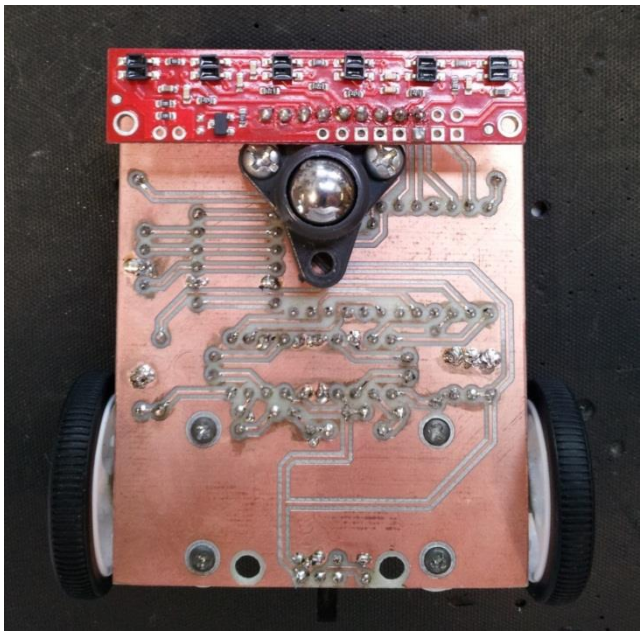


Figura 5-5: Colocación del QTR-8RC

Después se pusieron sobre sus conectores al Arduino Pro Mini y al L6205N.

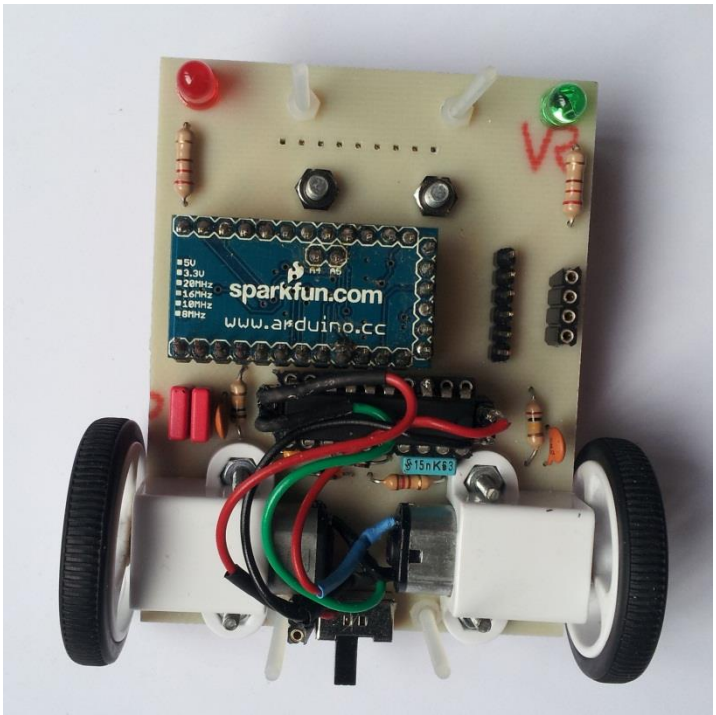


Figura 5-6: Colocación del Arduino Pro Mini y del L6205N

Por último se atornilló en la parte superior del robot una superficie para ofrecer sustento a la pila. La pila se colocará sobre la parte delantera del robot para darle a este más estabilidad.

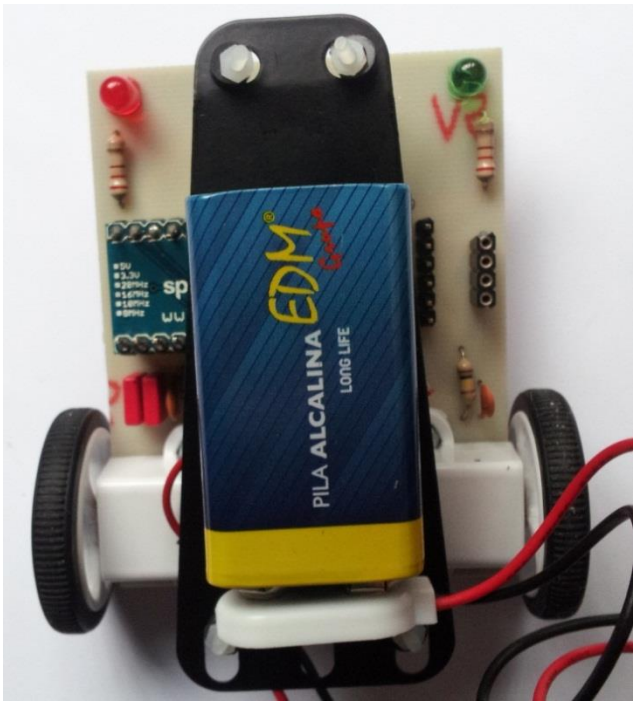


Figura 5-7: Colocación de soporte para pila y pila

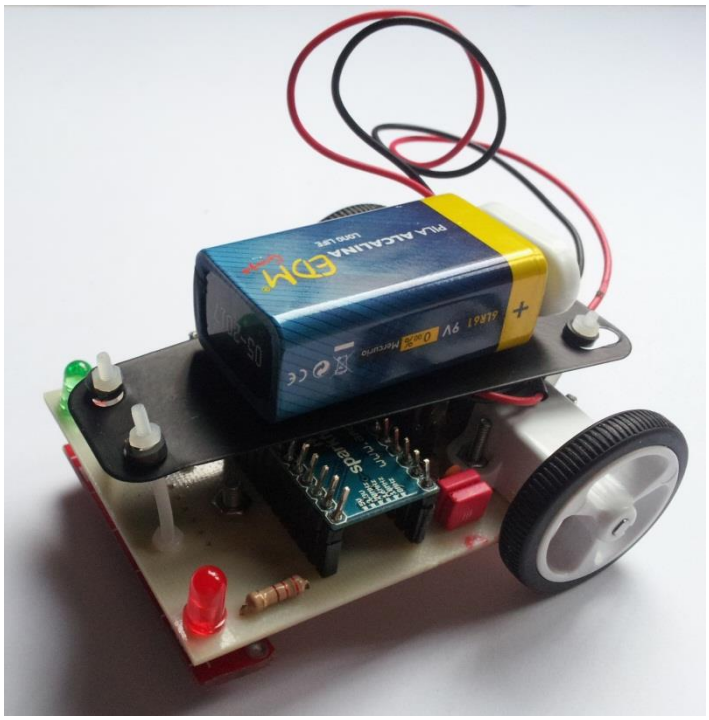


Figura 5-8: Resultado final. Vista isométrica 1

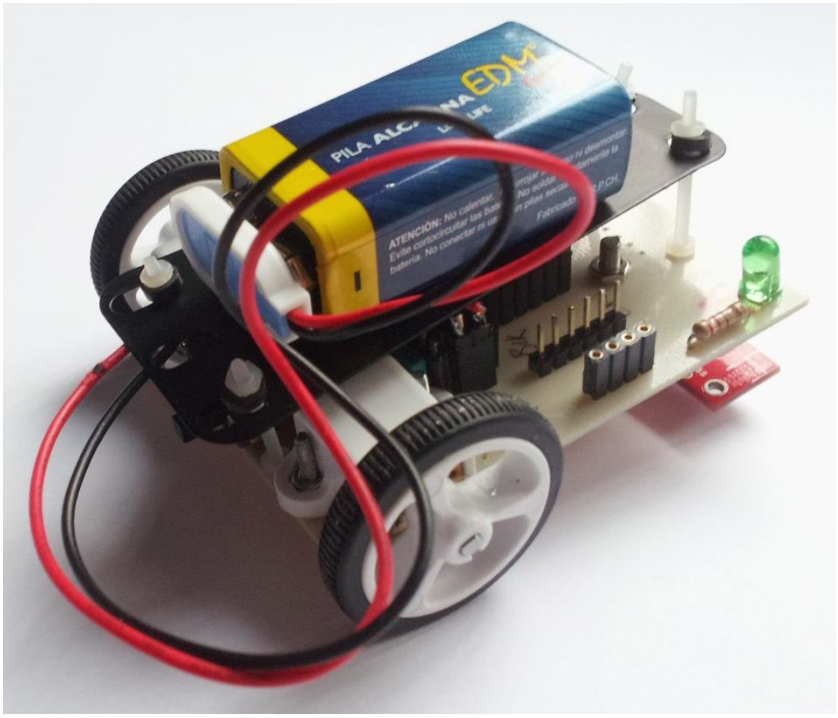


Figura 5-9: Resultado final. Vista isométrica 2

CAPÍTULO 6. PRUEBAS DE LOS COMPONENTES DEL ROBOT

En este apartado se van a desarrollar las diferentes pruebas a las que se sometió al robot para comprobar su perfecto funcionamiento.

Como ya se ha explicado con anterioridad, el Arduino Pro Mini carece de conversor Serie – USB, por lo que hay que emplear un conversor externo para programar el Arduino. Para este proyecto se empleó el conversor FTDI232[32].

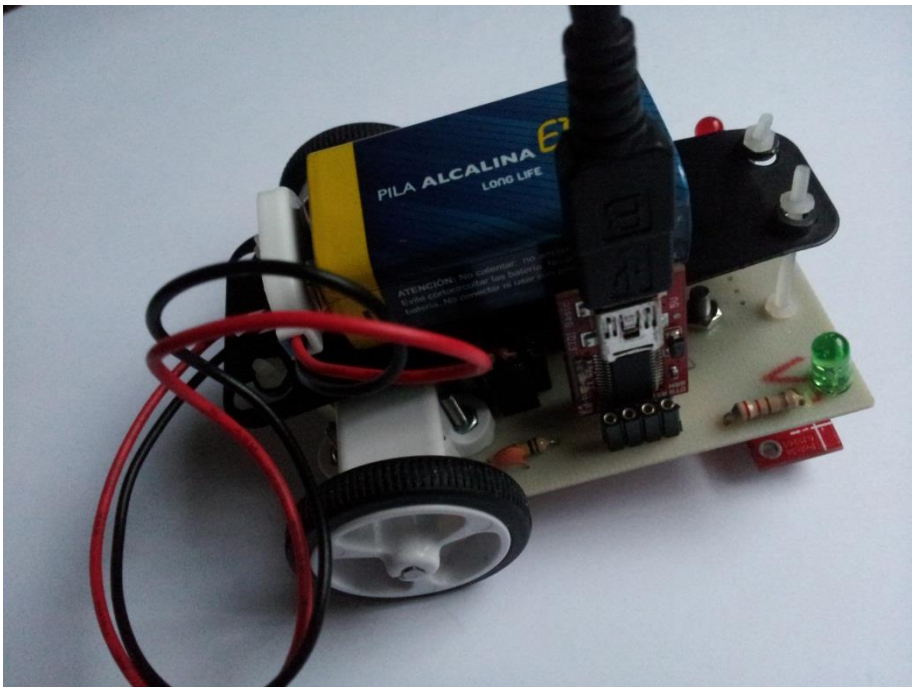


Figura 6-1: Conexión del conversor FTDI232

Las pruebas que se van a realizar son las siguientes:

- Sensores infrarrojos.
- Motores y LEDs.
- Bluetooth.

6.1. SENSORES INFRARROJOS

Al igual que como se hizo en el apartado 3.5.2 se va a probar el funcionamiento de los sensores del QTR-8RC. Para ello se va a emplear el mismo programa que entonces, cambiando únicamente los pines (ANEXO A.5). Si se recuerda el esquema del código:

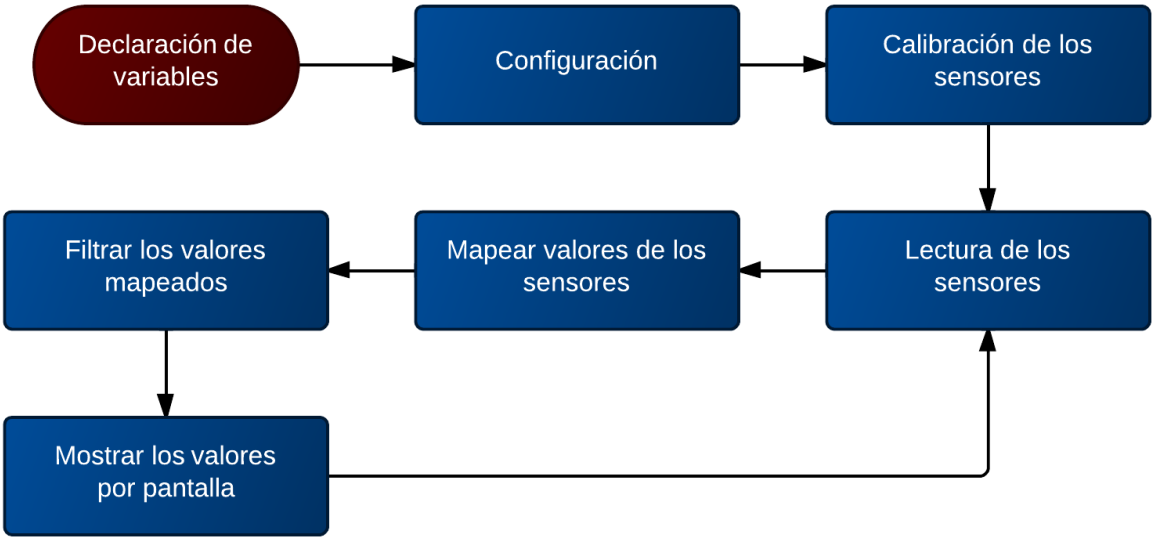


Figura 6-2: Diagrama del *sketch* empleado para la prueba de funcionamiento del QTR-8RC

Para poder mostrar los valores de forma clara, en primer lugar se calibran los sensores para obtener los valores máximos y mínimos de cada sensor para después poder mapearlos. Además se incluye un filtro para evitar valores indeseados. De esta forma se consigue que los sensores que estén justo encima de la línea muestren un valor distinto de cero mientras que los sensores situados fuera de la línea muestran cero.

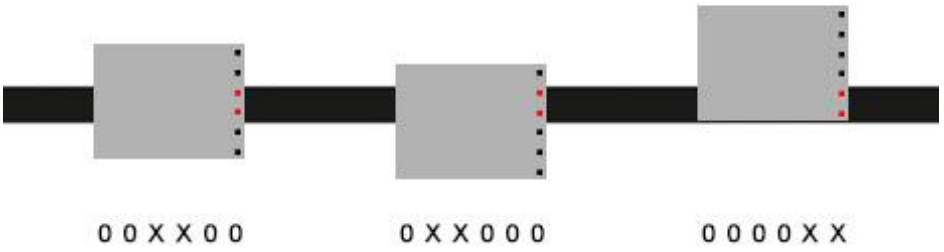


Figura 6-3: Valores obtenidos en función de la posición relativa del robot con respecto a la línea

Al llevar a cabo esta prueba se detectaron dos importantes características geométricas del robot:

- El número mínimo de sensores que pueden detectar la línea negra al mismo tiempo, en recta y sin bifurcaciones, son dos. Esta característica se empleará cuando haya que obtener la posición del robot con respecto a la línea negra.
- El número máximo de sensores que pueden detectar la línea negra al mismo tiempo, en recta y sin bifurcaciones, son tres. Esta característica del robot se empleará para detectar bifurcaciones en el camino.

Como los resultados obtenidos fueron satisfactorios, se pasó a probar los motores.

6.2. MOTORES

Una vez comprobados los sensores infrarrojos se probaron los motores y los LEDs. Conviene recordar del apartado 3.2 que se escogió que el robot empleara un sistema diferencial. Esto quiere decir que para que el robot gire se deben aplicar diferentes velocidades a cada rueda.

Además, como se explicó en el apartado 3.4, para controlar tanto la velocidad de giro como el sentido de giro de las ruedas se va a emplear el circuito integrado L6205N. Este permite que las ruedas giren como uno desee en función de las entradas de control que reciba. Estas entradas de control se las proporciona el Arduino Pro Mini en forma de pulsos PWM. Las señales PWM son señales cuadradas que mantienen constante su periodo pero varían el tiempo que la señal está a nivel alto.

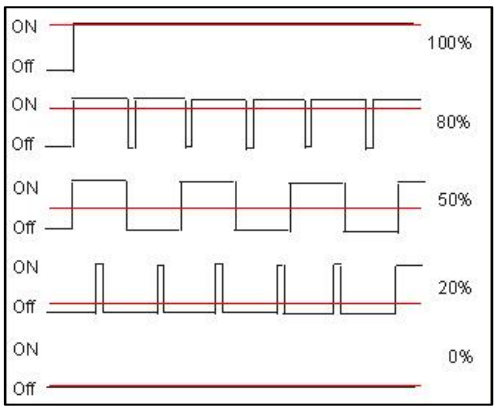


Figura 6-4: Diferentes señales PWM

De esta forma, las salidas PWM del Arduino Pro Mini ofrecen señales de 490Hz y se controlan mediante un byte, es decir, valores de 0 a 255. Así, para frenar el motor se le dará al valor de control un 0, mientras que para hacer que vaya a su máxima velocidad se le dará un 255[10].

Para probar los motores y los LEDs lo que se hizo fue programar que el robot siguiera ciertas trayectorias y emplear los LEDs a modo de intermitentes: si el robot iba recto se encenderían los dos LEDs, si giraba a la derecha que se encendiera el LED derecho y si giraba a la izquierda que se encendiera el LED izquierdo.

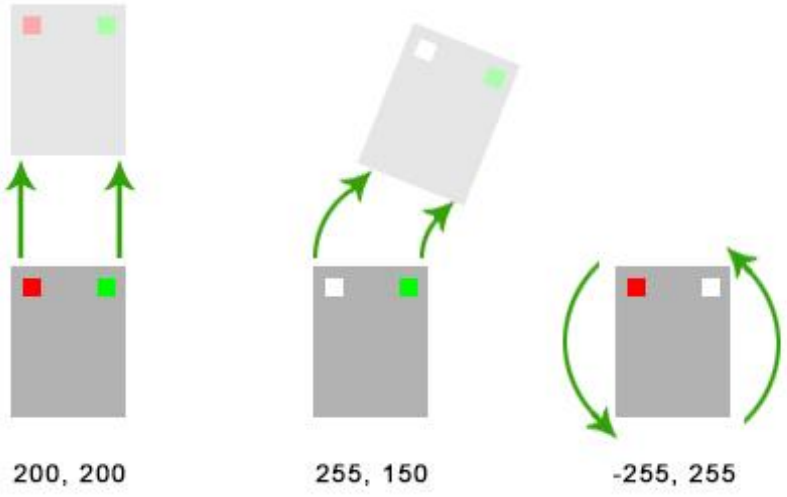


Figura 6-5: Movimientos que puede realizar el robot

De esta forma, se programó que el robot realizara todas las posibles situaciones (ANEXO A.6):

- Marcha adelante.
- Marcha atrás.
- Giro a la izquierda.
- Giro a la derecha.
- Giro en el sitio a la izquierda.
- Giro en el sitio a la derecha.

En esta parte se diseñó la función que mueve los motores. Se le llamó *motor* y cuenta con dos entradas, una para cada motor. La función en sí consiste en analizar cada una de las entradas por separado. Primero se comprueba si el valor es mayor o menor

que cero. Si es mayor o igual que cero se lleva el valor correspondiente a la entrada 1 del puente. Sin embargo, si la entrada es negativa se lleva a la entrada 2 del puente. De esta forma se consigue controlar el sentido de giro de los motores.

Además, como ya se ha explicado, los valores están limitados entre 0 y 255. De esta forma se emplea la función *constrain*, que lo que hace es estudiar si un valor está entre dos límites. Si el valor es mayor que el límite superior, el valor tomará el valor del límite superior. Si el valor es inferior que el límite inferior, el valor tomará el valor del límite inferior.

```
/*
*********************************************************
*      Función para mover los motores según el PWM      *
*********************************************************
*/
void motor(int out1, int out2){
  if (out1 >= 0){
    analogWrite(in1A, constrain(abs(out1), 0, vMax));
    analogWrite(in2A, 0);
  }
  else{
    analogWrite(in1A, 0);
    analogWrite(in2A, constrain(abs(out1), 0, vMax));
  }
  if (out2 >= 0){
    analogWrite(in1B, constrain(abs(out2), 0, vMax));
    analogWrite(in2B, 0);
  }
  else{
    analogWrite(in1B, 0);
    analogWrite(in2B, constrain(abs(out2), 0, vMax));
  }
}
```

Como los resultados fueron los esperados, se pasó a comprobar el módulo bluetooth.

6.3. BLUETOOTH

Por último se comprobó que el bluetooth funcionara de forma correcta. Además se aprovechó esta prueba para estudiar el comportamiento del robot al mezclar diferentes componentes. Así, en primer lugar se comprobó la conectividad por bluetooth entre el robot y el ordenador y después la conectividad por bluetooth entre el robot y un móvil Android.

Señalar que al emplear tanto el módulo bluetooth como el conversor los mismos pines para enviar y recibir datos no podrán estar conectados al mismo tiempo. De esta forma cada vez que se quiera programar el robot habrá que desconectar el módulo bluetooth y una vez cargado el programa en el robot volver a conectar el bluetooth. Esta acción se podría haber facilitado mediante la inclusión de un interruptor que desconectara el módulo bluetooth del interfaz serie durante la programación y así no haría falta separar físicamente el módulo bluetooth del robot para programarlo.

6.3.1. CONEXIÓN ROBOT - ORDENADOR

Desde un primer momento se pensó que una buena utilidad del bluetooth sería la comprobación de diferentes variables del robot en un ordenador. Este era el principal objetivo de esta prueba. De esta forma se pensó que una de las variables más fáciles de analizar sería el estado de los sensores infrarrojos. Para no mostrar solamente números por pantalla se pensó que se podría representar en alguna forma de gráfico el estado de los sensores y qué mejor herramienta para mostrar gráficas que Matlab. Así que se decidió que para comprobar la conexión robot – ordenador se enviaría por bluetooth el estado de los sensores infrarrojos al ordenador y este mediante Matlab representaría el estado de estos en un gráfico de barras.

Figura 6-6: Esquema de comunicación para la prueba del módulo bluetooth

En primer lugar se realizó el *sketch* que ejecutaría el robot. Para evitar que cada vez que se iniciase el código se calibrasen los sensores, se realizaron un par de medidas sobre la pista que posteriormente se utilizaría para observar los resultados para obtener los valores máximos y mínimos de cada sensor.

Después hubo que obtener el tiempo que tardaba Matlab en procesar los datos. Matlab no realiza los cálculos instantáneamente, sino que presenta un pequeño retardo,

y si no se tiene en cuenta no se puede conseguir una sincronización correcta entre el robot y Matlab. Tras sucesivas pruebas, se obtuvo que este tiempo era de 10ms, por lo que al final del *sketch* se introdujo un *delay* de 10ms. El diagrama del *sketch* (ANEXO A.7) es el siguiente:

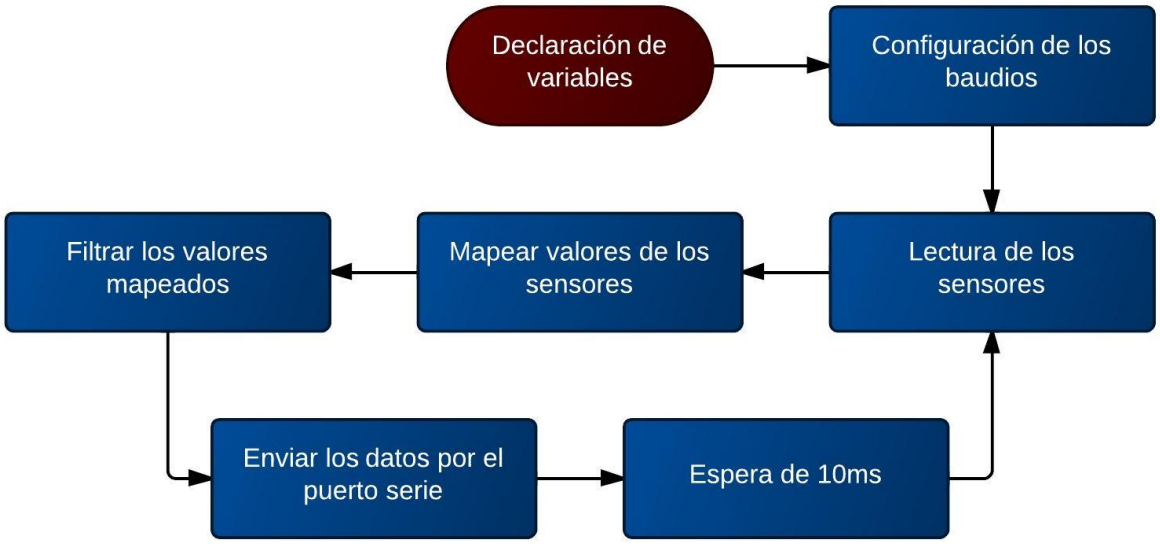


Figura 6-7: Diagrama del *sketch* empleado para la prueba de conexión entre robot y ordenador

Una vez configurado el robot se enviarán los datos por bluetooth, pero simulando un puerto serie que se debe conocer. Para ello se debe vincular el ordenador con el módulo bluetooth. Lo que hay que hacer es que el ordenador detecte el módulo bluetooth y después introducir la contraseña que este tiene asociada, en este caso 4242:

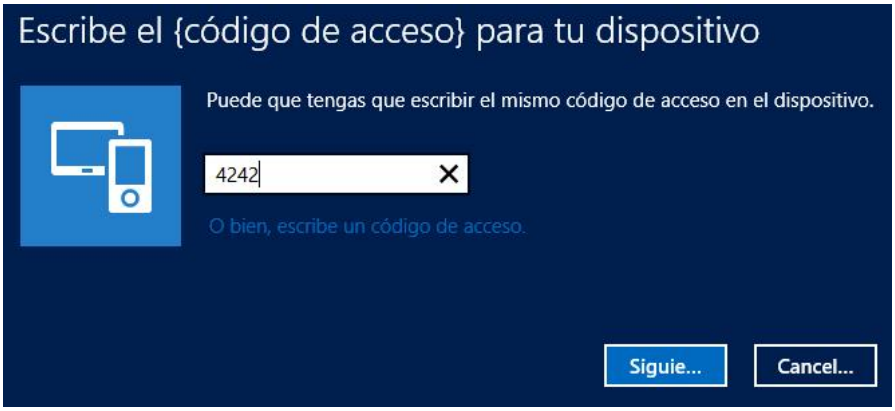


Figura 6-8: Vinculación entre ordenador y módulo bluetooth

Después hay que hacer clic con el secundario en el icono del bluetooth del área de notificación y hacer clic en Abrir configuración. Por último en la pestaña Puertos COM

está el puerto que se debe utilizar. El puerto que se tiene que utilizar es el que en el nombre tenga la extensión 'Dev B', en este caso es el puerto COM6:

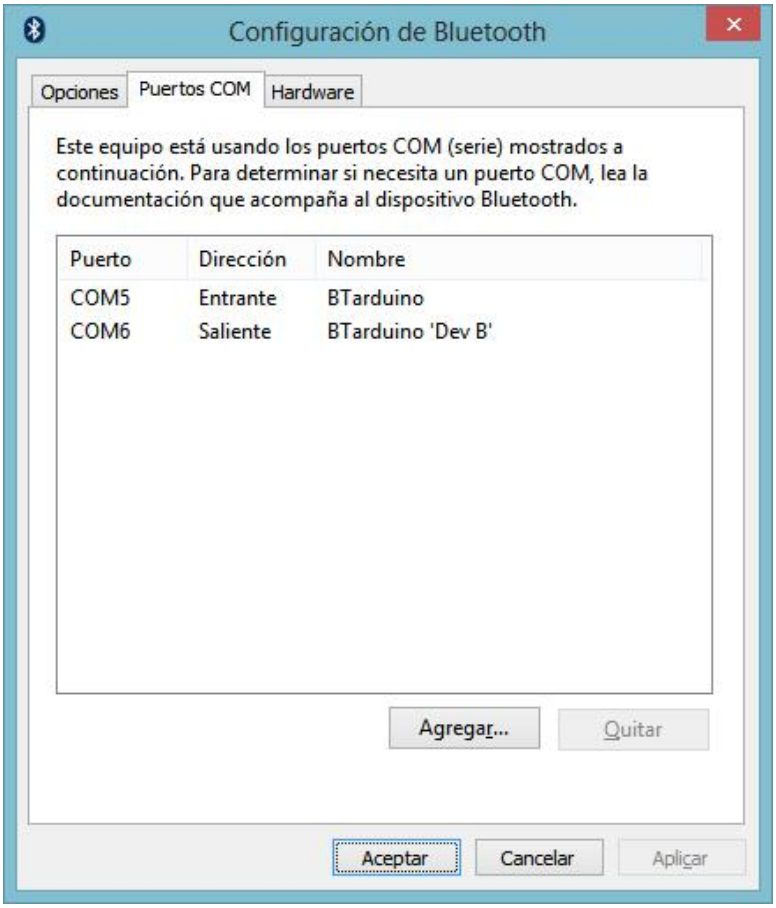


Figura 6-9: Comprobación del puerto serie que se debe emplear

Una vez obtenido el puerto asociado al bluetooth del robot se empieza a realizar el programa de Matlab (ANEXO A.8). En primer lugar se crea un objeto serie en Matlab y se abre para empezar a leer[33]:

```
%borrar previos
delete(instrfind({'Port'},{'COM6'}));
%crear objeto serie
s = serial('COM6','BaudRate',57600,'Terminator','CR/LF');
%abrir puerto
fopen(s);
```

Como se puede ver, la primera línea de código lo que hace es eliminar cualquier aplicación que esté usando el puerto COM6. Después se crea el objeto serie. En este caso se va a denotar con s. En este punto habrá que indicarle los baudios a los que

funciona el módulo bluetooth, en este caso 57600. Por último se abre el puerto con la función *fopen*.

```
%Tiempo que dura la medición en segundos
tmax = 20;
%Inicializar
t = 0;
```

Después hay que definir el tiempo total en el que va a estar funcionando la conexión robot – ordenador. Para esta prueba se pensó que sería suficiente con 20 segundos. Después se inicializa un contador t a cero que servirá para medir el tiempo que lleva ejecutándose el programa.

```
%Ejecutar bucle cronometrado
tic
%Con tic y toc mide el tiempo que hay entre ellos
while t<tmax
    t = toc;
    %Leer del puerto serie
    puerto = fscanf(s,'%d %d %d %d %d %d')';
    %Dibujar el diagrama de barras
    bar(puerto);
    %Actualizar el diagrama
    drawnow
end
```

El núcleo del programa es el bucle de medida, en el cual iremos leyendo del puerto serie los datos en el formato que hemos especificado en la función *fscanf*. En este caso, desde el robot se está enviando el estado de los seis sensores de forma consecutiva separando cada valor mediante un espacio. Después se emplea la función *bar* para dibujar el diagrama de barras en función de los valores de los sensores y con la función *drawnow* se muestra por pantalla el diagrama de barras. Si no se incluyera solo se mostraría el último resultado. Aclarar también el funcionamiento de *tic* y *toc*. Son dos variables internas que usa Matlab. Con *tic* se empieza a ejecutar un reloj interno y con *toc* se puede acceder al valor de este reloj. De esta forma, si se le va dando en cada ciclo a t el valor de *toc* se podrá conocer cuándo se ha alcanzado el tiempo máximo.

```
clc;
%% Cerrar y eliminar el puerto creado
fclose(s);
delete(s);
clear s;
```

Por último, si el resultado ha sido satisfactorio, se limpiará la pantalla principal de Matlab y se cerrará y eliminará el puerto creado. Así, se obtiene la siguiente figura:

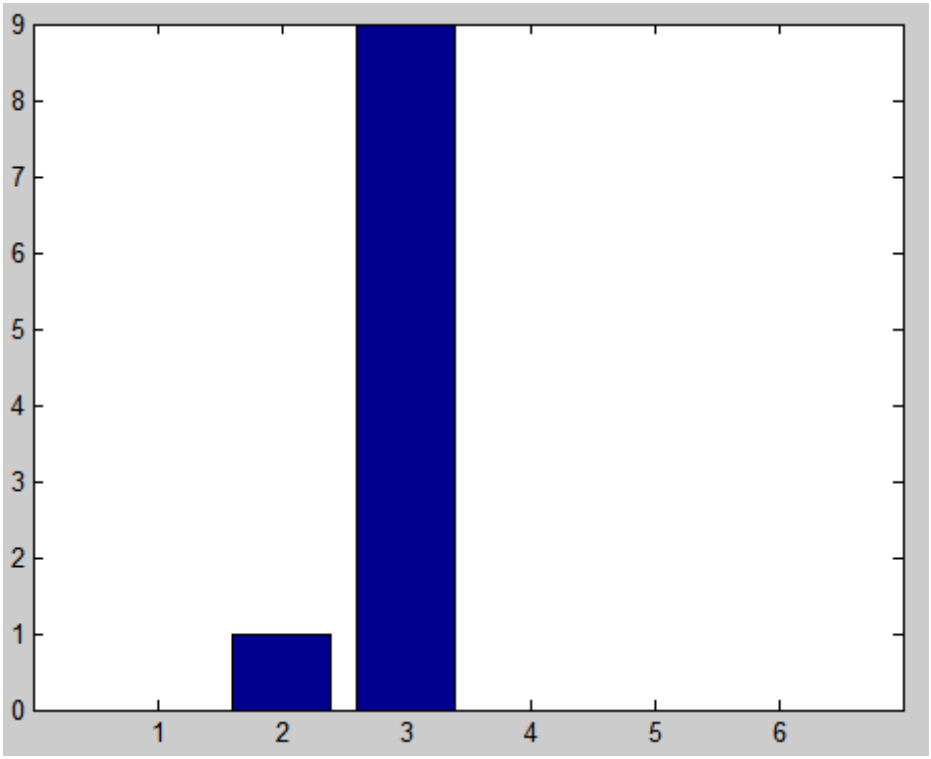


Figura 6-10: Resultado de la prueba

En el gráfico en el eje de abscisas se representa cada sensor infrarrojo por separado y en el eje de ordenadas el valor en ese instante de cada sensor. Si el sensor está situado sobre la línea negra mandará un 9, mientras que si no lo está mandará un 0. Si el sensor no está totalmente situado sobre la línea negra se enviará un dato intermedio. En este caso, el sensor 3 se encontraba totalmente situados sobre la línea negra mientras que el sensor 2 solo estaba un poco situado sobre esta. Los demás sensores ninguno estaba sobre la línea negra.

Como los resultados obtenidos fueron satisfactorios, se pasó a realizar la última prueba al robot antes de comenzar con la programación del sigue líneas.

6.3.2. CONEXIÓN ROBOT – MÓVIL ANDROID

Para realizar esta prueba se empleó la misma técnica que en el apartado 3.6.2. Mediante la aplicación disponible en el *Play Store* de Android llamada *BlueTerm* se enviarán datos al robot y este según los valores que obtenga ejecutará una u otra acción. El diagrama del *sketch* (ANEXO A.9) es el siguiente:

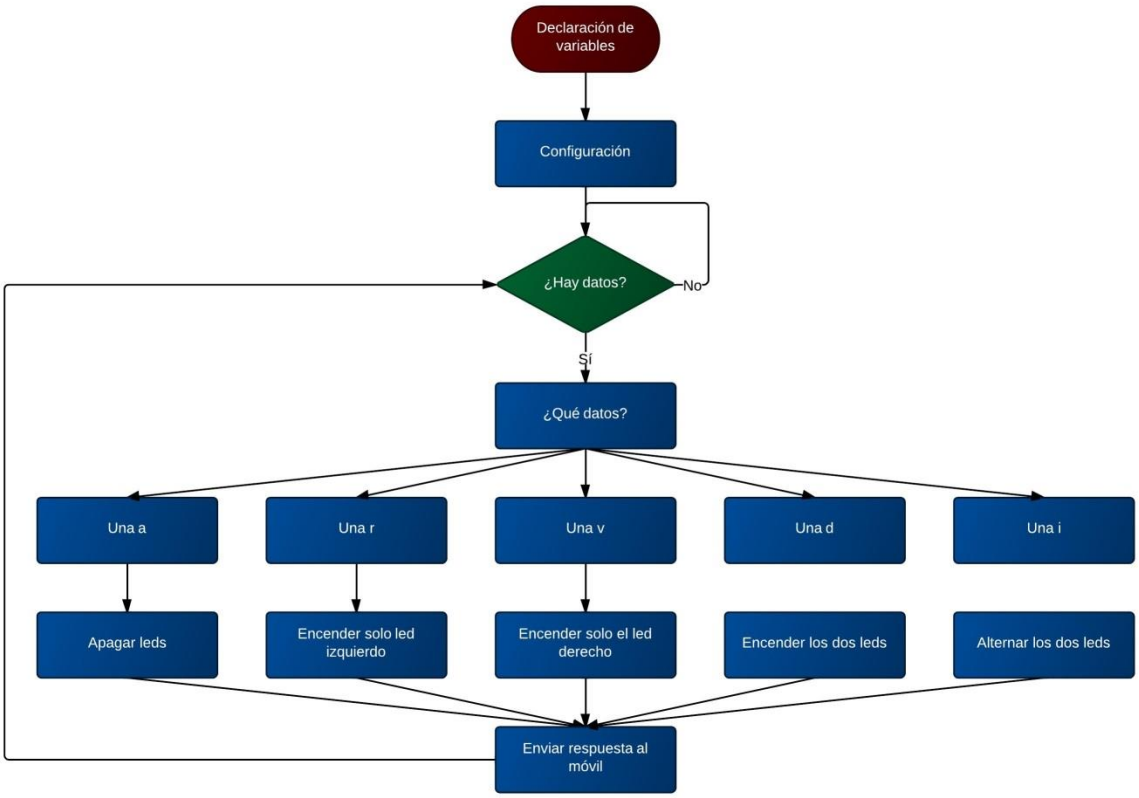


Figura 6-11: Diagrama del *sketch* empleado para la prueba de funcionamiento de la conexión entre robot y Android

Como los resultados obtenidos fueron satisfactorios, se concluyó que todas las partes del robot funcionaban correctamente y se empezó a programar el sigue líneas.

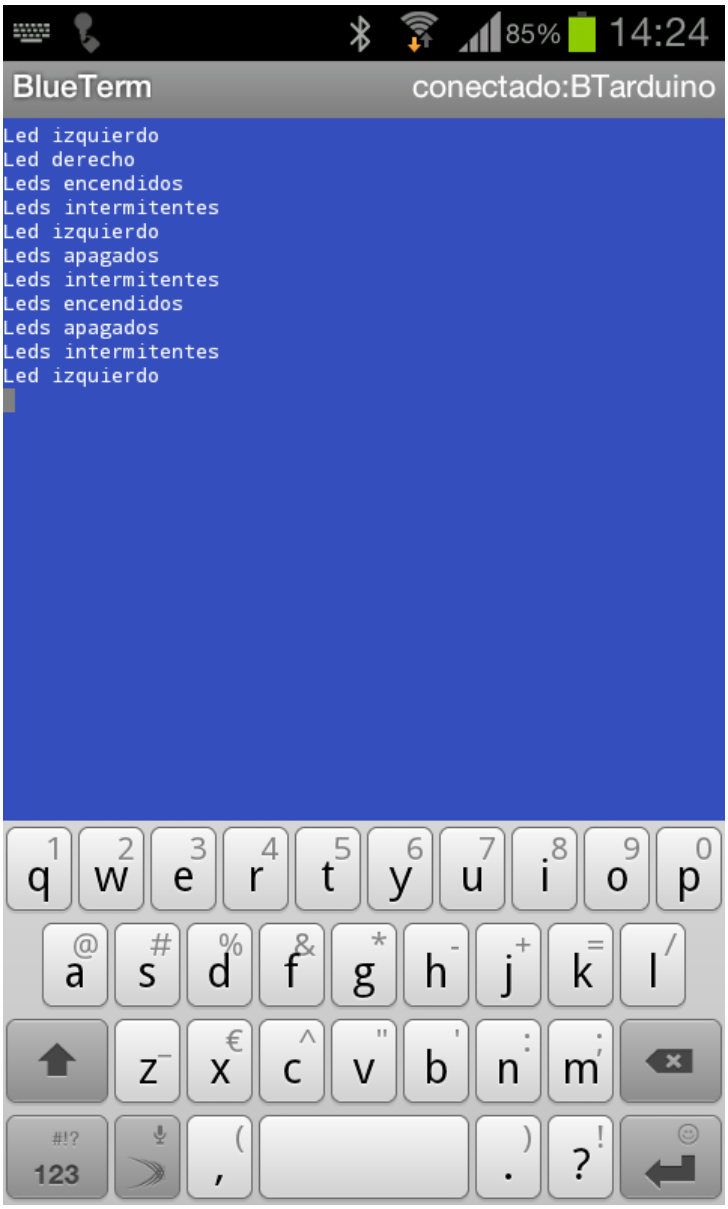


Figura 6-12: Resultado obtenido en el dispositivo Android

CAPÍTULO 7. SIGUE LÍNEAS

En esta parte del proyecto se va a desarrollar el funcionamiento del robot como un seguidor de líneas. A continuación se muestra el diagrama de todo el *sketch* (ANEXO A.10).

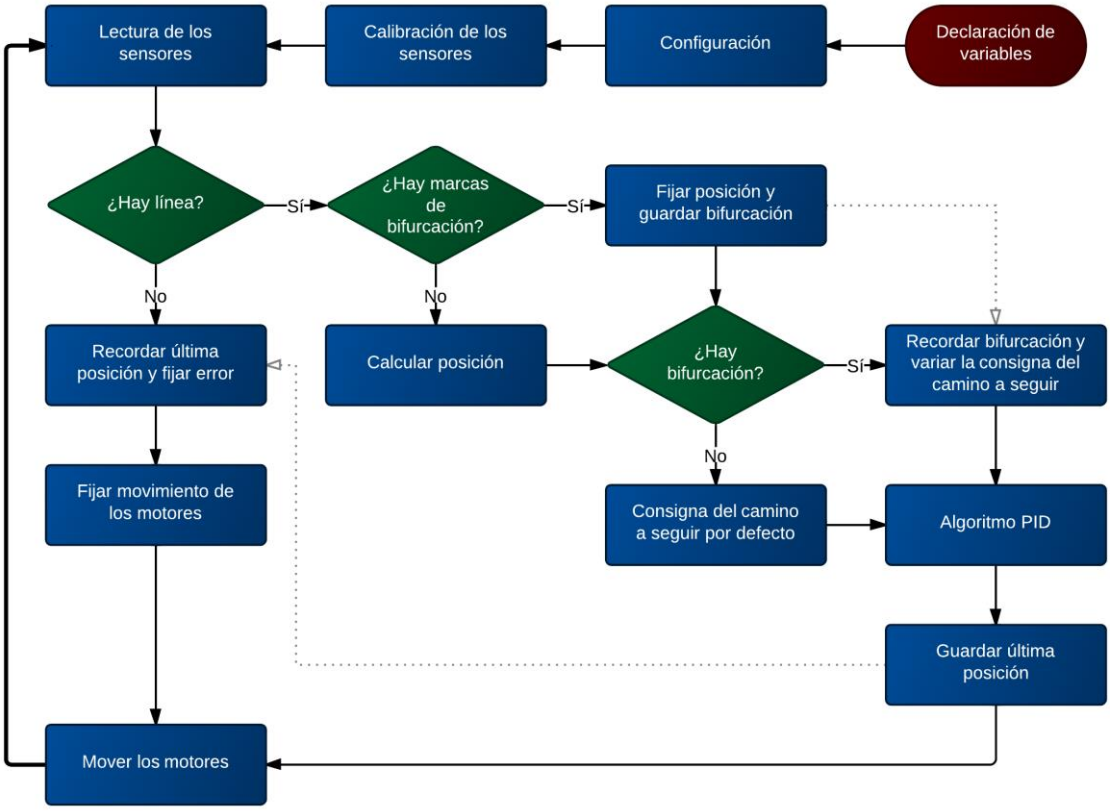


Figura 7-1: Diagrama del *sketch* empleado por el robot para seguir líneas

En él se pueden distinguir diferentes partes.

- Declaración de variables.
- Configuración del robot.
- Calibración de los sensores.
- Seguimiento de la línea.
- Algoritmo PID.
- Salida del trazado.

En los siguientes apartados se procederá a desarrollar cada una de estas partes.

7.1. DECLARACIÓN DE VARIABLES

En primer lugar se deben declarar las variables que empleará el robot. Para poder manipularlas más adelante con facilidad se agrupan en conjuntos según cuál sea su finalidad.

7.1.1. ASIGNACIÓN DE PINES

En este conjunto de variables se definen los pines del Arduino Pro Mini que le corresponden a las diferentes partes del robot. De esta forma no hace falta ir recordando a que pin le corresponde por ejemplo el LED delantero derecho, sino que con una variable podemos manipular el LED y olvidarnos de qué pin del Arduino Pro Mini le corresponde.

En este tipo de variables también están las entradas y los enables que manipulan el L6205N, los pines a los que están conectados los sensores infrarrojos y los ya mencionados LEDs delanteros.

7.1.2. VARIABLES DE LOS SENSORES INFRARROJOS

Estas variables tienen como finalidad la de manipular los datos que se reciben de los sensores infrarrojos. De esta forma, todas las variables de este tipo son de tipo vector, ya que se necesita manipular la información de seis sensores.

En este tipo de variables está las variables donde se guardan las lecturas de los sensores, donde se guardan los pesos de los sensores (que más adelante se emplearán para calcular la posición), las variables para poder calibrar los sensores y las variables para mostrar los resultados de los sensores de una forma más clara.

7.1.3. PARÁMETROS DEL PID

Aquí se definen las variables que se emplearán para llevar a cabo el control PID. Como se explicará en el apartado 7.5, aquí se definen las variables de la parte

proporcional, de la parte integral y de la parte derivativa del control, además de otras variables para poder realizar todos los cálculos que se necesitan.

7.1.4. VARIABLES PARA EL CÁLCULO DE LA POSICIÓN

Como su nombre indica, la finalidad de estas variables son las de poder obtener en qué posición se encuentra el robot en relación a la línea negra. Así, se necesita una variable que defina el camino que debe seguir el robot, otra que almacene su posición y de otras variables que se emplearán cuando el robot se encuentre ante una bifurcación o cuando se haya salido del trazado.

7.1.5. VARIABLES PARA EL MOVIMIENTO DE LOS MOTORES

Con estas variables se controla el movimiento de los motores. Se necesitan dos variables para definir la velocidad de los motores (una por cada motor), además de otra para definir la velocidad máxima que pueden alcanzar y una última variable que es la que define la velocidad del robot en línea recta, es decir, cuando el robot se encuentra totalmente centrado en la línea negra.

En un primer momento se puede pensar en poner esta velocidad igual a la máxima velocidad, pero si se recuerda el apartado 6.2 los motores tienen limitadas sus velocidades entre 0 y 255. De esta forma, si se ajustara que el robot fuera a la máxima velocidad en recta y se encontrara una curva su capacidad de maniobrar se verá drásticamente reducida. Por lo que hay que elegir una velocidad que vaya los suficientemente rápido en recta y que permita al robot funcionar perfectamente en las curvas.

7.1.6. OTRAS VARIABLES

Aquí se engloban al resto de variables, tales como las de tiempo o las que controlan el brillo de los LEDs.

7.2. CONFIGURACIÓN DEL ROBOT

Una vez definidas todas las variables del robot se procede a su configuración. En primer lugar se define la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para ello se emplea la función *Serial.begin*. Debido a que el módulo bluetooth se ha configurado a una velocidad de 57600 baudios, esta será también la velocidad que empleará el robot.

Después se definen cómo van a funcionar los pines del Arduino Pro Mini con la función *pinMode*. Un pin puede funcionar como pin de salida o de entrada. Por defecto todos los pines están configurados como pines de entrada, por lo que solo hay que configurar los pines de salida, aunque siempre es recomendable configurar también los pines de entrada para que quede un código más claro. En el caso del robot, como se ha empleado la estructura *QTRSensorsRC* incluida en la librería *QTRSensors.h* ya se define dentro de esa estructura los pines correspondientes como pines de entrada. Por otro lado, se definen como pines de salida los pines que controlan el L6205N y los pines que controlan los LEDs delanteros.

Por último se prepara al robot para entrar en fase de calibración. Para ello en primer lugar se ponen todas las salidas que controlan al L6205N a nivel bajo para asegurar que el robot empieza en posición de reposo. Después se activan ambos puentes en H del L6205N poniendo a nivel alto los enables de ambos puentes. También se apagan los LEDs para reducir el consumo. De esta forma el robot ya se puede empezar con la calibración de los sensores infrarrojos con la función *calibracion*.

7.3. CALIBRACIÓN DE LOS SENSORES INFRARROJOS

En realidad la calibración de los sensores infrarrojos forma parte de la configuración del robot, pero debido a su importancia se ha pensado en dedicarle un apartado solo a él para analizarlo más en profundidad.

Las razones por las que se pensó en introducir una fase de calibración fueron dos:

- Todos los sensores no presentan las mismas lecturas ante una misma situación. De esta forma, al realizar una calibración inicial se pueden detectar cuáles son

las máximas y las mínimas medidas que realiza cada sensor y después ponderar las medidas en función de estos valores. De otra forma lo que para un sensor puede significar estar sobre la línea negra para otro simplemente puede significar estar parcialmente sobre la línea negra.

- Se puede emplear el robot para que siga la línea negra sobre otras superficies que no sean necesariamente blancas. Cuando se empezó a realizar pruebas con el robot era bastante laborioso tener que estar usando solo superficies blancas. De esta forma se puede emplear cualquier color, evitando evidentemente el negro. Así, simplemente utilizando cinta aislante negra y el parqué de tu casa uno se puede construir su propio trazado.

A continuación se muestra el diagrama de la calibración de los sensores infrarrojos del robot:

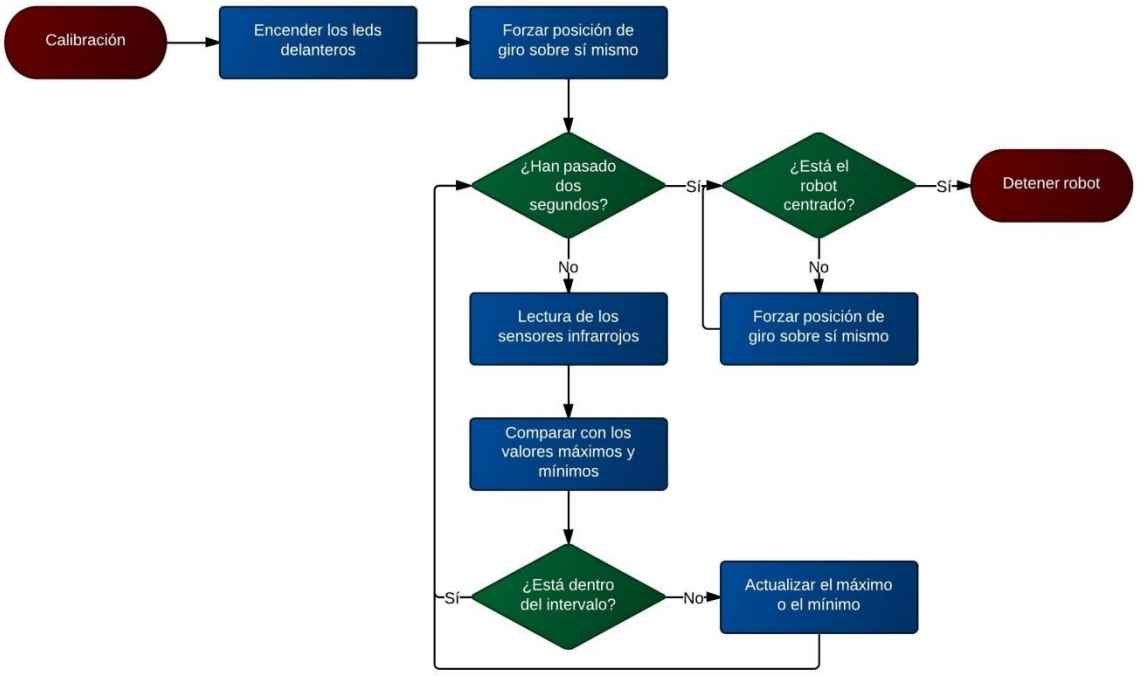


Figura 7-2: Diagrama del *sketch* para calibrar los sensores infrarrojos

Una vez entrado en la fase de calibración, lo primero que se hace es encender los dos LEDs delanteros para indicar que el robot está en la fase de calibración. Después se fuerza al robot a girar sobre sí mismo a una determinada velocidad y durante dos segundos los sensores realizan mediciones. Estas mediciones se comparan con las máximas y mínimas medidas que ha presentado cada sensor hasta ese momento con la

función *minmax*. Si la medida es mayor al máximo, la medida pasará a ser el nuevo máximo. Si la medida es menor que el mínimo, la medida pasará a ser el nuevo mínimo.

Tras haber calibrado los sensores el propio robot, de forma autónoma, se sitúa de forma correcta sobre la línea. Para ello de nuevo se le hace girar al robot a una determinada velocidad hasta el robot se encuentre centrado sobre la línea. El cómo se calcula la posición relativa del robot con respecto a la línea se explica más adelante.

Una vez que el robot se ha situado correctamente sobre la línea se apagan los LEDs delanteros y se da por concluida la calibración de los sensores infrarrojos.

7.4. SEGUIMIENTO DE LA LÍNEA

Tras la calibración de los sensores infrarrojos empieza el bucle del programa. Lo primero que se realiza es la medida de los sensores infrarrojos. Para ello se emplea la función *qtrrc.read* de la librería *QTRSensor*[34].

Tras la lectura de los sensores se realiza un mapeado de los valores entre 0 y 9 en función de los valores máximos y mínimos que se han obtenido en la fase de calibración. De esta forma se consigue que los valores que ofrezca cada sensor sea independientemente del sensor que sea. Se ve más claro en la siguiente imagen:

Figura 7-3: Diferencia entre los valores devueltos por los sensores infrarrojos con y sin mapear

Al mapear los valores de los sensores se consigue una medida más fidedigna con la realidad. Sin mapear es posible que un sensor que está sobre la línea negra ofrezca una medida menor a la que ofrece un sensor que no está sobre la línea negra.

Tras mapear los sensores se filtran estos valores con la función *filtro*. Es posible que por el motivo que sea los sensores ofrezcan una medida que se salga de los límites establecidos. Este filtro evita que eso suceda. Además, este filtro permite contabilizar cuántos sensores están sobre la línea. Si resulta que no hay ningún sensor sobre la línea negra es que el robot se ha salido del trazado. Este caso se tratará en el apartado 7.6. Por otro lado, si al menos un sensor ve línea se ejecuta la función *si_linea*.

Con esta función lo que se pretende es barajar las opciones posibles existentes si el robot está sobre la línea. Lo primero que hace esta función es detectar si hay marcas de bifurcaciones en los laterales del trazado. Si se recuerda del apartado 2.1.2, se marca en los laterales del trazado con pequeñas marcas la dirección que deberá tomar el robot en la siguiente bifurcación.

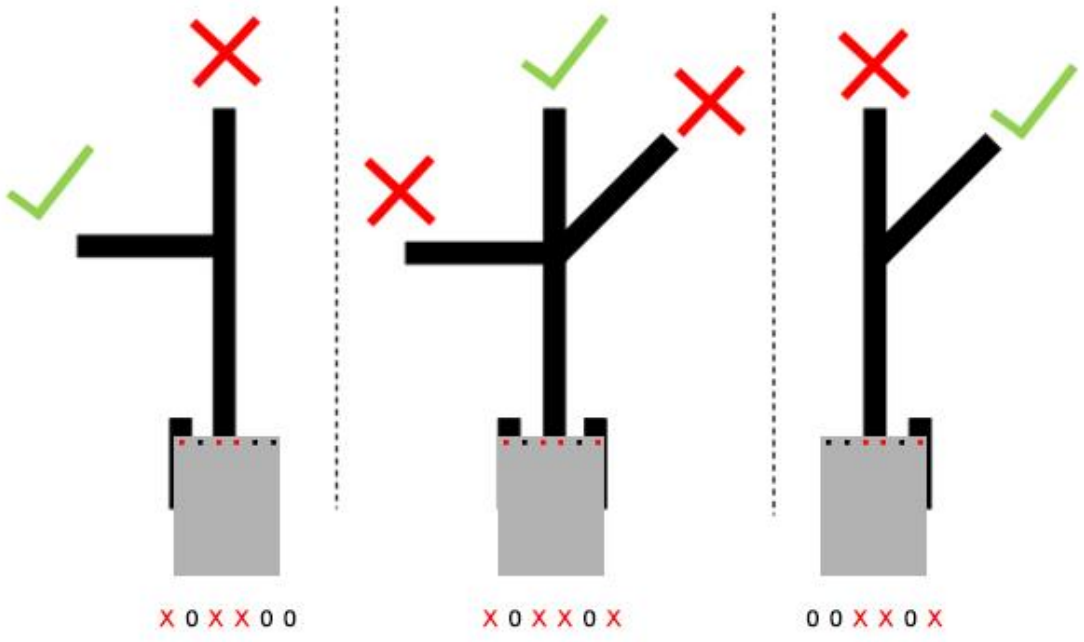


Figura 7-4: Detección de las marcas de bifurcación

Así, lo que hace esta función es detectar si el robot se encuentra en alguna de estas posiciones. Si se encuentra en alguna de ellas se ejecuta la función *si_marca*, la cual guarda qué marca era y obliga al robot a seguir recto. Por otro lado, si el robot no se encuentra en ninguna de esas posiciones ejecuta la función *calculo_posicion*.

Esta función es la que permite determinar la posición del robot en relación a la línea negra. Para ello aplica la siguiente ecuación:

$$Posición = \frac{\sum_{i=0}^{5}(valor\ sensor)_i * (peso\ sensor)_i}{\sum_{i=0}^{5}(valor\ sensor)_i}$$

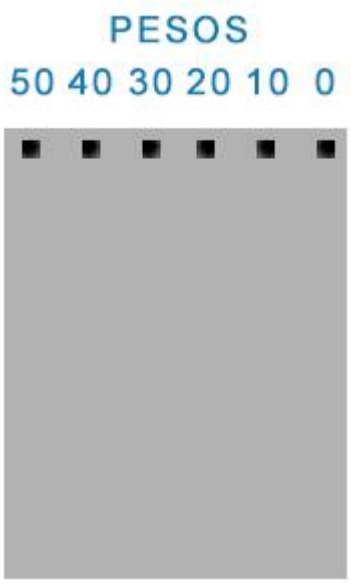


Figura 7-5: Valores de los pesos que tiene cada sensor

En ella se tiene en cuenta tanto el valor de cada sensor como el peso que tiene asociado cada sensor. De esta forma, el valor que tiene que seguir el robot para ir completamente recto es 25.

Tras obtener la posición relativa del robot con respecto a la línea, se estudia si el robot está en una bifurcación. Como se explicó en el apartado 6.1, el número máximo de sensores que pueden detectar la línea negra al mismo tiempo es tres. Por lo tanto, en el momento en el que más de tres sensores detecten línea es que hay presente una bifurcación. De esta forma, para obligar al robot a tomar una cierta dirección se cambia el valor que tiene que seguir el robot a 0, a 25 o a 50 en función de si tiene que ir a la izquierda, seguir recto o ir a la derecha. Así se consigue que el robot sitúe la mayoría de sus sensores sobre la dirección que debe tomar y cuando vuelva a seguir el camino de forma normal (consigna de 25) ya estará sobre el camino a seguir.

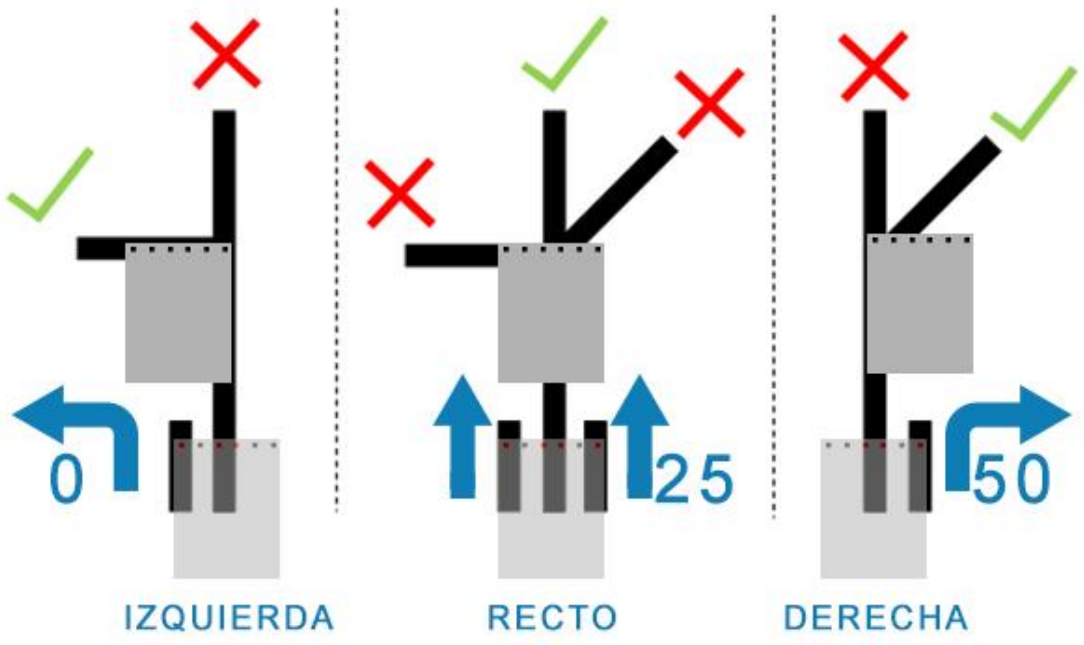


Figura 7-6: Obligación de giro del robot según las marcas de bifurcación

Tras detectar las posibles bifurcaciones se calcula el error. Para calcularlo se aplica la siguiente ecuación:

$$Error = (posición\ a\ seguir) - (posición\ actual)$$

Además se almacena el último error que ha tenido el robot por si este se sale de la trayectoria y de esta forma poder volver a la línea.

Una vez repasadas todas las posibles configuraciones que pueden aparecer si el robot está sobre la línea se aplica el control PID. Debido a su importancia se explicará en el siguiente apartado.

7.5. PID

Cuando se estudiaron los diferentes robots sigue líneas ya existentes prácticamente la totalidad de ellos empleaban un control PID para corregir la posición del robot y hacer que este siga la trayectoria de forma correcta. Pero ninguno de ellos contaba con ninguna herramienta para obtener los valores óptimos de las constantes que forman este control, sino que simplemente iban probando diferentes valores hasta que consideraban que el robot seguía lo suficientemente bien la línea.

En los siguientes apartados se hará una breve introducción sobre el control PID, después se describirá una herramienta para obtener valores óptimos de las constantes y por último se aplicará esta herramienta en el robot.

7.5.1. ¿QUÉ ES UN CONTROLADOR PID?

Un controlador PID (proporcional, integral, derivativo) es un mecanismo de control retroalimentado en bucle cerrado ampliamente utilizado en sistemas de control industrial[8]. Un controlador PID calcula el error como la diferencia entre el valor actual del sistema y un valor de referencia objetivo.

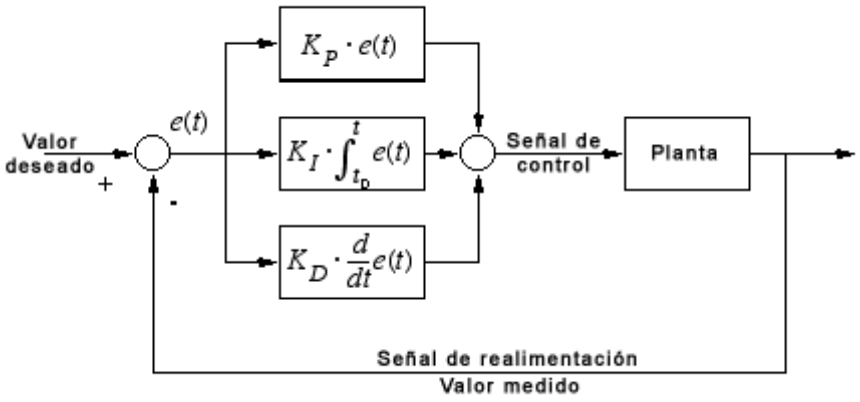


Figura 7-7: Diagrama de bloques de un controlador PID

En el cálculo del controlador PID intervienen tres parámetros, los cuales pueden ser interpretados en función del tiempo. La parte proporcional P depende del error actual, la parte integral I depende de la suma de todos los errores pasados y la parte derivativa D es la predicción de errores futuros, basándose en la tasa de cambio actual. La suma ponderada de los tres términos permiten ajustar un proceso mediante un elemento de control como por ejemplo la velocidad que debe alcanzar un motor de corriente continua.

$$U = P + I + D$$

Mediante el ajuste de estas tres constantes del algoritmo de control PID, el controlador es capaz de proporcionar acciones de control específicas a los

requerimientos de un sistema. La efectividad del controlador se puede medir con tres parámetros. El ajuste estos parámetros se logra mediante el análisis del sistema.

- **Tiempo de pico tp**: es el tiempo que tarda la señal de salida del sistema en alcanzar el pico máximo de sobreoscilación.
- **Pico de sobreoscilación Mp**: expresa cuánto se eleva la evolución temporal de la señal de salida del sistema respecto al valor final. Es por lo tanto la amplitud del primer pico y se suele medir en porcentaje respecto al valor final.
- **Tiempo de estabilización ts**: tiempo que tarda la señal de salida del sistema en entrar en la banda del 5% alrededor del valor final y ya no vuelve a salir de ella.

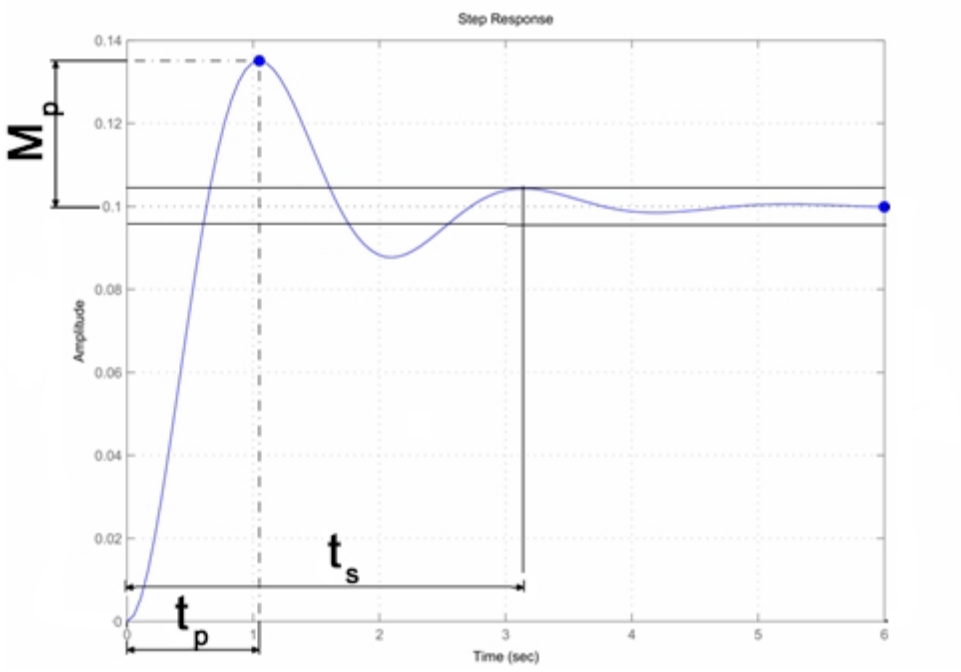


Figura 7-8: Representación del tiempo de pico, del pico de sobreoscilación y del tiempo de estabilización

Estos parámetros están intrínsecamente relacionados unos con otros. Un ejemplo de esto es que, por ejemplo, si se desea una respuesta muy rápida esto provocará que el sistema presente oscilaciones. En un ejemplo práctico esto es que un motor alcance una determinada posición y que debido a la inercia se pase de la referencia y oscile entorno a ella hasta llegar a la posición determinada por la referencia. Es claramente un efecto negativo ya que por intentar disminuir el tiempo de estabilización se ha provocado una sobreoscilación y como consecuencia se ha ralentizado la respuesta.

Es necesario aclarar que el uso de un controlador PID no garantiza un control óptimo ni estabiliza todos los sistemas. Hay sistemas, como los de fase no mínima, que no se estabilizan con el mero hecho de añadir un controlador PID en ese sistema.

Algunas aplicaciones sólo requieren el uso de uno o dos parámetros para proporcionar un control apropiado al sistema. Para no usar un parámetro simplemente se debe poner la ganancia de ese parámetro a cero. Dependiendo de los parámetros activos, los controladores se llaman PI, PD, P o I. Los controladores PI son muy comunes ya que la acción derivativa D es sensible al ruido. Por otro lado, la ausencia de la acción integral puede evitar que el sistema llegue al valor deseado.

7.5.1.1. TÉRMINO PROPORCIONAL

El término proporcional modifica la salida proporcionalmente con el error actual. La respuesta proporcional se puede ajustar multiplicando el error por una constante K_p, conocida como ganancia proporcional.

$$P = K_P e_{(t)}$$

Siendo e el error en un instante de tiempo, cuanto mayor sea este error más rápida será la respuesta y viceversa.

Un controlador proporcional no siempre alcanzará su valor objetivo, ya que no se eliminará el error de estado estacionario. Para eliminarlo hay que emplear el término integral.

7.5.1.2. TÉRMINO INTEGRAL

El término integral es proporcional a la magnitud del error y a la duración del error. Es decir, la suma de todos los errores en cada instante de tiempo o, como indica su nombre, la integración de todos los errores. Esta suma compensa la diferencia que debería haber sido corregida anteriormente. El error acumulado se multiplica por la ganancia integral K_i que indicará la cantidad de acción integral respecto de toda la acción de control.

$$I = K_i \int_0^t e_{(\tau)} d\tau$$

Al usar la integral junto al proporcional se acelera el movimiento del sistema llegando antes al valor deseado, es decir, menor tiempo de respuesta. Además elimina el error estacionario que se produce al usar un controlador proporcional. Sin embargo el término integral tiene en cuenta los errores acumulados en el pasado y puede hacer que el valor actual sobrepase el valor deseado, lo cual creará un error en el otro sentido, creando oscilaciones alrededor del valor deseado. Para evitarlo se emplea el término derivativo.

7.5.1.3. TÉRMINO DERIVATIVO

El término derivativo calcula la variación del error mediante la pendiente del error en cada instante de tiempo, es decir, la primera derivada con respecto al tiempo y multiplica esa variación del error con la ganancia derivativa K_d, la cual indica la cantidad de acción derivativa respecto de toda la acción de control.

$$D = K_d \frac{d}{dt} e_{(t)}$$

El término derivativo ralentiza la tasa de cambio de la salida del controlador y cuánto más cerca está del valor deseado, se hace aún más lento. Por lo tanto, el uso del término derivativo sirve para disminuir las oscilaciones producidas por el término integral y mejorar así la estabilidad del proceso. Uno de los inconvenientes de emplear el término derivativo es que amplifica el ruido, por lo que se puede inestabilizar el sistema con más facilidad.

7.5.1.4. ECUACIÓN DEL CONTROLADOR DE FORMA DISCRETA

Los términos proporcional, integral y derivativo son sumados para hallar la salida del controlador PID, sabiendo que u(t) es la salida, la ecuación que define al algoritmo es la siguiente:

$$u_{(t)} = K_p e_{(t)} + K_i \int_0^t e_{(\tau)} d\tau + K_d \frac{d}{dt} e_{(t)}$$

Sin embargo un microcontrolador solo es capaz de procesar valores discretos y finitos, así que para poder implementar el algoritmo de controlador PID es necesario discretizar la ecuación del controlador PID. La ecuación discreta del controlador para poder ser implementada en un microcontrolador se puede expresar mediante la siguiente ecuación:

$$u_{(t)} = K_p e_{(t)} + K_i T_s \sum_{k=0}^{t} e_k + K_d \frac{e_{(t)} - e_{(t-1)}}{T_s}$$

Donde e(t) es el error de la respuesta del sistema en el instante t, T es el periodo de muestreo de la señal y K_p, K_i y K_d son la ganancia proporcional, integral y derivativa del controlador, respectivamente.

A continuación se muestra en una tabla los efectos que provocan en el sistema si se aumenta cada parámetro por separado y se dejan los demás constantes:

Parámetro	Tiempo de pico	Sobreoscilación	Tiempo de estabilización	Error en estado estacionario	Estabilidad
Kp	Aumenta	Aumenta	Aumenta	Disminuye	Disminuye
Ki	Disminuye	Aumenta	Aumenta	Lo elimina	Disminuye
Kd	Aumenta	Disminuye	Disminuye	No afecta	Aumenta

Tabla 7.1: Efectos en el sistema que tiene el aumento de cada uno de los parámetros

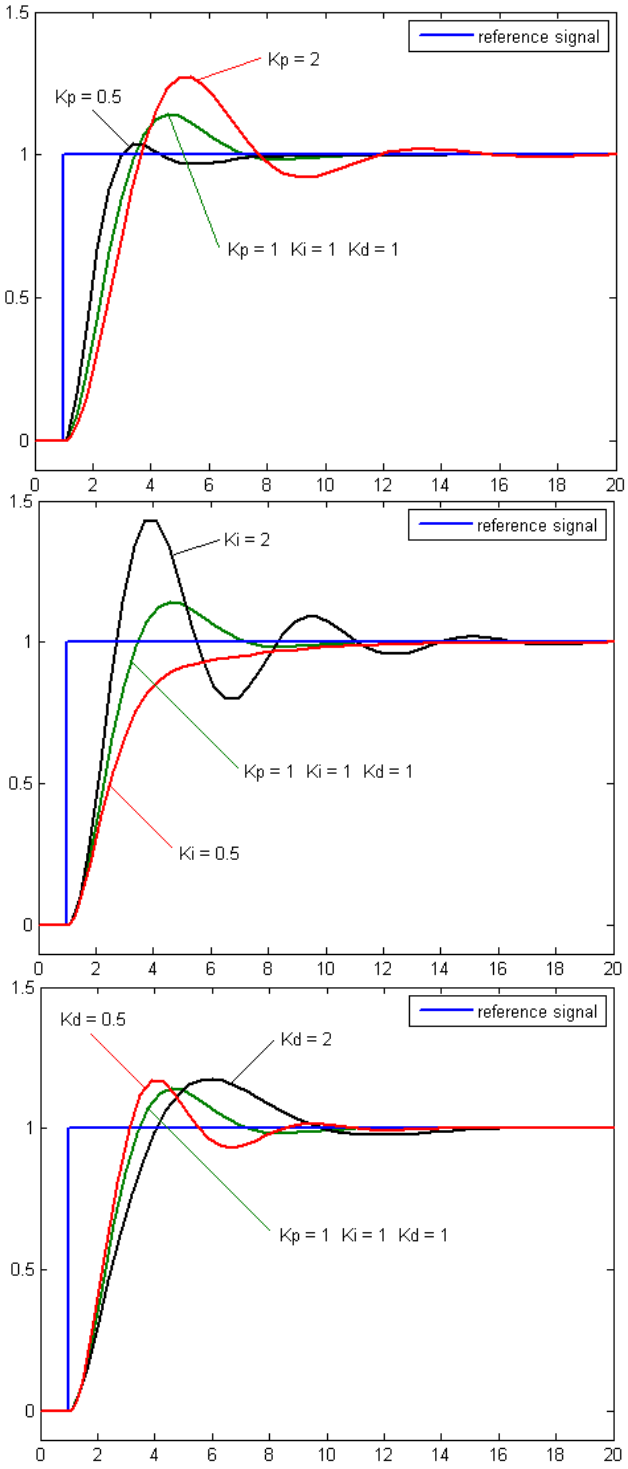


Figura 7-9: Resultados ante diferentes parámetros

7.5.2. SEGUNDO MÉTODO DE ZIEGLER-NICHOLS

Muchas veces no se conoce el modelo de un sistema e interesa regularlo en lazo cerrado sin necesidad de identificarlo, como es el caso de este robot. En estos casos son muy útiles las reglas de sintonización de Ziegler-Nichols. Estas reglas permiten el ajuste empírico de diversos controladores, incluido el PID.

Existen dos métodos de sincronización de Ziegler-Nichols, los cuales intentan asegurar una sobreoscilación del orden del 25% ante una entrada escalón[5]. El primero de ellos solo es aplicable a sistemas estables en lazo abierto, por lo que, al ser el sistema formado por el robot un sistema inestable, se desechó.

El segundo método de Ziegler-Nichols realiza el ajuste de los parámetros empleando los datos que se obtienen de la respuesta en lazo cerrado del sistema ante una entrada escalón. Este método no puede ser utilizado en sistemas que no se conviertan en marginalmente estables para ninguna ganancia. Dicho en otras palabras, que todas sus ramas del Lugar de las Raíces estén siempre en el semiplano complejo izquierdo, lo que supone la práctica mayoría de los sistemas de segundo orden, excepto los de fase no mínima. Para poder llevar a cabo este método hay que seguir los siguientes pasos:

i. Ajustas las ganancias integral y derivativa a cero.

ii. Partiendo de un valor bajo de la ganancia proporcional K_p se va aumentando esta gradualmente hasta conseguir una oscilación no amortiguada.

iii. Esta ganancia será la ganancia crítica K_c y el periodo de oscilación será T_c.

iv. Por último se ajustan los valores del controlador según los valores de la tabla.

Controlador	K_p	K_i	K_d
P	$K_c/2$	-	-
PI	$K_c/2'2$	$1'2K_p/T_c$	-
PID	$0'6K_c$	$2K_p/T_c$	$K_p \cdot T_c/8$

Tabla 7.2: Método de Ziegler-Nichols

De esta forma en el siguiente apartado se mostrará cómo se aplicó el segundo método de Ziegler-Nichols en el robot para obtener así los parámetros del controlador PID.

7.5.3. OBTENCIÓN DE LOS PARÁMETROS PID DEL ROBOT

La metodología que se va a proceder para obtener los parámetros PID del robot es la siguiente. En primer lugar se programará el robot para que envíe el estado de los sensores al ordenador. En el ordenador, para poder interpretar los resultados, se dibujará una gráfica con la ayuda de Matlab que representará la posición del robot con respecto a la línea a seguir. De esta forma, se igualarán a cero tanto K_i como K_d y empezando desde 1 se irá aumentando el valor de K_p hasta que se obtenga una oscilación no amortiguada.

Para poder llevar a cabo el método de Ziegler-Nichols lo que se hizo en primer lugar fue escribir la fórmula del controlador PID en forma discreta, apartado 7.5.1.4, de forma que la entendiera el Arduino Pro Mini. La fórmula está formada por cuatro términos. El primero de ellos es el determinará la velocidad de los motores cuando el error sea cero. El segundo término es el correspondiente a la parte proporcional del controlador. Esta parte estará formada por una constante de proporcionalidad, K_p, que multiplicará al error. El tercer término es el término integral. Está formado por una constante, K_i, y un término integral. Para obtener este término hay que añadir el error en cada instante de tiempo. Aunque parezca una fórmula abstracta, posee unidades. Debido a que los retrasos de tiempo en Arduino se miden en milisegundos, hay que añadir un término que convierta los milisegundos en segundos. El último término es el término derivativo. Está formado por una constante, K_d, y un término derivativo. Este se obtiene

como la diferencia entre el error actual y el error anterior en cada instante de tiempo. Al igual que en la parte integral, hay que añadir un término que convierta los milisegundos en segundos. De esta forma se obtiene:

Como se puede observar, el resultado que se obtiene se lleva a cada motor. Señalar que a cada motor se le aplica la corrección del controlador PID con signos diferentes. De esta forma se consigue que cuando se produzca un error el robot tienda a volver a la posición correcta. Para entenderlo mejor se va a suponer el siguiente supuesto:

$$\left.\begin{array}{l} errora = 10 \\ error = 8 \\ Kp = 2 \\ Ki = 20 \\ Kd = 1 \\ dt = 20 \\ vel = 160 \end{array}\right\} \begin{array}{l} motorA = 79 \\ motorB = 240 \end{array}$$

Debido a que el robot presenta un error actual inferior al error anterior está acercándose a la posición central. Además, como este error es positivo, recordando los pesos que se le da a cada sensor del apartado 7.4, la línea se encuentra en la parte derecha del robot. Así, para que el robot se vuelva a centrar este necesita girar a la derecha. Para conseguir eso debe ir más rápido su rueda izquierda, el motor B. Por lo tanto, en principio, los signos asignados a cada motor están bien escogidos.

Figura 7-10: Comprobación de los signos al aplicar el PWM en el robot

Tras obtener la expresión que empleará el Arduino Pro Mini para ejecutar el controlador PID se procedió a pensar en el circuito que se empleará en la prueba. En el método de Ziegler-Nichols se debe someter al sistema a una entrada escalón. En este caso se pensó que se podría emplear el siguiente circuito:

Figura 7-11: Circuito que se empleará para la prueba

En un primer lugar el robot encuentra una zona recta sin perturbaciones. Después encuentra el escalón. Y por último vuelve a una zona recta sin perturbaciones. Es importante que en ningún momento el robot deje de detectar la línea, por lo que el escalón no debe ser muy grande. De esta forma, según el método de Ziegler-Nichols, lo que se pretende es que tras el escalón el robot empiece a oscilar de una forma constante:

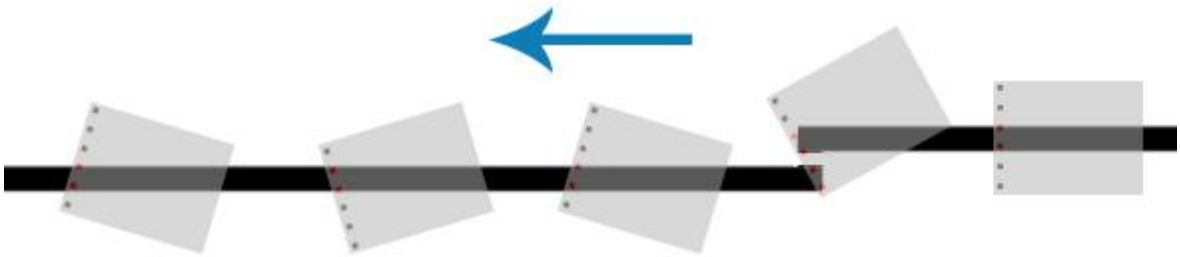


Figura 7-12: Efecto deseado en la prueba

Además, para evitar que el robot tuviera que calibrarse cada vez que iniciaba la prueba, se hicieron varias medidas sobre el circuito y se hizo una media con los valores máximos y mínimos obtenidos para cada sensor.

También se realizó un programa en Matlab para interpretar los datos (ANEXO A.11). En primer lugar, al igual que en el apartado 6.3.1, se creó un objeto serie y se abrió. También se definió el tiempo máximo, así como el número de muestras que se toma por segundo.

```
%borrar previos
delete(instrfind({'Port'},{'COM6'}));
%crear objeto serie
s = serial('COM6','BaudRate',57600,'Terminator','CR/LF');
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%abrir puerto
fopen(s);
% parámetros de medidas
tmax = 10; % tiempo de captura en s
rate = 33; % resultado experimental
```

Después se preparó la figura que representaría los resultados y se inicializaron las diferentes variables que se emplearían más adelante.

```
% preparar la figura
f = figure('Name','Captura');
a = axes('XLim',[0 tmax],'YLim',[0 2]);
l1 = line(nan,nan,'Color','r','LineWidth',2);
l2 = line(nan,nan,'Color','b','LineWidth',1);
xlabel('Tiempo (s)')
ylabel('Error')
title('Captura de posición en tiempo real con Arduino')
grid on
hold on
% inicializar
v1 = zeros(1,tmax*rate);
v2 = zeros(1,tmax*rate);
i = 1;
t = 0;
```

Tras esto, de nuevo con la ayuda de las variables *tic* y *toc*, se ejecuta el bucle de toma de datos.

```
% ejecutar bucle cronometrado
tic
while t<tmax
    t = toc;
    % leer del puerto serie
    puerto = fscanf(s,'%d %d')';
    v1(i)=puerto(1)/25;
    v2(i)=puerto(2)/25;
    % dibujar en la figura
    x = linspace(0,i/rate,i);
    set(l1,'YData',v1(1:i),'XData',x);
    set(l2,'YData',v2(1:i),'XData',x);
    drawnow
    % seguir
    i = i+1;
end
```

En primer lugar se lee el puerto serie que se ha creado al inicio del programa y se le indica la forma en la que le llegan los datos. El primer dato es el error, mientras que el segundo es la referencia que sigue, en este caso se busca que el error sea 0. Tras esto se dibujan los resultados. De esta forma se obtuvieron las siguientes gráficas:

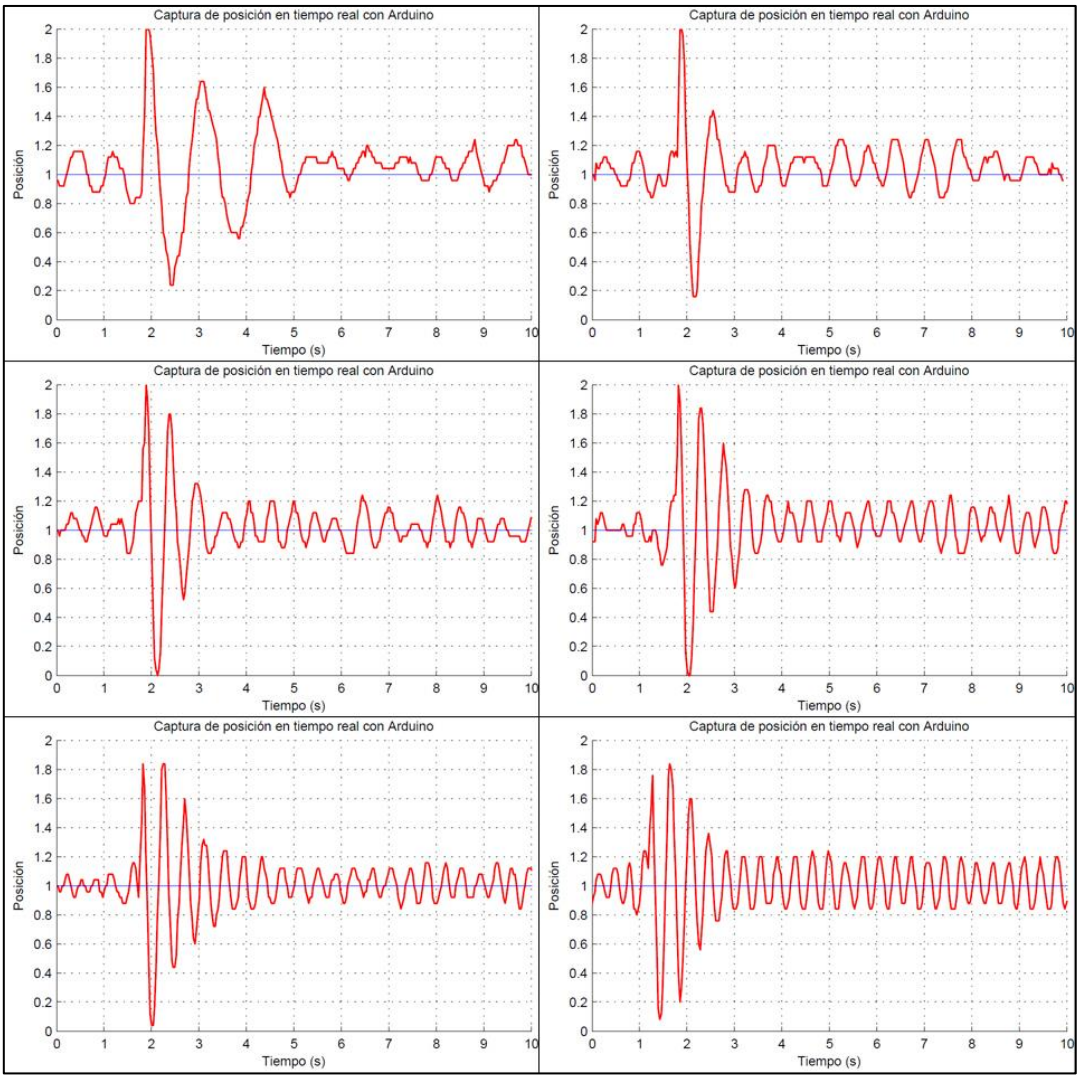


Figura 7-13: Resultados de la prueba. De izquierda a derecha y de arriba abajo: K_p=1, K_p=3, K_p=5, K_p=7, K_p=8, K_p=9

Para finalmente obtener:

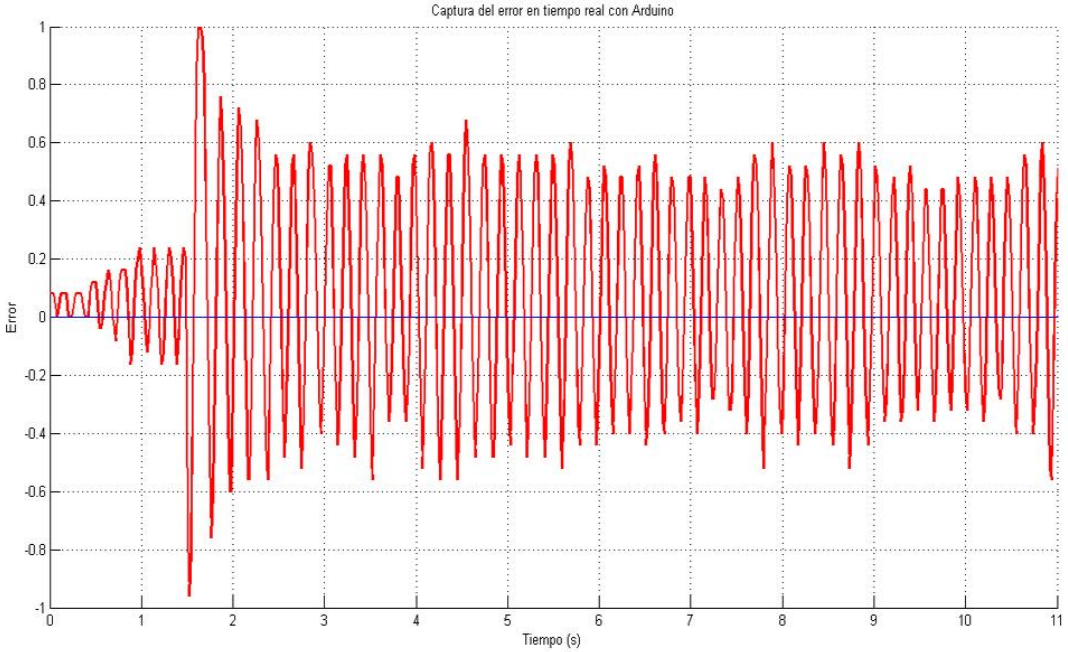


Figura 7-14: Resultado de la prueba para K_p=10

En esta última gráfica se puede observar como el sistema tras la primera oscilación el sistema sigue oscilando de forma constante. De esta forma, los resultados que se obtuvieron fueron:

$$K_c = 10 \quad T_c = 0'18 \quad \begin{cases} K_p = 6 \\ K_i = 66'67 \\ K_d = 0'135 \end{cases}$$

Así, una vez definidos los parámetros del PID se tenía prácticamente completo el *sketch*. Para finalizarlo se trató qué debería realizar el robot en caso de salirse del trazado.

7.5.4. COMPARATIVA DE PARÁMETROS

A continuación se va a estudiar el comportamiento del robot si se varían los parámetros del controlador PID. En primer lugar se va estudiar el comportamiento del robot con los parámetros obtenidos por el método de Ziegler-Nichols.

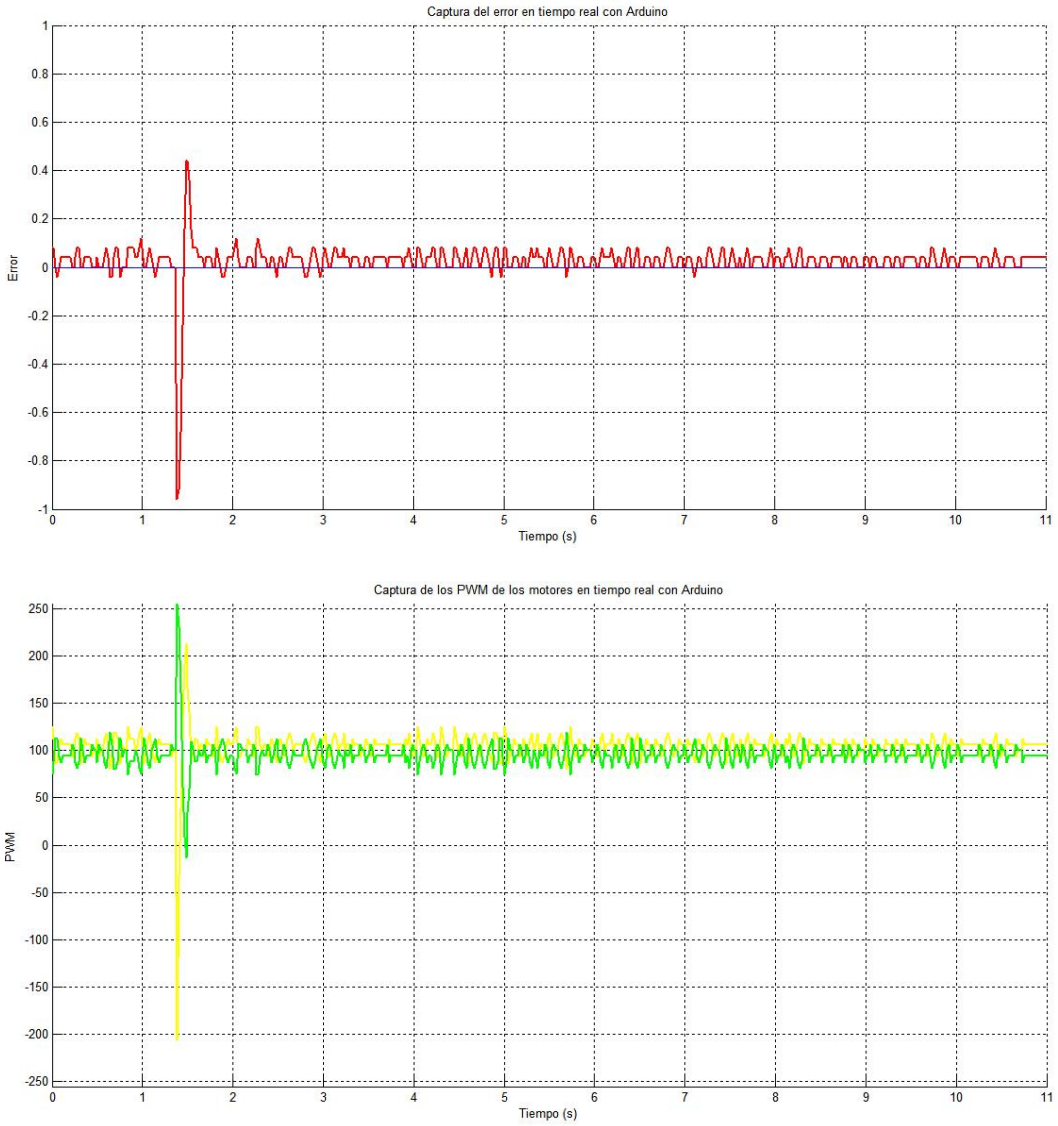


Figura 7-15: Resultados empleando los parámetros obtenidos en el método de Ziegler-Nichols. Arriba se representa el error y abajo los PWM de los motores

Se puede observar que con el PID obtenido por el método de Ziegler-Nichols que, aunque aparece una sobreoscilación del 40%, el tiempo de estabilización es de tan solo 0’3 segundos. Además se puede apreciar en la gráfica de los motores que se encuentran prácticamente todo el rato girando a la velocidad nominal (en este caso 100). Por lo que se puede concluir que este controlador es muy apropiado para la finalidad de este robot.

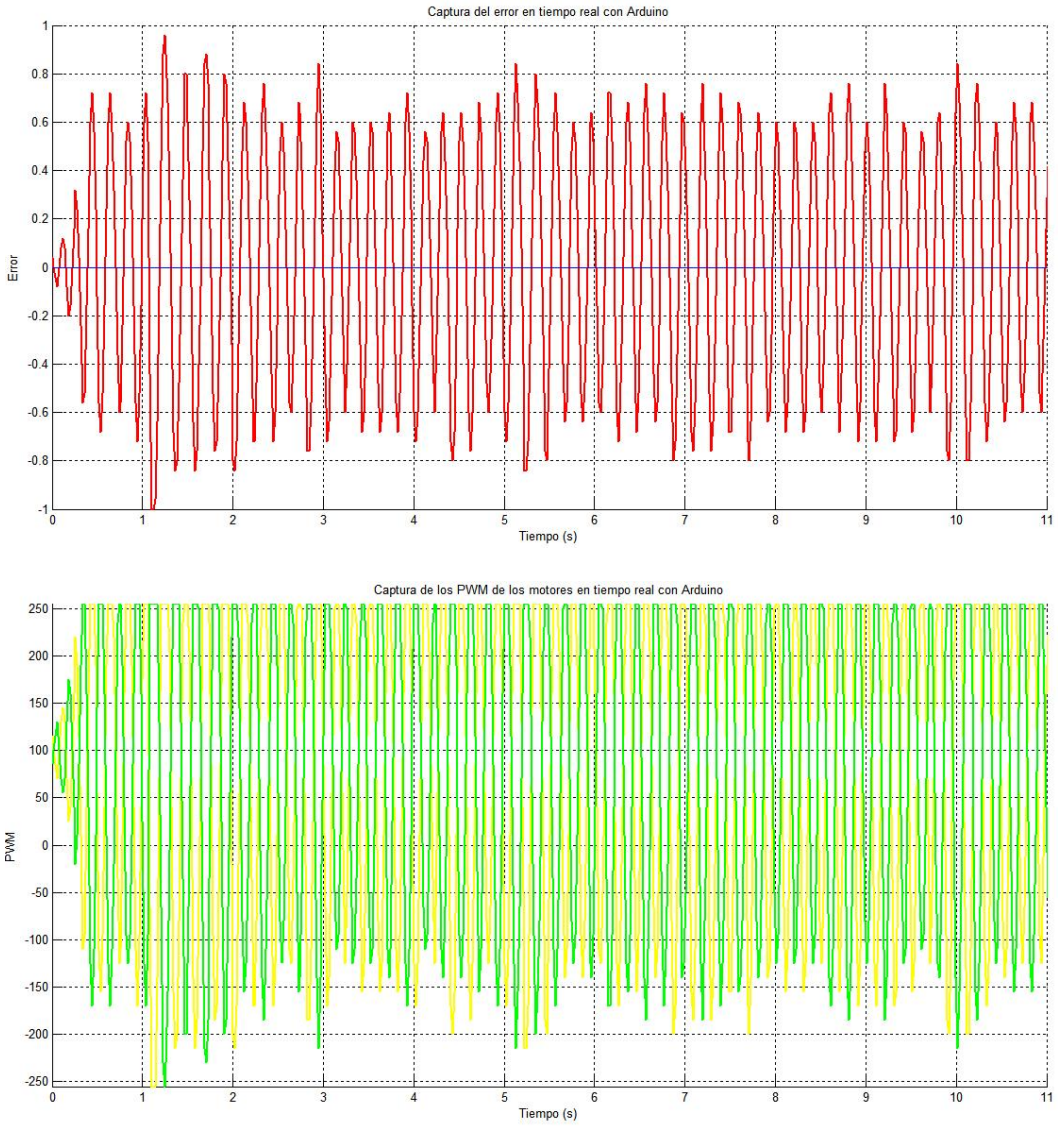


Figura 7-16: Resultados empleando K_p=15. Arriba se representa el error y abajo los PWM de los motores

Al aumentar la K_p hasta 15 se puede observar que el sistema se vuelve muy inestable ya que, como ya se había comentado, al aumentar K_p la sobreoscilación aumenta y la estabilidad del sistema disminuye. El sistema nunca llega a estabilizarse y los motores se encuentran girando a sus velocidades máximas.

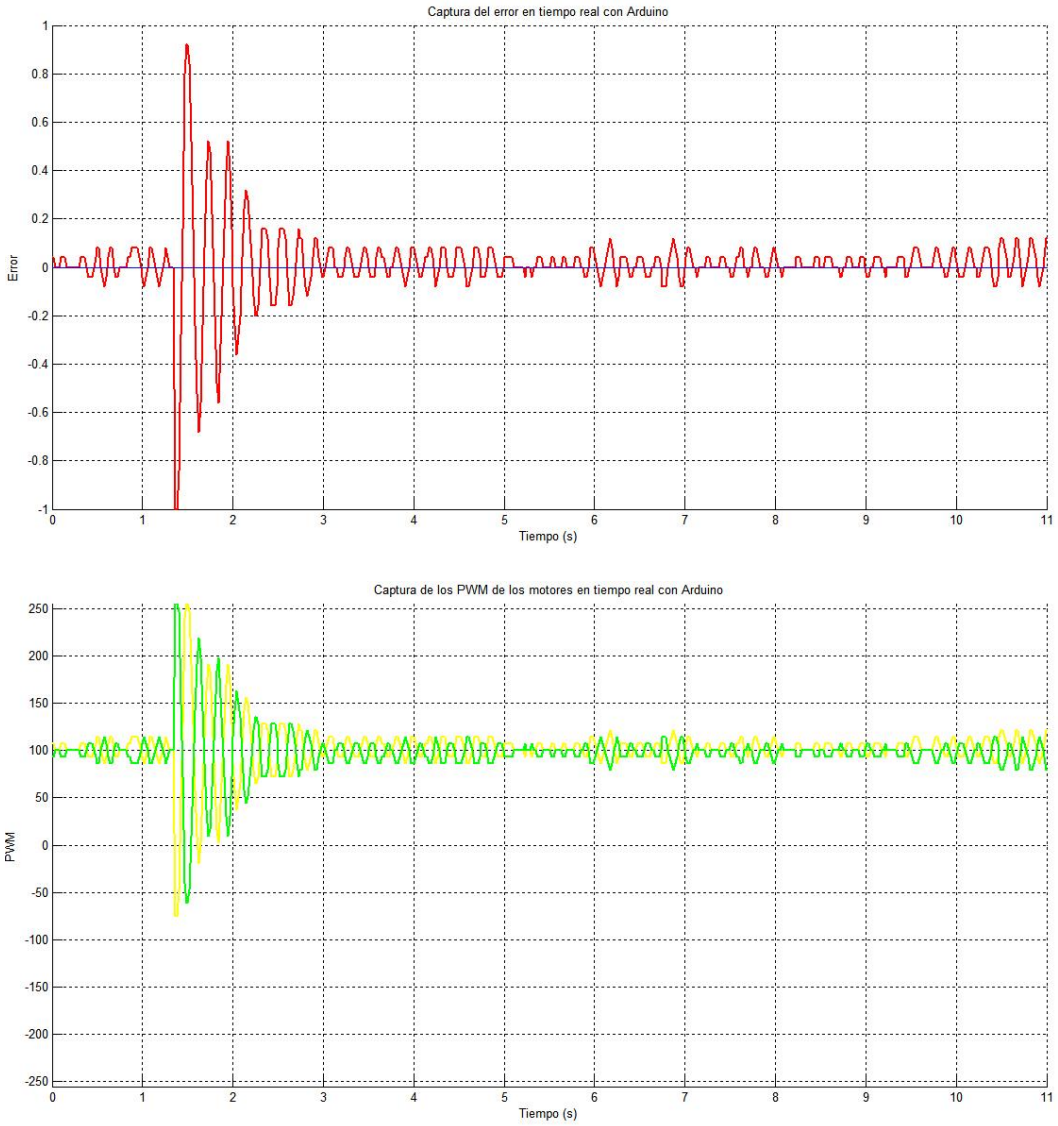


Figura 7-17: Resultados empleando K_i=250. Arriba se representa el error y abajo los PWM de los motores

Si se aumenta la K_i hasta 250 se observa como el sistema llega a estabilizarse pero tarda mucho más tiempo que con los parámetros obtenidos por el método de Ziegler-Nichols. De esta forma, aparece una sobreoscilación del 92% y el tiempo de oscilación asciende hasta los 1’76 segundos. Además, tras la perturbación se puede ver como los movimientos de los motores son bruscos durante un largo periodo de tiempo.

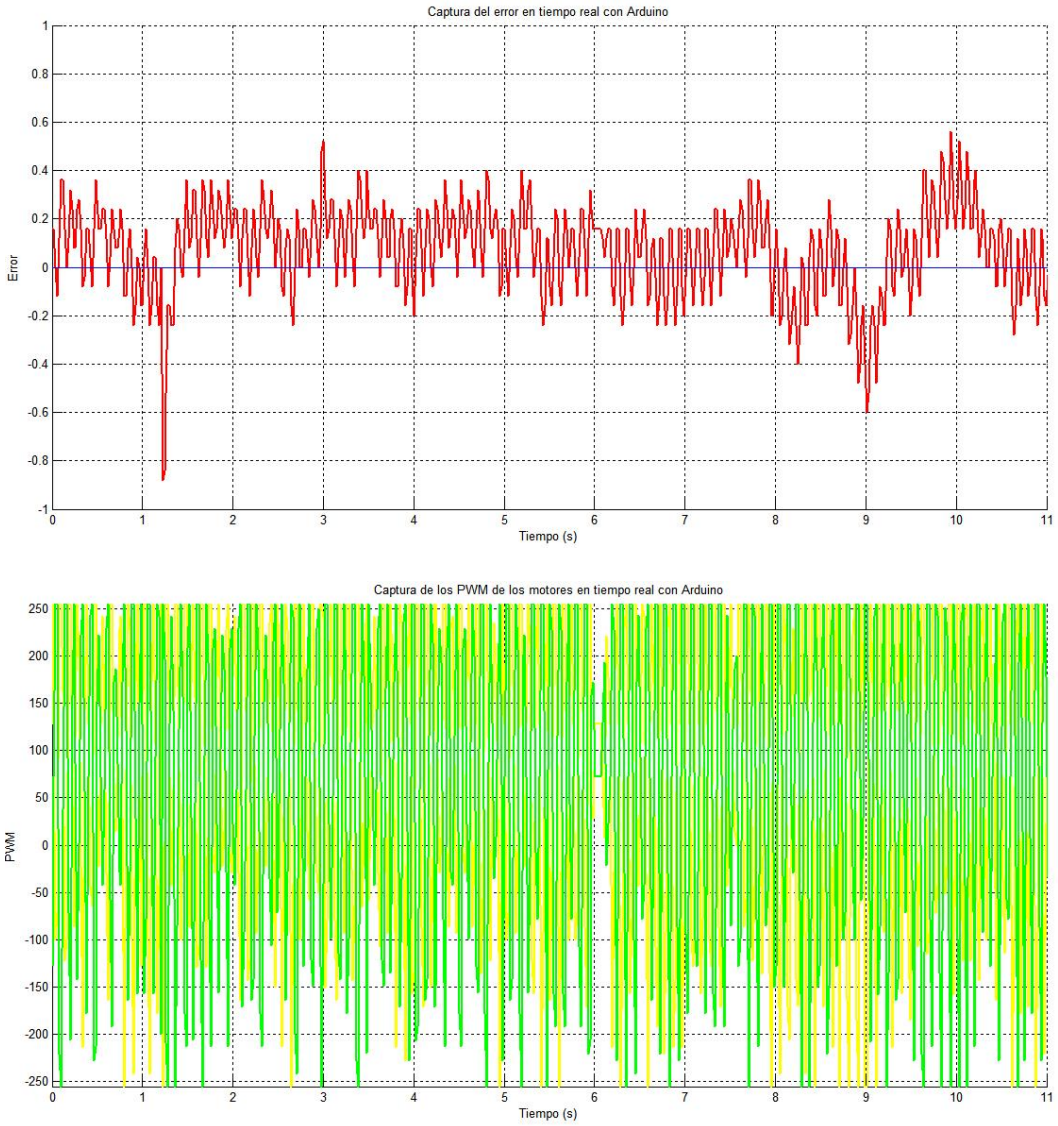


Figura 7-18: Resultados empleando K_d=1. Arriba se representa el error y abajo los PWM de los motores

Al aumentar la K_d hasta 1, una vez más, se obtienen los resultados esperados. El sistema no consigue estabilizarse e incluso se puede apreciar como la señal del error posee cierta componente de una señal de alta frecuencia. Como ya se había comentado, la parte derivativa lo que hace es amplificar el ruido del sistema. Así, se puede ver en la gráfica de los motores que en todo momento se encuentran en sus velocidades máximas.

De esta forma, se concluye que los parámetros para el controlador PID obtenidos por el método de Ziegler-Nichols son muy buenos y se escogen como parámetros finales del robot.

7.6. SALIDA DEL TRAZADO

Tras explicar los pasos que sigue el robot para seguir la línea se va a explicar qué hace el robot cuando se sale del trazado. La principal causa de que se salga el robot, por no decir la única, es que se produzca un giro demasiado brusco. De esta forma, se forzará al robot a girar sobre sí mismo para volver al trazado por el punto que había perdido la línea.

Como se ha explicado en el punto 7.4, si al menos un sensor detecta línea se ejecuta la función *si_linea*. Sin embargo, si ningún sensor detecta línea se ejecuta la función *no_linea*.

Esta función lo que hace es forzar unas velocidades de giro al robot según la última posición en la que se detectó la línea. De esta forma si se vio la línea por última vez a la izquierda se fuerza a que el robot gire a la izquierda, si se vio la línea por última vez a la derecha se fuerza a que el robot gire a la derecha y si se vio en el centro se obliga a ir marcha atrás.

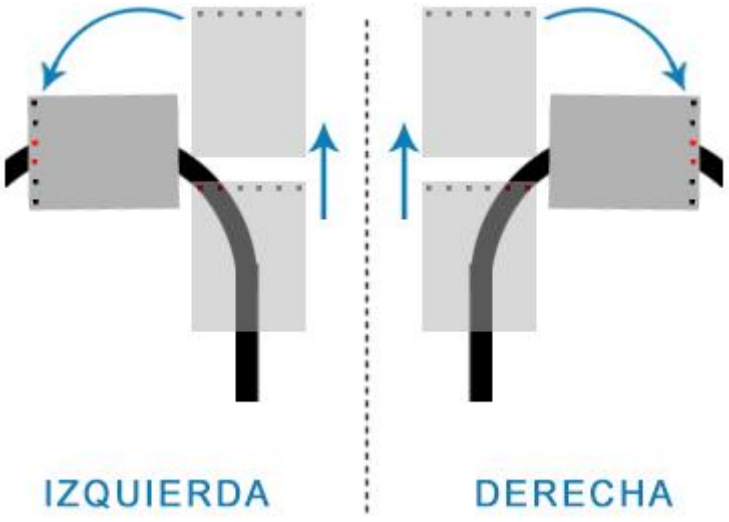


Figura 7-19: Salida del trazado del robot

CAPÍTULO 8. CONTROL REMOTO DEL ROBOT

La idea de poder controlar el robot como si fuera un coche de radiocontrol nació cuando, gracias a la aplicación *BlueTerm*, se consiguió controlar los motores del robot. De esta forma, en los siguientes apartados se desarrollará cómo se realizó la aplicación Android y la programación del robot. El diagrama de todo el sistema es el siguiente:

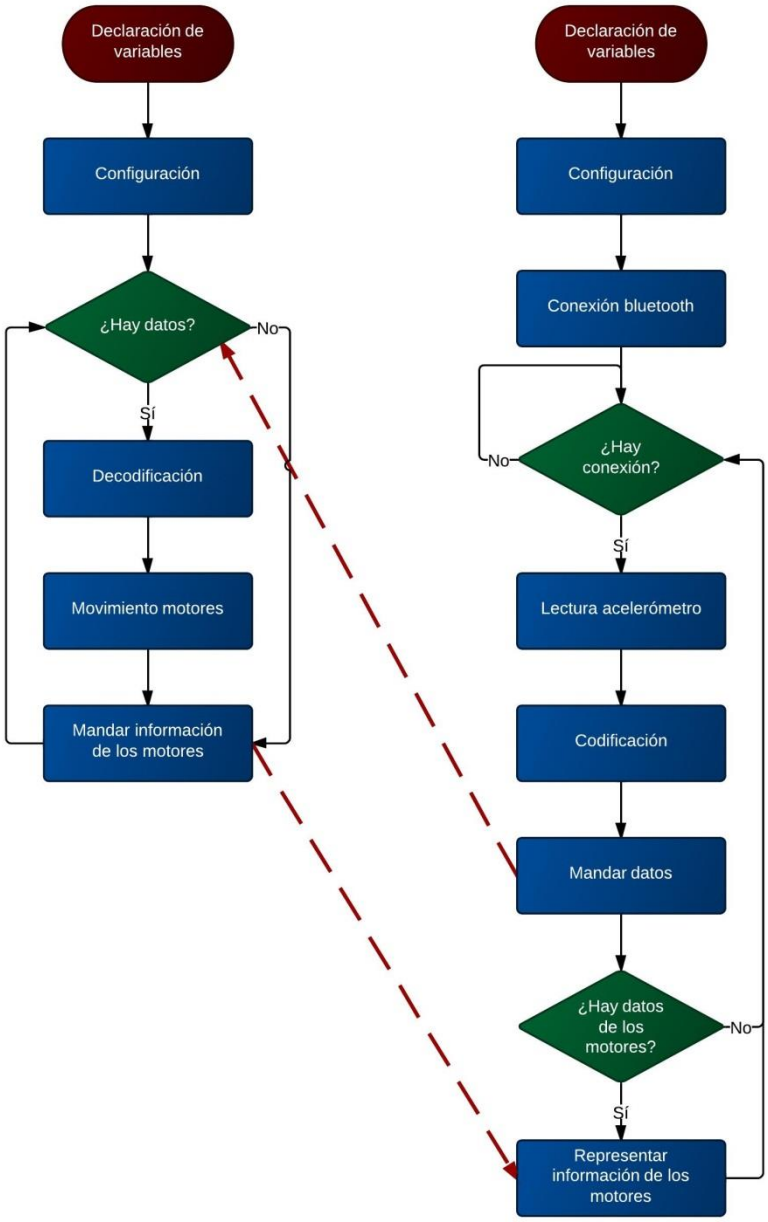


Figura 8-1: Diagrama del funcionamiento del control remoto del robot. A la izquierda se representa el *sketch* del robot. A la derecha se representa la aplicación Android

Se puede ver que el sistema está formado por dos partes. Por un lado el robot (a la izquierda) (ANEXO.A.12) y por otro lado la aplicación Android (ANEXO A.13). En primer lugar se va a explicar cómo se realizó la aplicación Android y después el programa del robot.

8.1. APLICACIÓN ANDROID

En primer lugar se pensó el lenguaje que se iba a utilizar para realizar la aplicación. Antes de comenzar con el proyecto solo conocía Visual Basic y C, por lo que se investigó si era posible desarrollar una aplicación Android a través de esos lenguajes de programación.

Tras una breve búsqueda se llegó a un programa llamado Basic4Android[35]. Este es un entorno para el desarrollo de aplicaciones para Android que permite programar de la misma forma que se hace en Visual Basic. Esto quiere decir que se tiene un entorno gráfico en donde se pueden añadir botones, ventanas y otros componentes en apenas unos cuántos clicks. Además cuenta con una amplia variedad de librerías para trabajar con bases de datos, bluetooth, GPS, cámara,…

Figura 8-2: Logo de Basic4Android

Pero la gran desventaja de este programa es que no es software libre, por lo que hay que pagar una licencia para poder utilizar el programa en plenas condiciones. Existe una versión de prueba, pero no permite cargar ningún tipo de librería, por lo que está muy

limitado. El precio de esta licencia oscila entre 50 y 500€, por lo que se desechó esta opción.

Buscando más se llegó a dar con Amarino[36]. Amarino está formado por una librería de Arduino junto a una aplicación Android. De esta forma se puede comunicar de una forma sencilla un dispositivo Android con una tarjeta de Arduino. Así, mediante una conexión bluetooth, se pueden controlar los diferentes pines del Arduino: nombrar los pines como pines de entrada o salida, controlar los pines digitales PWM,…

Figura 8-3: Logo de Amarino

Su principal ventaja es que es software libre y además existen bastantes recursos por internet. Pero al igual que se había desechado desde un primer momento usar directamente la aplicación *BlueTerm*, se buscaba realizar una aplicación entera, incluida la interfaz, y esto no lo permitía Amarino.

Después se pensó en programar directamente en Java con la ayuda del SDK de Android[37]. Su gran ventaja, y a la vez desventaja, según como se mire, es la cantidad de información que hay disponible por internet. Esto requería un gran trabajo de documentación; eso sin contar que desconocía por completo java, por lo que tendría que aprender a programar en este lenguaje. De esta forma se desechó esta opción.

Tras desechar la opción de programar directamente en java se pensó en utilizar Processing[38]. Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java.

Figura 8-4: Logo de Processing

La principal ventaja de emplear este lenguaje es que este es el lenguaje empleado por Arduino, por lo que gran parte del aprendizaje ya se tenía. Además, la página oficial del programa cuenta con una gran cantidad de ejemplos y tutoriales que facilitan su aprendizaje. Por otro lado, cuando ya me puse a programar la aplicación en sí, me di cuenta de que muchos conceptos no los tenía bien asimilados en este lenguaje de programación, por lo que los errores de compilación eran frecuentes y el programa no era nada explícito en cuanto a cuáles eran las causas de estos errores.

Finalmente se decidió emplear una mezcla entre las dos últimas opciones estudiadas. Se programó usando como lenguaje de programación Processing, pero se empleó como IDE Eclipse y se instaló el SDK de Android. De esta forma, pude trabajar en un lenguaje que conocía, Processing, y Eclipse me iba advirtiendo de los errores que se iban produciendo en el programa.

Figura 8-5: Conexión Processing y Eclipse

La conectividad entre Processing y Eclipse es muy buena. Para poder pasar el código realizado en Processing se debe ir en el propio Processing a File -> Export Android Project:

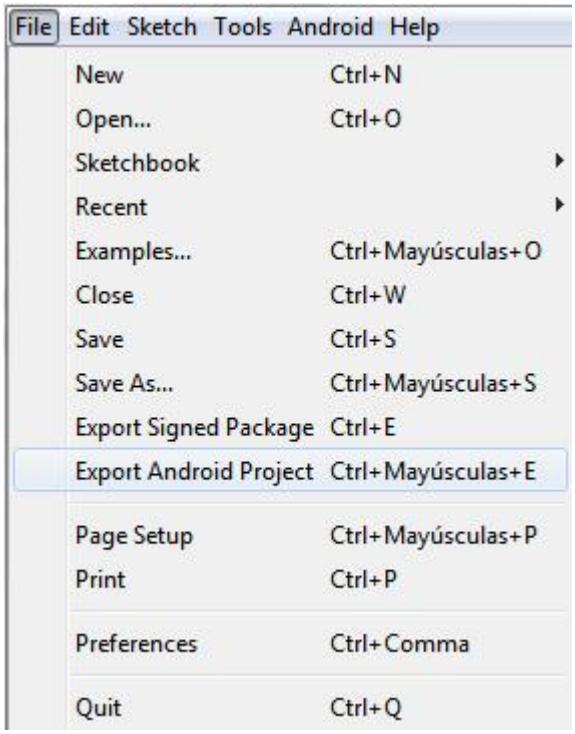


Figura 8-6: Exportación de proyectos Android en Processing

Tras esto se debe abrir Eclipse y seguir la siguiente ruta: File -> New -> Project -> Android Project from Existing Code. Tras esto se elige la ruta donde ha exportado Processing el código y ya se tiene disponible el código en Eclipse.

Una vez comprobada la conexión entre Processing y Eclipse se pasó a desarrollar la aplicación Android.

8.1.1. DECLARACIÓN DE VARIABLES

En este apartado del código se declaran las variables que se van a emplear a lo largo de todo el programa, entre las que destacan las relacionadas con el manejo de los acelerómetros y del control del bluetooth.

8.1.1.1. VARIABLES DEL ACELERÓMETRO

Para poder manipular los acelerómetros se van a necesitar cuatro variables[39]. Estas son:

- *mSensorManager*: esta variable de tipo *SensorManager* es la permite acceder a los diferentes sensores que posee un teléfono Android.
- *accSensorEventListener*: esta variable de tipo *MySensorEventListener* es la encargada de recibir la información que le envían los sensores.
- *acc_sensor*: esta variable de tipo *Sensor* permite elegir el tipo de sensor que se quiere emplear. En la actualidad existen multitud de tipos de sensores en los dispositivos Android, tales como acelerómetros, de temperatura, giroscopios, de luz, magnéticos...
- *acc_values*: esta variable es un vector de variables tipo *float* que será la encargada de almacenar los valores de los acelerómetros.

8.1.1.2. VARIABLES DE LA COMUNICACIÓN BLUETOOTH

Para poder realizar la comunicación bluetooth entre el teléfono Android y el robot de forma correcta se van a necesitar seis variables[40].

- *dispositivos*: esta variable es un vector de variables tipo *BluetoothDevice* que servirá para almacenar los dispositivos bluetooth que encuentre el teléfono.
- *bttelefono*: esta variable de tipo *BluetoothAdapter* representa el bluetooth del teléfono.
- *dispositivo*: esta variable de tipo *BluetoothDevice* representa el bluetooth de los dispositivos ajenos al teléfono.
- *socket*: esta variable de tipo *BluetoothSocket* es la encargada de establecer la conexión entre el teléfono y el robot.
- *ins*: esta variable de tipo InputStream permite recibir datos del robot. El formato de estos datos es el byte.
- *ons*: esta variable de tipo OutputStream permite enviar datos al robot. El formato de estos datos es el byte.

8.1.2. CONFIGURACIÓN

En la función setup se declaran diferentes propiedades de la aplicación. Con la función *size* se obtiene el ancho y alto de la pantalla del teléfono. Esto será de gran utilidad a la hora de querer representar cosas por pantalla ya que no todos los dispositivos Android tienen las mismas dimensiones. Así, al asignar el ancho y el alto de la pantalla a unas variables se podrán ofrecer datos por pantalla en relación a estas medidas.

También se fijará que la posición de la pantalla sea apaisada con la función *orientation* y que el tipo de fuente que emplee la aplicación sea la que emplea por defecto el teléfono. Además se evitará que la pantalla se apague mientras se esté ejecutando la aplicación al emplear el flag *FLAG_KEEP_SCREEN_ON*.

Por último se instancian dos variables, *pwmA* y *pwmB*, y se definen todas sus variables para más adelante poder representar el estado de los motores.

8.1.3. CONEXIÓN BLUETOOTH

Tras realizarse la configuración general de la aplicación se establece la conexión bluetooth. La primera función que ejecuta la aplicación es la función *onStart*. En ella se detecta el estado del bluetooth. Si el teléfono no dispone de bluetooth mostrará un mensaje de error y saldrá de la aplicación. Por otro lado, si el teléfono dispone de bluetooth pero no está activado se le pedirá al usuario que lo active. De esta forma habrá que darle a la aplicación permisos para manipular el bluetooth del teléfono. Para eso habrá que ir al *AndroidManifest.xml* y después a la pestaña *Permissions* y una vez aquí añadir los permisos para manipular el bluetooth.

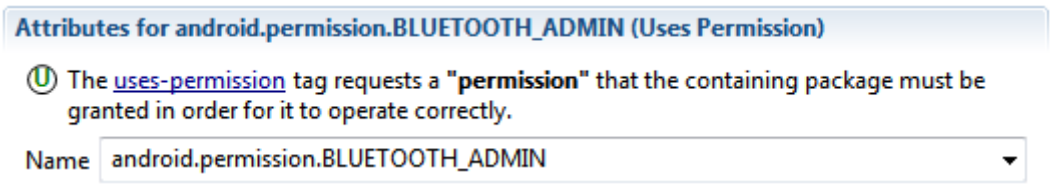


Figura 8-7: Ventana de permisos de Eclipse

Después se ejecuta la función *empieza*. Esta función añade los dispositivos bluetooth que estén vinculados con el teléfono a una lista para posteriormente mostrarlo por pantalla mediante la función *listaDispositivos*.

```
dispositivos = new ArrayList<BluetoothDevice>();
    for (BluetoothDevice dispositivo : bttelefono.getBondedDevices()){
        dispositivos.add(dispositivo);
    }
```

Para mostrar los dispositivos de una forma en la que se puedan escoger de forma sencilla, se mostrará de cada dispositivo tanto el nombre como como la dirección mac del dispositivo.

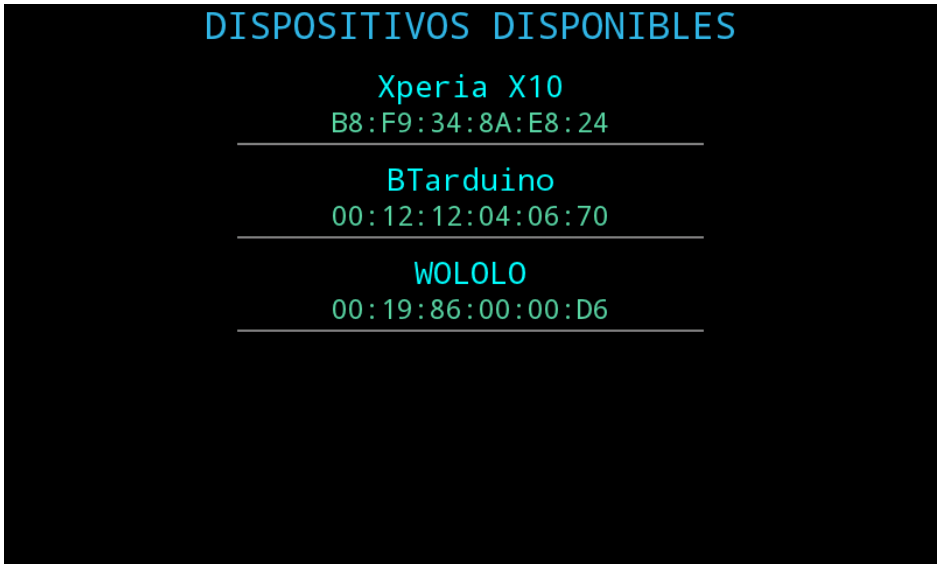


Figura 8-8: Interfaz de la aplicación Android de selección de dispositivo a conectar

Después mediante la función *compruebaEleccion* se verifica que se haya pulsado la pantalla en un lugar correspondiente a un dispositivo y se procede a la conexión bluetooth entre el teléfono y el robot mediante la función *conectaDispositivo*.

Para conectar ambos dispositivos en primer lugar se crea socket para que pueda existir dicha conexión. Para ello se emplea la codificación *UUID*. Un *UUID* (del inglés, ***U**niversally **U**nique **ID**entifier*) es un identificador estándar usado en el desarrollo de software. Lo que intentan los *UUID* es proporcionar un identificador de información único a cada tipo de dispositivo. De esta forma, si se consulta el SDK de Android este indica que el *UUID* del bluetooth es:

$$00000000 - 0000 - 1000 - 8000 - 00805F9B34FB$$

Sin embargo, la conexión que se va a realizar no es una conexión al uso, sino que se va a simular un puerto serie. Para ello hay que variar los primeros términos. Así se obtiene que el *UUID* para realizar una conexión bluetooth simulando un puerto serie es el siguiente[41]:

$$00001101 - 0000 - 1000 - 8000 - 00805F9B34FB$$

Una vez definido correctamente el *UUID* solo queda conectar el socket e indicar cuál será la forma de recibir y enviar datos. Para ellos se emplean las extensiones *getInputStream* y *getOutputStream*, respectivamente. De esta forma, la conexión queda establecida.

```
socket = dispositivo.createRfcommSocketToServiceRecord(UUID.fromString("00001101-
0000-1000-8000-00805F9B34FB"));
Method m = dispositivo.getClass().getMethod("createRfcommSocket", new Class[] {
int.class });
socket = (BluetoothSocket) m.invoke(dispositivo, 1);
socket.connect();
ins = socket.getInputStream();
ons = socket.getOutputStream();
```

8.1.4. CONFIGURACIÓN DE LOS ACELERÓMETROS

Cuando se pensó en controlar el robot a través de un dispositivo Android enseguida se pensó que los acelerómetros eran los sensores más indicados. Un acelerómetro es un sensor que mide la aceleración relativa, tomando como referencia la de caída libre, es decir, en la Tierra la referencia será $g = 9'81 m/s^2$. Por lo tanto, la medida que ofrece un acelerómetro es[42]:

$$A = g - \frac{\sum F}{m}$$

Esto quiere decir que, si el dispositivo Android se encuentra en reposo, la media del acelerómetro será $A = g = 9'81 m/s^2$. Sin embargo, si sufre una caída libre se obtendrá como medida $A = 0$.

En un dispositivo Android el sistema de coordenadas es relativo a la pantalla y no cambia con los cambios de orientación[7]:

- Eje x: es horizontal y apunta a la derecha.
- Eje y: es vertical y apunta hacia arriba.
- Eje z: es la normal saliente de la pantalla del dispositivo.

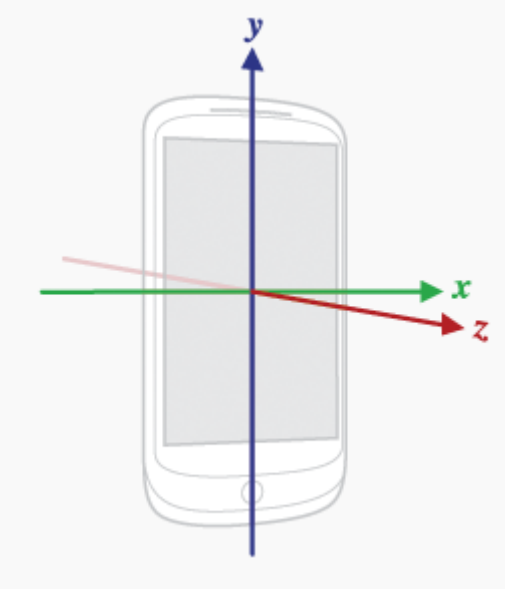


Figura 8-9: Representación de los ejes de los acelerómetros en un dispositivo Android

La lectura de los acelerómetros no es algo que se realice de forma puntual, sino que es un proceso que se va a estar realizando continuamente en segundo plano. Es por ello por lo que la configuración del acelerómetro se incluye dentro de la función *onResume*. En primer lugar lo que se debe hacer es definir el *SensorManager*, ya que es el encargado de permitir los accesos a los sensores del teléfono:

```
mSensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
```

Después se debe crear una variable que recoja los datos que le envíen los sensores. Esta será la variable que recoja las variaciones de los acelerómetros:

```
accSensorEventListener = new MySensorEventListener();
```

Después se eligen los sensores de los que se quiera extraer información. En este caso solo se tendrán en cuenta los acelerómetros. Si se deseara algún otro sensor solo habría que cambiar el *TYPE*:

```
acc_sensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Por último se asocia el *SensorEventListener*, junto al sensor tipo de sensor que hace referencia y al *SensorManager* empleado. Además también hay que indicar el tipo de *delay* que se le quiere dar a la aplicación. Existen cuatro tipos: *SENSOR_DELAY_FASTEST*, para obtener los datos de los sensores lo más rápido posible; *SENSOR_DELAY_GAME*, velocidad recomendada para juegos; *SENSOR_DELAY_NORMAL*, es la recomendada para cambios en la pantalla principal del teléfono y *SENSOR_DELAY_UI*, velocidad utilizada por la interfaz d usuario. Tras diversas pruebas, el que ofreció mejores resultados fue el *SENSOR_DELAY_GAME*:

```
mSensorManager.registerListener(accSensorEventListener, acc_sensor,
SensorManager.SENSOR_DELAY_GAME);
```

Tras esto el acelerómetro quedaría configurado y ya se podrían obtener sus lecturas.

8.1.5. LECTURA DE LOS ACELERÓMETROS

Para poder leer los acelerómetros correctamente primero hay que tener en cuenta tres aspectos de los acelerómetros:

- Los datos de los acelerómetros quedan recogidos en la variable *acc_sensor*. Esta variable es un vector de tres componentes, en el que la primera es el valor del acelerómetro del eje x, la segunda es el valor del acelerómetro del eje y la tercera es el valor del acelerómetro del eje z.
- Estos valores, recordando la fórmula anterior, deberían estar acotados por ± 10, por lo que habrá que incluir algún filtro para evitar lecturas no deseadas.
- Por último estos valores tienen mucho ruido, por lo que habrá que incluir otro filtro para filtrar este ruido.

Lo primero en lo que se da cuenta uno cuando empieza a manipular los acelerómetros es en su inestabilidad. Los valores que ofrecen los acelerómetros son números reales que incluso dejando el teléfono en reposo sobre la mesa cambian varias veces por segundo en la primera cifra decimal. Si se llevara este número tal cual al robot se produciría en este un movimiento nervioso nada deseable.

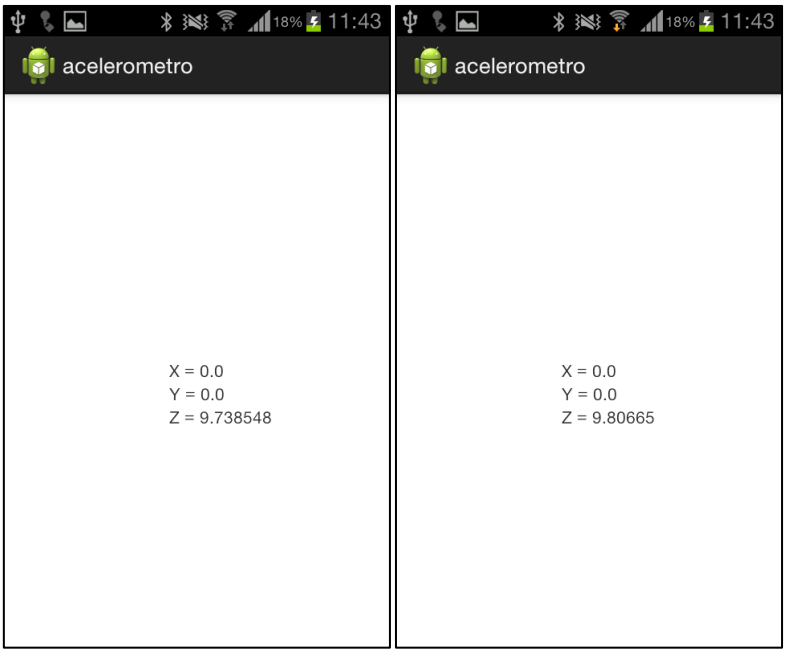


Figura 8-10: Valores de los acelerómetros en dos instantes consecutivos estando el dispositivo Android en reposo sobre la mesa

De esta forma se llevan a cabo tres filtros dentro de la función *muestraDatos*. El primero de ellos, y quizás el más grande, es el de limitar los valores de los acelerómetros a números enteros. Se puede pensar que se pierde gran parte de información, pero no es así. Al estar limitados los valores de los acelerómetros entre ± 10, se disponen de 20 posiciones posibles. Además, al disponer los sensores de simetría con respecto a sus ejes, estos solo tienen 180° efectivos. Esto hace que se dispongan de 9° de sensibilidad para manipular el robot, más que suficientes para controlar el robot de una forma óptima.

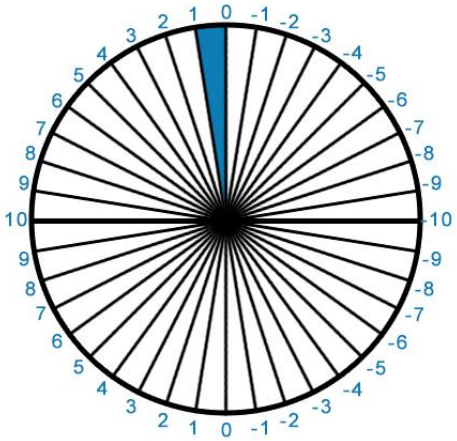


Figura 8-11: Valores que puede tomar el acelerómetro del eje x. Representación de 9°

El segundo filtro que se realiza es para evitar que el ruido haga que se tome una mala medida. Para ello lo que se hace es tomar como dato que se enviará al robot la media de las dos últimas medidas.

El último filtro, el de limitar los valores para que estén entre ± 10, se llevará a cabo en el propio robot. Es el filtro que menos actúa, ya que es raro que un acelerómetro ofrezca como resultado más de 10. Así que tras realizar diversas pruebas se obtuvieron mejores resultados cuando este filtro lo realizaba directamente el robot.

8.1.6. ENVÍO Y RECEPCIÓN DE DATOS

Tras haber filtrado los valores de los acelerómetros se procede al envío de estos al robot. En este caso se va a emplear el acelerómetro del eje x para manipular la velocidad de avance del robot y el acelerómetro del eje y para manipular el giro del robot.

Tanto la función que se va a emplear para enviar datos como para recibirlos, write y read respectivamente, utilizan como datos bytes. En primera instancia se pensó que no supondría mayor problema, pero tras empezar a realizar pruebas sí surgieron problemas. En java un byte permite manipular números enteros desde -128 hasta 127, ambos incluidos[43]. Sin embargo, en Arduino un byte permite manipular números desde 0 hasta 255, ambos incluidos[44]. Esto hace que en ambas plataformas un byte permita manipular 256 números, pero cada plataforma ofrece unos límites distintos, ya que por ejemplo Arduino no puede procesar de forma correcta un byte que representa un número negativo. Para solventar esto, antes de enviar los datos al robot se suma 10 a los valores de los acelerómetros. De esta forma no pueden aparecer valores negativos y Arduino puede entender de forma correcta los números que se le van a enviar.

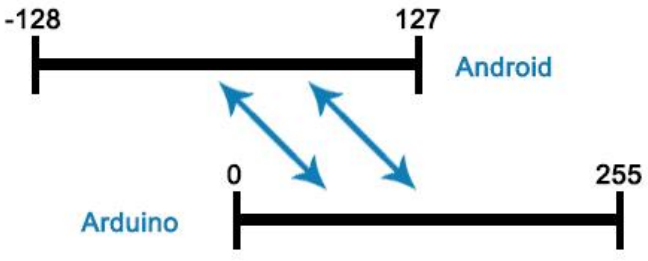


Figura 8-12: Problema de formatos entre Arduino y Android

De forma análoga, los valores que recibirá el dispositivo Android serán los PWM de los motores del robot. En Arduino estos valores oscilarán entre 0 y 255, por lo que de nuevo hay que procesar los datos para que ambas plataformas se entiendan.

Una vez que los datos son procesados se procede al envío y recepción de datos. Para el envío de datos se emplea la variable *ons*, que es de tipo *OutputStream*, y la función *write* de la siguiente forma:

```
ons.write(valor,0,valor.length);
```

Así, se escribe en el puerto serie los datos que hay en la variable valor (vector de dos componentes que almacena los valores de los acelerómetros filtrados), desde la posición 0 del vector y se le indica la longitud total que tiene que enviar[45].

Análogamente, para recibir datos del robot se emplea la variable *ins*, que es de tipo *InputStream*, y la función *read* de la siguiente forma:

```
while(ins.available() > 0){
      ins.read(ardu,0,ardu.length);
}
```

De esta forma, lee cuántos bytes hay en el *InputStream* y los coloca en el vector *ardu* (vector de cuatro componentes que almacena los PWM y los signos de los motores del robot), desde la posición 0 y hasta la longitud total del vector ardu.

8.1.7. REPRESENTACIÓN GRÁFICA

Lo último que queda es la representación gráfica de los resultados en la pantalla del dispositivo Android. Como se especificó en el punto 8.1.2 toda la representación gráfica se lleva a cabo en relación a las dimensiones de la pantalla en la que se esté ejecutando la aplicación. Si se pusieran las medidas de los diferentes elementos en coordenadas absolutas, según el dispositivo, la representación gráfica sería defectuosa. En cambio, al poner todas las medidas en coordenadas relativas al ancho y a la altura de la pantalla del dispositivo todos los elementos están relacionados unos con otros.

Figura 8-13: Origen de coordenadas y dimensiones máximas en una pantalla de un dispositivo Android

En primer lugar se pensó qué se quería realizar. Por un lado se quería mostrar en un gráfico la información que envían los acelerómetros. Y por otro lado se quería mostrar en otro gráfico la información de los motores que se recibe del robot.

Para representar la información de los acelerómetros se pensó en emplear un círculo que representase dichos valores. Este círculo se moverá en relación a los valores de los acelerómetros dentro de unos rangos. Así, en segundo lugar se definió el fondo de la aplicación. Se pensó en una serie de círculos concéntricos que representasen la fuerza de los acelerómetros:

```
background(0);
strokeWeight(1);
fill(31,91,76);
stroke(92,214,166);
ellipse(width/2,height/2,(float)(0.98*height),(float) (0.98*height));
noFill();
for (i = 1; i*50 < l/2; i++){
  ellipse(width/2,height/2,100*i,100*i);
}
```

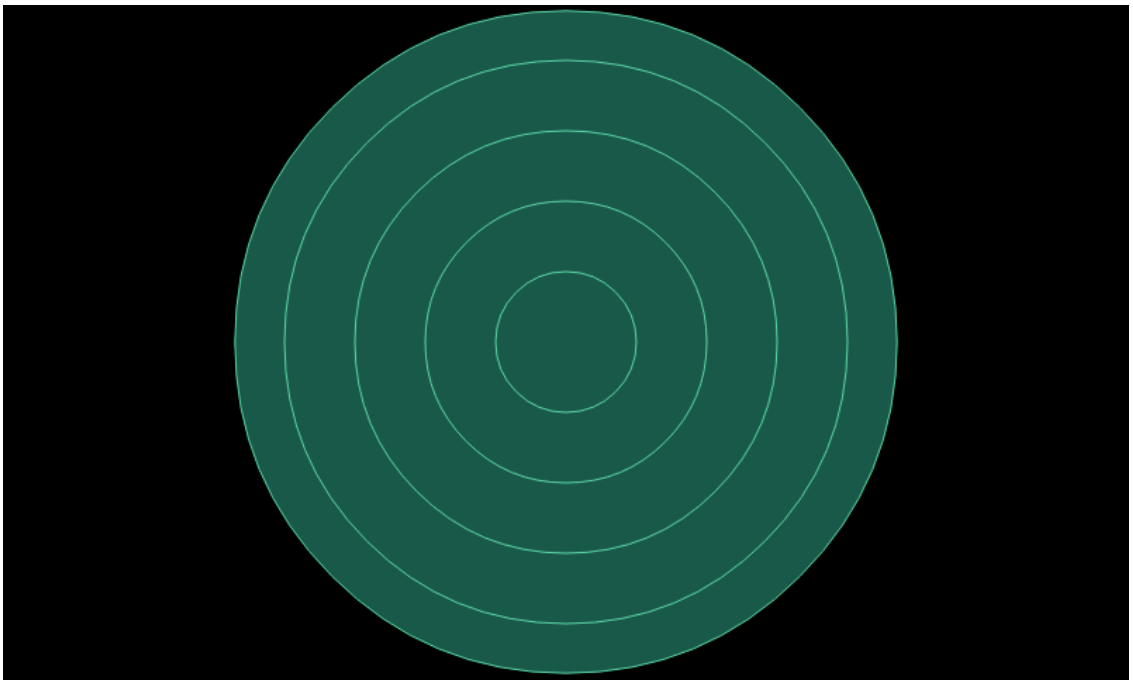


Figura 8-14: Fondo de la representación de los acelerómetros

En tercer lugar se desarrolló el movimiento del círculo. Este estaría relacionado con los acelerómetros. Además se pensó que para que fuera más visual la transparencia del círculo fuera variando también cuanto más se alejase de la situación de reposo. Como ya se ha explicado, al final resultó más complicado obtener las relaciones matemáticas que lo que es dibujar el círculo en sí:

```
strokeWeight(2);
stroke(2,52,77);
fill(0, 255, 255, 10 + sqrt(ax*ax+ay*ay)*245/15);
ellipse(constrain(map(ax,   -10,   10,   width/2-l/2+width/40,   width/2+l/2-width/40),
width/2-l/2+width/20, width/2+l/2-width/20),
constrain(map(ay,  -10,  10,  height/2-l/2+width/40,  height/2+l/2-width/40),  height/2-
l/2+width/20, height/2+l/2-width/20),width/20, width/20);
```

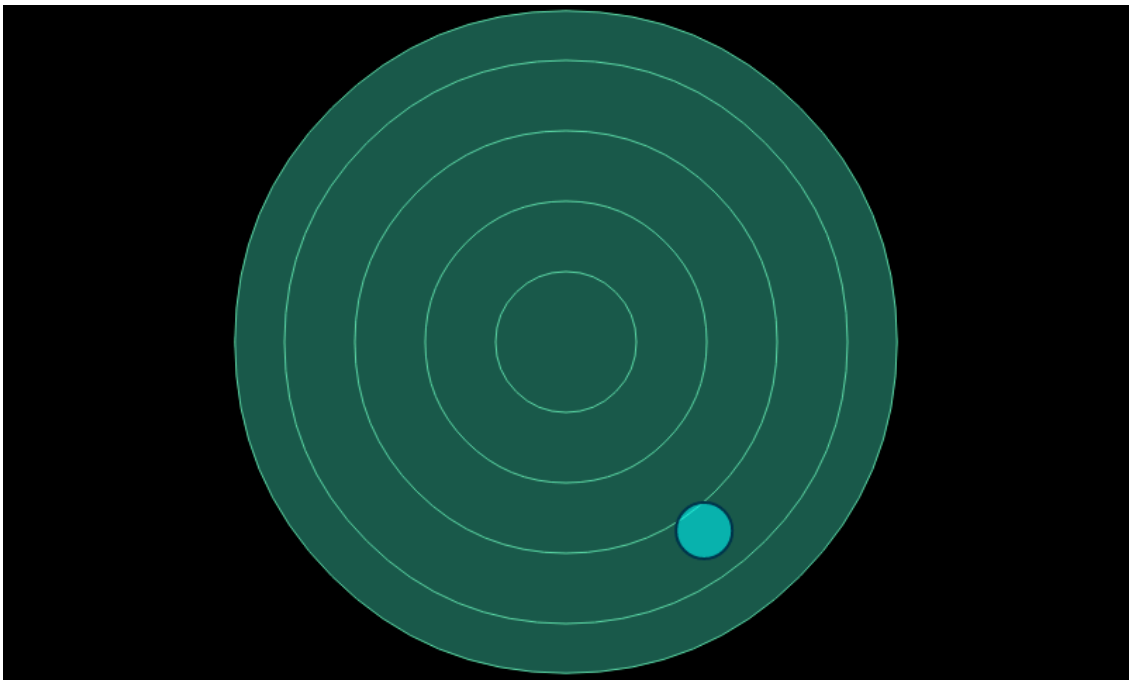


Figura 8-15: Representación de los acelerómetros. En este caso el robot iría marcha atrás y hacia la derecha

De esta forma se dio por concluida la representación de los acelerómetros. Para representar los PWM de los motores se pensó en emplear un gráfico de barras. Este representaría que los motores están girando marcha adelante si la barra es verde y que los motores están girando marcha atrás si la barra es roja. Para ello, se modificó el código realizado disponible en el C.I.r.E (Club de Informática, robótica y Electrónica)[46] y se obtuvo el resultado final de la representación gráfica:

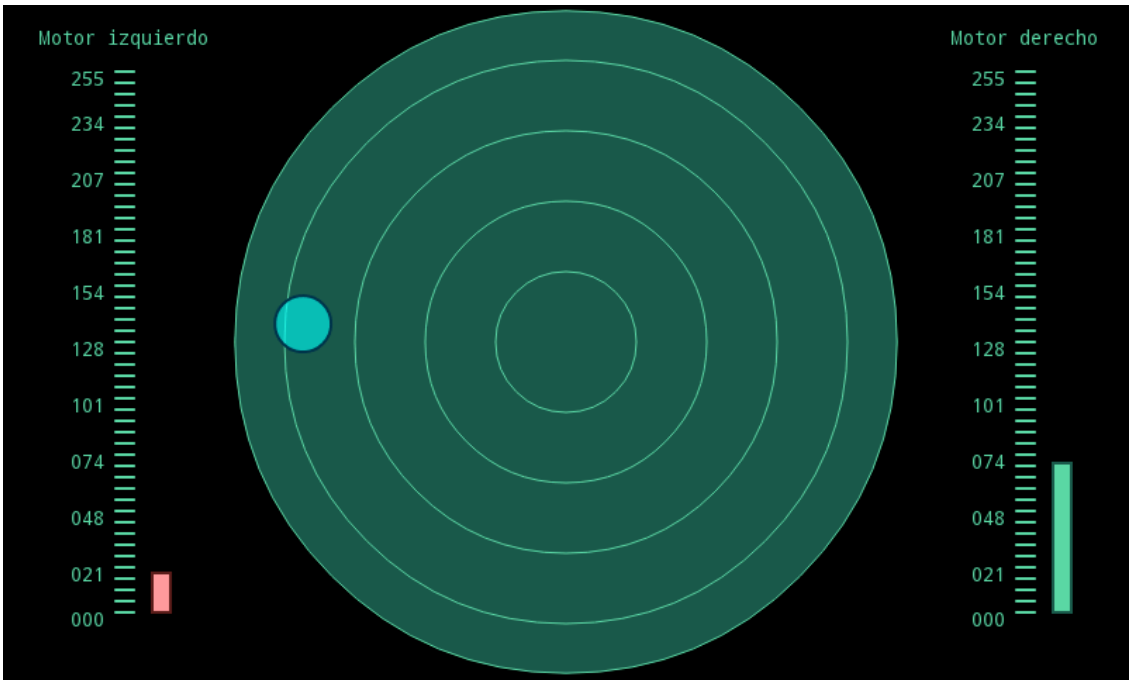


Figura 8-16: Interfaz final de la aplicación Android. En este caso el robot estaría trazando una circunferencia en sentido contrario a las agujas del reloj

8.1.8. NOMBRE DE LA APLICACIÓN E ICONO

Una vez terminada la interfaz gráfica se procedió a darle un nombre a la aplicación y un logotipo. Para ponerle nombre a la aplicación se debe crear un archivo con el nombre *strings.xml* dentro de la carpeta *values* que se encuentra dentro de la carpeta *res*.

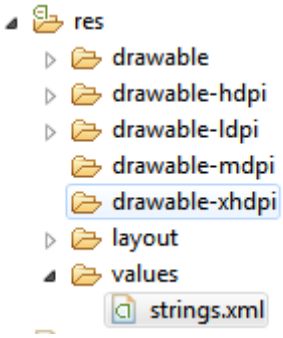


Figura 8-17: Lugar que ocupa el archivo strings.xml

Dentro de este archivo se deberá añadir una variable de tipo string cuyo nombre sea *app_name* y el nombre de la aplicación. En este caso se le va a llamar *Robot – Control Remoto*.

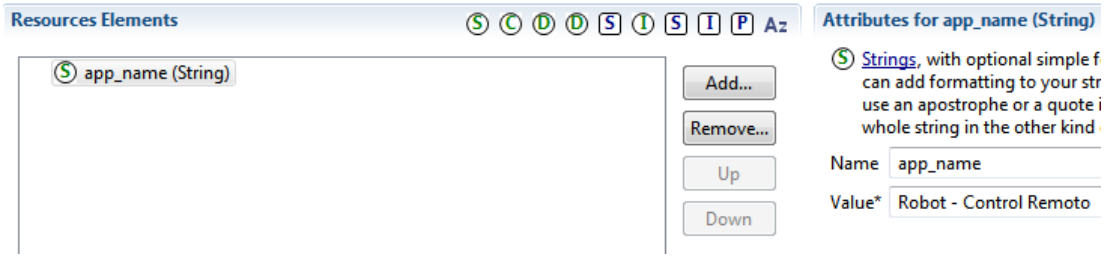


Figura 8-18: Nombre de la aplicación

Para esta aplicación se ha decidido emplear como logo el diagrama que representa a los acelerómetros. De esta forma, se debe guardar el logo que se haya creado dentro de la ruta res/drawable, res/drawable-hdpi y res\drawable-ldpi desde el explorador de archivos.

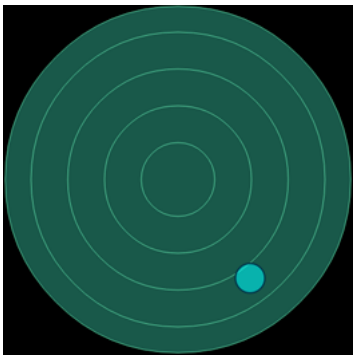


Figura 8-19: Logo de la aplicación Robot – Control Remoto

De esta forma se da por concluida la aplicación Android. En el apartado siguiente se explicará el código que emplea el robot.

8.2. PROGRAMACIÓN DEL ROBOT

Como todo *sketch*, en primer lugar se realiza la declaración de variables y la configuración del robot. Para poder manipular este robot hay dos tipos de variables. Por un lado, al igual que en el modo sigue líneas, están las variables de asignación de pines. Por otro lado están las variables que llevarán la velocidad a los motores. Para ello se declaran dos variables, una para que el robot avance y otra para que el robot gire.

En cuanto a la configuración, en primer lugar hay que configurar el robot para que envíe datos a 57600 baudios, obligado por el módulo bluetooth. En segundo lugar se

deben configurar los diferentes pines que controlan el puente en H como pines de salida y en tercer lugar preparar los motores. Para ello se ponen todas las entradas a nivel bajo y se activan los enables de ambos puentes. Además se apagan ambos LEDs delanteros del robot.

En los apartados siguientes se describirá cómo recibe y envía datos el robot.

8.2.1. RECEPCIÓN DE DATOS

Mediante la función *while* se estudia si se ha establecido la conexión bluetooth con el móvil. Si se ha establecido se realizarán los siguientes cálculos. En primer lugar se lee el puerto serie. Si se recuerda el apartado 8.1.6 los datos que llegan al robot están codificados. Esto es debido a que la codificación es diferente entre Android y Arduino. De esta forma lo primero que se hace es recoger los datos de los acelerómetros, descodificarlos y guardarlos en variables diferentes. Para la velocidad del robot se emplearán los datos del acelerómetro del eje x y para el giro se emplearán los datos del acelerómetro del eje y. Además, se lleva a cabo el filtro para evitar que los valores sobrepasen los límites ± 10.

```
giro = Serial.read() - 10;
vel = Serial.read() - 10;
filtro(vel, -10, 10);
filtro(giro, -10, 10);
```

Una vez obtenidos los valores se mapean para poder traducirlos en velocidades de los motores. Hay que recordar que estos están acotados desde 0 hasta 255. También hay que tener en cuenta que la dirección del acelerómetro del eje x es inversa con respecto al movimiento del robot. Esto quiere decir que un valor positivo de este acelerómetro corresponde con que el robot vaya marcha atrás. Para solucionar esto, la variable de la velocidad se mapeó con los límites cambiados Además, tras diversas pruebas, se ajustó para que el giro máximo fuera solo de un 20%. Giros mayores hacían que el robot se controlase peor.

```
if (vel >= 0){
  velm = map(vel, 0, 10, 0, -255);
}
else{
  velm = map(vel, -10, 0, 255, 0);
}
if (giro >= 0){
  girom = map(giro, 0, 10, 0, 50);
}
else{
  girom = map(giro, -10, 0, -50, 0);
}
```

Por último, tras haber obtenido las velocidades de ambos motores, se llevan estas a los motores. Hay que tener en cuenta que para que el robot gire adecuadamente los signos de giro deberán ser distintos en ambos motores.

```
motor(velm - girom, velm + girom);
```

8.2.2. ENVÍO DE DATOS

De nuevo para poder establecer una correcta comunicación entre Android y Arduino se deben codificar los datos que se quieren mandar. En este caso se quieren mandar las velocidades de ambos motores. Si se recuerda del apartado 8.1.6, en Android un byte va desde -128 hasta 127, mientras que en Arduino un byte va desde 0 hasta 255. De esta forma se enviará el dato que maneja cada motor -128. Así no habrá que preocuparse por incompatibilidades. Además se enviarán dos datos más, uno por cada motor, que indicarán si los motores están girando hacia adelante o hacía atrás.

```
Serial.write(byte(constrain(abs(velm-girom),0,vMax)-128));
  if (velm-girom >=0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
  Serial.write(byte(constrain(abs(velm+girom),0,vMax)-128));
  if (velm+girom >=0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
```

De esta forma se da por concluida la programación del robot.

CAPÍTULO 9. TELEMETRÍA

Tras terminar la aplicación Android que permite controlar el robot de forma remota, se pensó en realizar una aplicación que permitiera recoger datos en tiempo real del robot mientras se encuentra siguiendo la línea. De esta forma se podría comprobar el funcionamiento tanto de los sensores como de los motores.

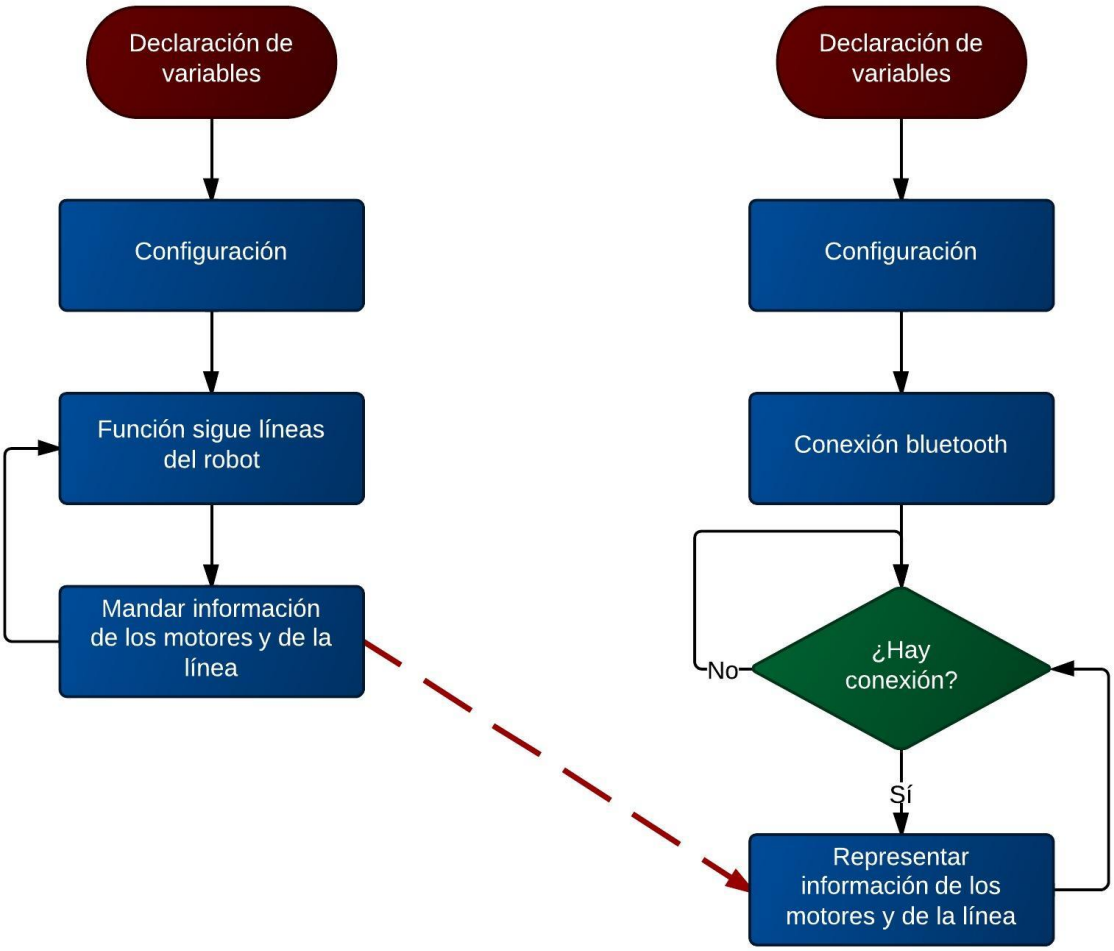


Figura 9-1: Diagrama del funcionamiento de la telemetría del robot. A la izquierda se representa el *sketch* del robot. A la derecha se representa la aplicación Android

El *sketch* que emplea el robot es el mismo que el que empleaba el robot para seguir la línea, pero se han añadido las siguientes líneas de código para enviar los datos al dispositivo Android, sin olvidar que hay que codificar los datos antes de enviarlos:

```
Serial.write(byte(constrain(abs(motorA),0,vMax)-128));
  if (motorA >= 0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
  Serial.write(byte(constrain(abs(motorB),0,vMax)-128));
  if (motorB >= 0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
  Serial.write(byte(posicion));
```

De esta forma se envían al dispositivo Android cinco valores: los PWM de los motores, los signos que tienen estos PWM y la posición relativa del robot con respecto a la línea.

Para la representación gráfica de los resultados se pensó en utilizar una interfaz similar a la que se emplea en la Fórmula1.

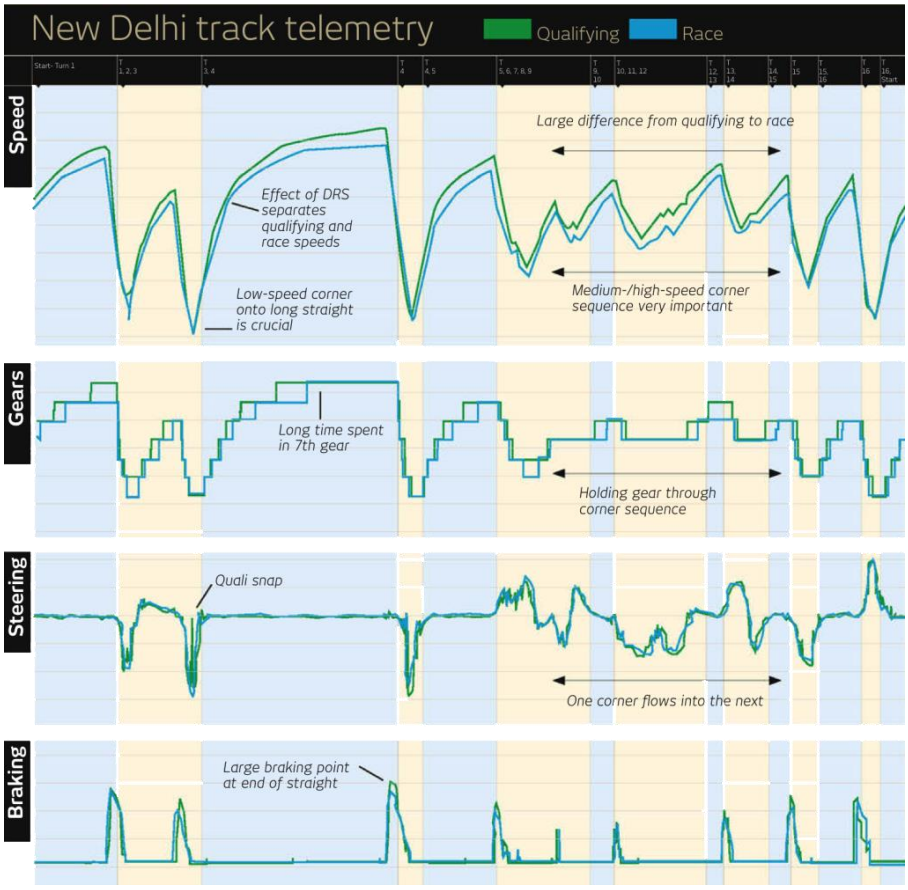


Figura 9-2: Telemetría de Bruno Senna (Williams-Renault) en el Gran Premio de la India 2012

Para realizar esta aplicación se empleó gran parte del código de la otra aplicación (ANEXO A.14). Para que los datos se representen de forma continua en la pantalla del dispositivo se van a emplear vectores de longitud igual al ancho de la pantalla. Después, en cada iteración se guardará el valor de X_i en X_{i-1}. En primer lugar se crean las variables donde se van a guardar los valores.

```
pwmA = new int[width];
pwmB = new int[width];
posicion = new int[width];
```

Después se guardan los valores en las variables.

```
if (ardu[1]==1){
            pwmA[i-1]=ardu[0]+128;
  }
  else if (ardu[1]==2){
            pwmA[i-1]=-1*(ardu[0]+128);
  }
  if (ardu[3]==1){
            pwmB[i-1]=ardu[2]+128;
  }
  else if (ardu[3]==2){
            pwmB[i-1]=-1*(ardu[2]+128);
  }
  posicion[i-1]=ardu[4];
```

Después se mapean los resultados para que queden las diferentes gráficas de una forma ordenada. De esta forma, el resultado final es el siguiente:

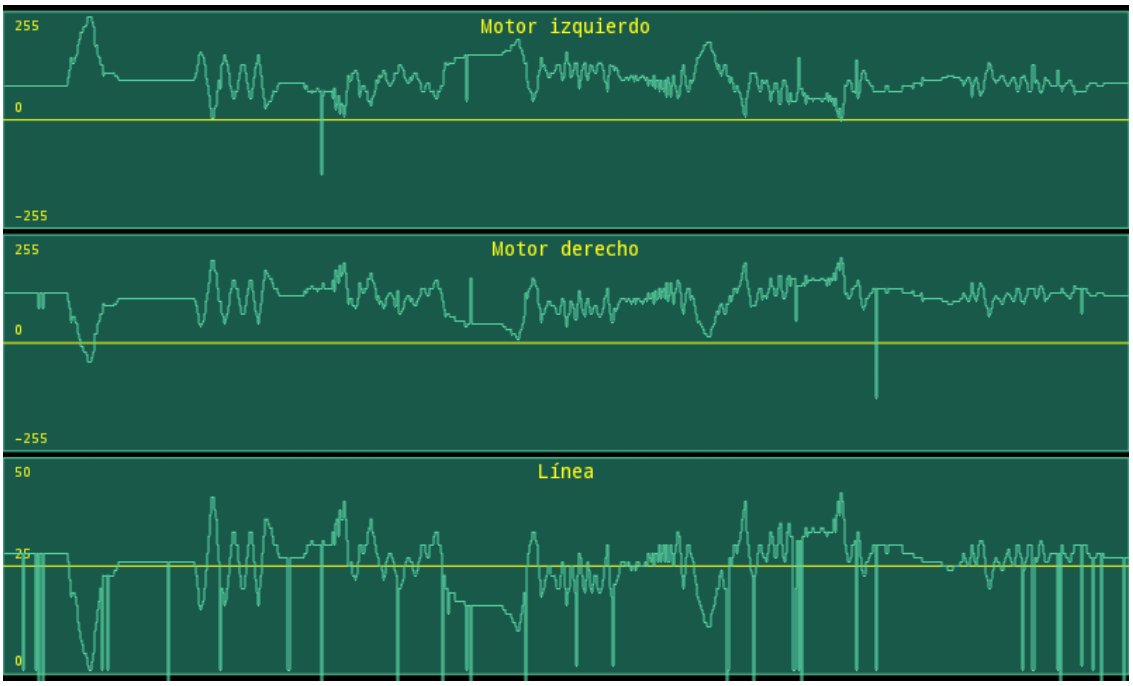


Figura 9-3: Interfaz de la aplicación de telemetría

CAPÍTULO 10. PRESUPUESTO

10.1. COSTE DE LA MANO DE OBRA

10.1.1. SALARIO

Ingenieros	Meses	Sueldo/mes	Total
1	6	1.705,00 €	10.230,00 €
		Total	10.230,00 €

Tabla 10.1: Salario

10.1.2. CARGAS SOCIALES

Concepto	Porcentaje
Contingencias Generales	28,30%
Desempleo, fondo de garantía salarial y formación profesional	8,90%
Accidentes	0,99%
Total	38,19%
38,19% del salario (10.230€)	3.906,84 €

Tabla 10.2: Cargas sociales

10.1.3. COSTE DE MANO DE OBRA

Concepto	Importe
Salario	10.230,00 €
Cargas sociales	3.906,84 €
Total	14.136,84 €

Tabla 10.3: Coste de mano de obra

10.2. COSTE DE MATERIALES

Microcontrolador	Unidades	Precio unitario	Total
Arduino Pro Mini	1	19,00 €	19,00 €
Conversor Serie - USB FTDI232	1	15,13 €	15,13 €
		Total	34,13 €
Ruedas	**Unidades**	**Precio unitario**	**Total**
Tamiya Ball Caster Kit	1	6,17 €	6,17 €
Rueda de goma 32x7mm (2 und)	1	7,20 €	7,20 €
		Total	13,37 €
Motores	**Unidades**	**Precio unitario**	**Total**
Motor Micro Metal DC con reductora 30:1	2	15,97 €	31,94 €
Funda protectora para motor micro metal (2 und)	1	5,08 €	5,08 €
		Total	37,02 €
Controlador motores	**Unidades**	**Precio unitario**	**Total**
L6205N	1	14,06 €	14,06 €
		Total	14,06 €
Sensores infrarrojos	**Unidades**	**Precio unitario**	**Total**
Array de sensores infrarrojos QTRC-8RC	1	14,40 €	14,40 €
		Total	14,40 €
Comunicación	**Unidades**	**Precio unitario**	**Total**
Módulo bluetooth JY-MCU	1	14,05 €	14,05 €
		Total	14,05 €
Alimentación	**Unidades**	**Precio unitario**	**Total**
Pilas 9V recargable	2	7,08 €	14,16 €
Cargador	1	20,20 €	20,20 €
		Total	34,36 €
Componentes electrónicos	**Unidades**	**Precio unitario**	**Total**
Condensador poliester 120nF	2	0,04 €	0,08 €
Condensador poliester 220nF	1	0,10 €	0,10 €
Condensador cerámico 15nF	1	0,03 €	0,03 €
Condensador cerámico 6,8nF	2	0,02 €	0,04 €
Diodo 1N4148	2	0,01 €	0,02 €
Resistencia 100kΩ	2	0,01 €	0,02 €
Resistencia 120Ω	1	0,01 €	0,01 €
Leds	2	0,02 €	0,04 €
Interruptor deslizante de dos posiciones	1	0,04 €	0,04 €
		Total	0,38 €
Soporte	**Unidades**	**Precio unitario**	**Total**
PCB	0,0044	100,00 €	0,44 €
		Total	0,44 €
		Total	162,21 €

Tabla 10.4: Coste de materiales

10.3. GASTOS VARIOS

Concepto	Importe
Material de oficina, fotocopias, encuadernación, cinta aislante	75,00 €
Total	75,00 €

Tabla 10.5: Gastos varios

10.4. GASTOS GENERALES

Son los concernientes al equipamiento empleado para medidas y pruebas en concepto de amortización de los equipos y gastos de energía. Se han estimado como el 10% de los apartados 10.1, 10.2 y 10.3.

Concepto	Importe	Porcentaje	Importe
Coste de mano de obra	14.136,84 €	10%	1.413,68 €
Coste de materiales	162,21 €	10%	16,22 €
Gastos varios	75,00 €	10%	7,50 €
		Total	1.437,40 €

Tabla 10.6: Gastos generales

10.5. IMPORTE TOTAL DEL PROYECTO

Concepto	Importe
Coste de mano de obra	14.136,84 €
Coste de materiales	162,21 €
Gastos varios	75,00 €
Gastos generales	1.437,40 €
Subtotal	15.811,45 €
I.V.A. (21%)	3.320,40 €
Subtotal	19.131,86 €

Tabla 10.7: Importe total del proyecto

El importe total del proyecto realizado asciende a DIECINUEVE MIL CIENTO TREINTA Y UN EUROS Y OCHENTA Y SEIS CÉNTIMOS.

CAPÍTULO 11. PUNTOS FUERTES Y FUTURAS MEJORAS DEL ROBOT

A continuación se van a desarrollar posibles mejoras que se le pueden aplicar al robot para mejorar su funcionamiento. Para ello en primer lugar se analizarán los principales puntos fuertes del robot y después se comentarán las posibles mejoras que se le pueden realizar al diseño.

11.1. PUNTOS FUERTES DEL ROBOT

11.1.1. TAMAÑO

El primer punto fuerte del robot es su tamaño. Se ha conseguido que en 58x76mm integrar todo un sistema capaz de seguir líneas y de ser controlado a distancia mediante un dispositivo Android. Cuando se inició el diseño del robot se puso como uno de los objetivos a conseguir el hacerlo más pequeño que el Pololu 3Pi. Este abarca una superficie de 70cm^2, mientras que nuestro robot abarca 44cm^2, es decir, es un 37% más pequeño. Este logro ha sido posible gracias al estudio pormenorizado de cada uno de los componentes que forman el robot y al aprovechamiento máximo del espacio disponible.

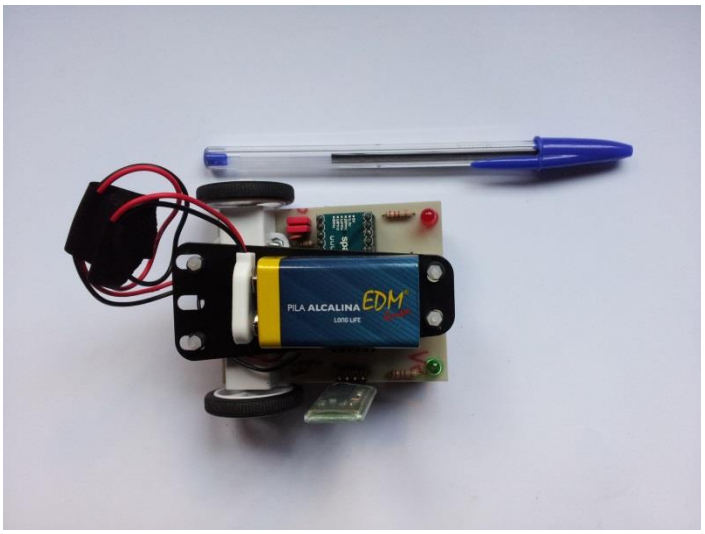


Figura 11-1: Comparativa de tamaño con un boli Bic

11.1.2. PESO

El segundo punto fuerte es su peso. Todo el conjunto pesa apenas 135g, frente a los 243g que pesa el Pololu 3Pi, incluidas las pilas. El hecho de que la mayoría de piezas sean de plástico ha facilitado lograr este objetivo que también se fijó al inicio del diseño del robot.

11.1.3. VELOCIDAD

El tercer punto fuerte es su velocidad. Haciendo pruebas en circuitos sin apenas curvas se han alcanzado velocidades de hasta 72cm/s, por lo que supera velocidad del Pocketbot 2 que se había puesto como objetivo, 70cm/s.

11.1.4. COMUNICACIÓN

El último punto fuerte del robot es el de la comunicación. Hasta la fecha pocos robots de estas características incluyen algún tipo de conexión inalámbrica. Además, el hecho de poder recoger datos y de controlarlo a distancia es un punto muy a tener en cuenta para proyectos similares que se realicen en el futuro.

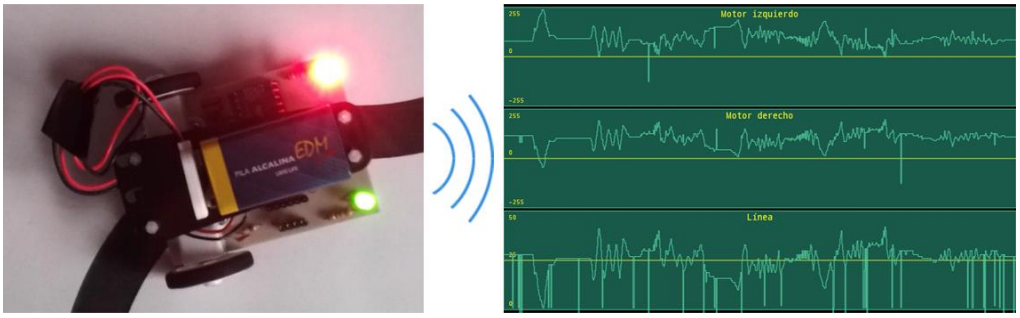


Figura 11-2: Comunicación en tiempo real

11.2. POSIBLES MEJORAS

11.2.1. DISPOSICIÓN DE LOS SENSORES

La mejora más evidente que se podría llevar a cabo es la de cambiar la disposición de los sensores infrarrojos. La disposición de matriz no es la más indicada para un robot rastreador. Sería mucho más acertado emplear una disposición en V invertida o de media luna ya que se podría detectar antes la presencia de una curva, por lo que el robot podría funcionar a una mayor velocidad. Además se podrían incluir otros dos sensores infrarrojos, uno a cada lado del robot, para detectar las marcas de bifurcación presentes en el circuito.

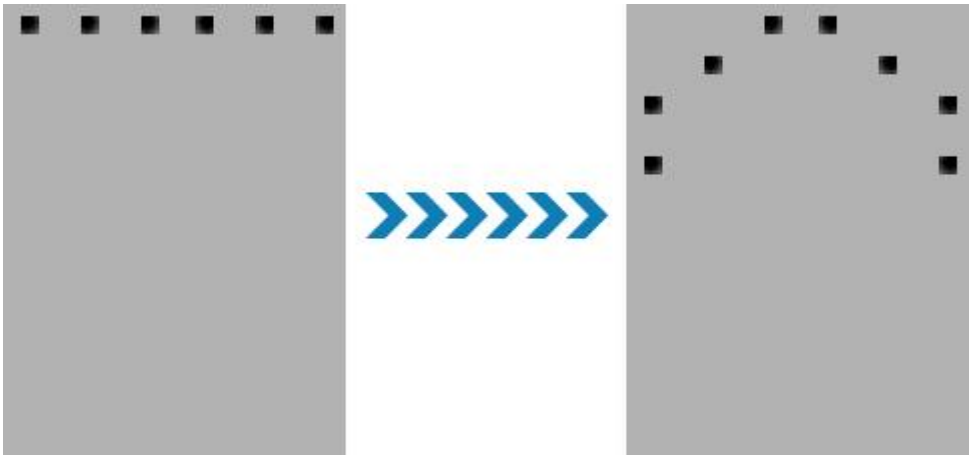


Figura 11-3: Cambio de disposición de los sensores

11.2.2. COSTE

Otro de los objetivos marcados al inicio del proyecto fue que el robot costara menos de los 100€ que cuesta el Pololu 3Pi. Este no incluye ni programador ni pilas, por lo que quitando esa parte del presupuesto nuestro robot costaría producirlo 112’28€.

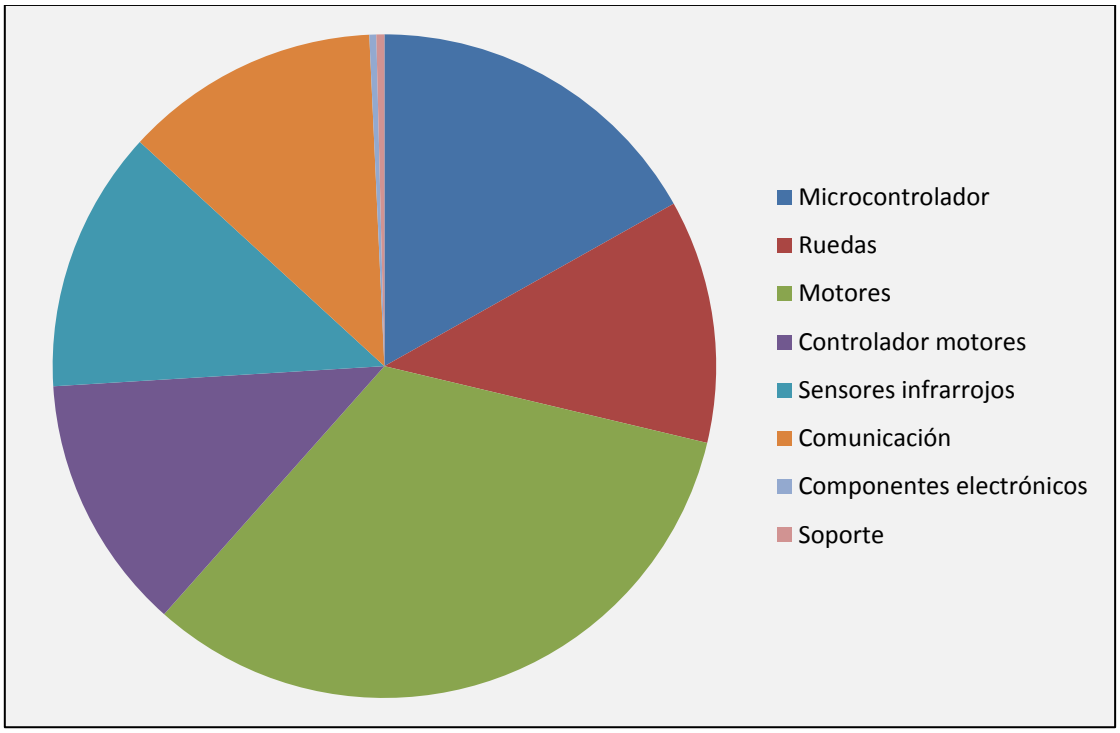


Figura 11-4: Representación de los costes según componentes quitando los componentes que no incluye el Pololu 3Pi

En el gráfico se puede ver que prácticamente un tercio del valor del robot es el coste asociado a los motores, por lo que podría ser interesante buscar otras opciones que ofrezcan las mismas características pero por un precio menor.

También se podría ahorrar algo de dinero en el controlador de los motores. Existen otras opciones, como los L293N, que por la mitad de dinero ofrecen las mismas características que el L6205N empleado en este robot.

11.2.3. CONEXIÓN MÓDULO BLUETOOTH

Tanto el módulo bluetooth como el programador emplean los mismos pines digitales para realizar sus funciones. Esto imposibilita que ambos dispositivos se encuentren conectados al mismo tiempo en el robot. De esta forma, cada vez que se tiene que programar el robot primero se debe de desconectar el módulo bluetooth y después se tiene que conectar el programador. Si se quiere volver a establecer de nuevo la conexión bluetooth se debe realizar el camino inverso, primero desconectar el programador y después conectar el módulo bluetooth. Esto hace que el proceso de

pruebas, como por ejemplo las pruebas realizadas para calibrar los parámetros PID, requieran un tiempo excesivo.

Para solventar este problema Arduino posee una librería llamada SoftwareSerial[9]. Por defecto la comunicación serie en un Arduino Pro Mini se realiza por los pines digitales 0 y 1. Esta librería permite que la comunicación serie se realice a través de otros pines. De esta forma es posible tener múltiples puertos series por software y no haría falta desconectar dispositivos.

11.2.4. PROGRAMA ANDROID

En la actualidad el robot ejecuta dos *sketch* distintos. Uno para seguir líneas y otro para que sea controlado por un dispositivo Android. Se podría desarrollar una aplicación Android que a través de un menú se seleccionara en qué modo debe funcionar el robot.

Figura 11-5: Posible interfaz de la aplicación

CAPÍTULO 12. CONCLUSIONES

El principal objetivo con el que se inició este proyecto fue el de profundizar en la programación de microcontroladores, ya que durante toda la carrera solamente se había tratado en una de ellas y resultó ser una de las asignaturas que más me gustaron. De esta forma, gracias a la robótica se pudo profundizar en la programación de microcontroladores de una forma entretenida e incluso, según el momento, divertida. Además, con la realización de este proyecto se han tratado diversos temas.

- Documentación: se ha tenido que leer multitud de hojas de características de componentes para conseguir que todos los elementos que formasen el robot fueran compatibles.
- Diseños electrónicos: se ha empleado software para el diseño de sistemas electrónicos tales como Eagle o DesignSpark que hasta antes de la realización del proyecto eran desconocidos para mí.
- Electrónica: se ha tenido que escoger elementos como resistencias y condensadores para que todo el sistema funcione de forma óptima.
- Sistemas de comunicación: tema totalmente desconocido para mí hasta antes de empezar a manipular el módulo bluetooth y los puertos serie tanto de los Arduino como de los ordenadores.
- Programación: se ha tenido que aprender a programar en dos lenguajes. Por un lado se ha aprendido a programar tarjetas Arduino y, por otro lado, se ha aprendido a programar aplicaciones para dispositivos Android.
- Control de sistemas: con la implantación del controlador PID se ha profundizado de una forma práctica en el control de sistemas.

Para concluir, espero que más estudiantes opten por realizar Proyectos Finales de Carrera apoyándose en la robótica y que este proyecto sirva para futuros proyectos.

BIBLIOGRAFÍA

RECURSOS BIBLIOGRÁFICOS

[1] Barbadillo Villanueva, Guillermo. Proyecto Arduino: Sumo robótico. Proyecto Final de Carrera. Universidad Pública de Navarra. Pamplona, 2012.

[2] Díaz, Pedro A. Artificial Intelligence In Vehicle. Proyecto Final de Carrera. Universitat Oberta de Catalunya. Barcelona, 2011.

[3] Evans, Brian W. Beginning Arduino Programming. Apress. 2011.

[4] Galbarra Goñi, David. Electrónica Digital y de Microprocesadores. Universidad Pública de Navarra. Pamplona, 2011.

[5] Galván Herrera, José B. Control Analógico de Sistemas Lineales. Ulzama digital. Pamplona, 2009.

[6] Iriarte Pastor, Aitor; Zuazu Ayesa, Carlos. Proyecto Arduino UPNA. Proyecto Final de Carrera. Universidad Pública de Navarra. Pamplona, 2011.

[7] Karvinen, Kimmo; Karvinen, Tero. Make: Arduino Bots and Gadgets. O'Reilly Media. 2011.

[8] Lera Carreras, Gabriel. Control Automático. Universidad Pública de Navarra. Pamplona, 2010.

[9] Margolis, Michael. Arduino Cookbook. O'Reilly Media. 2011.

[10] McRoberts, Michael. Beginning Arduino. Apress. 2011.

RECURSOS ELECTRÓNICOS

[11] Normativa AESSBOT 2012 Rastreadores:
http://aess.upc.es/aessbot/documentacion/normativa_aessbot2012_rastreadores_esp.pdf

[12] Guía de usuario del robot Pololu 3Pi:
http://www.pololu.com/file/0J137/Pololu3piRobotGuiaUsuario.pdf

[13] Precio Pololu 3Pi:
http://www.bricogeek.com/shop/robotica/106-seguidor-de-lineas-pololu-3pi.html

[14] Asociación de Robótica y Domótica de España. Proyecto n00b0t: http://foro.webdearde.com/viewtopic.php?f=6&t=3404&sid=d927e9812f6e45440f4b7bc2e682743a

[15] Blog n00b0t: http://n00b0t.tikitake.com/index-es.html

[16] Blog PocketBot 2: http://ostan.cz/PocketBot2/

[17] Robot Roomba: http://www.irobot.com/global/es/home.aspx

[18] Kiva Systems: http://www.kivasystems.com/

[19] Hospital Nemocnice Na Homolce: http://www.homolka.cz/en-CZ/home.html

[20] Características Pololu 3Pi: http://www.pololu.com/catalog/product/975

[21] Arduino: http://arduino.cc/

[22] Motores Paso a paso: http://en.wikipedia.org/wiki/Stepper_motor

[23] Control de motores con ULN2803: http://letsmakerobots.com/node/11772

[24] Hoja de características L6205N: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00002345.pdf

[25] Hoja de características QTR-8RC: http://www.pololu.com/docs/pdf/0J12/QTR-8x.pdf

[26] Shield SD: https://www.sparkfun.com/products/9802

[27] Módulo bluetooth JY-MCU: http://www.hobbycomponents.com/index.php?route=product/product&product_id=116

[28] Comandos AT: http://elecfreaks.com/store/download/datasheet/Bluetooth/HC-0305%20serail%20module%20AT%20commamd%20set%20201104%20revised.pdf

[29] Aplicación BlueTerm: https://play.google.com/store/apps/details?id=es.pymasde.blueterm&hl=es_419

[30] Eagle: http://www.cadsoftusa.com/

[31] DesignSpark: http://www.designspark.com/

[32] Conversor Serie-USB FTDI232: http://www.bricogeek.com/shop/295-conversor-serie-usb-ftdi232.html

[33] Conexión entre Arduino y Matlab: http://wechoosethemoon.es/2011/07/15/arduino-matlab-adquisicion-de-datos/

[34] Librería QTRSensor: http://www.pololu.com/docs/0J19/all

[35] Basic4Android: http://www.basic4ppc.com/

[36] Amarino: http://www.amarino-toolkit.net/

[37] SDK Android: http://developer.android.com/sdk/index.html

[38] Processing: http://processing.org/

[39] SDK Android. Acelerómetros:
http://developer.android.com/reference/android/hardware/SensorManager.html

[40] SDK Android. Bluetooth:
http://developer.android.com/guide/topics/connectivity/bluetooth.html

[41] UUID bluetooth: http://hykrion.com/?q=node/136

[42] SDK Android. Valores acelerómetros:
http://developer.android.com/reference/android/hardware/SensorEvent.html

[43] Byte en Java: http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Byte.html

[44] Byte en Arduino: http://arduino.cc/es/Reference/Byte

[45] Entrada / salida (streams):
http://www.lab.dit.upm.es/~lprg/material/apuntes/io/streams.htm

[46] Clase BarraInfo: http://webdelcire.com/wordpress/archives/408

ANEXO A: CÓDIGOS

ANEXO A.1: PRUEBA DE FUNCIONAMIENTO DEL L6205N

```
/*************************************************************
 *               Prueba funcionamiento L6205N                *
 *************************************************************/
int enA = 3;                // enable del Puente A
int enB = 4;                // enable del Puente B
int in1A = 5;               // entrada 1 del Puente A
int in2A = 6;               // entrada 2 del Puente A
int in1B = 10;              // entrada 1 del Puente B
int in2B = 11;              // entrada 2 del Puente B
int pwmA = 50;              // valor PWM del Puente A
int pwmB = 150;             // valor PWM del Puente B

void setup()  {
  // se declaran los pines como salidas:
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1A, OUTPUT);
  pinMode(in2A, OUTPUT);
  pinMode(in1B, OUTPUT);
  pinMode(in2B, OUTPUT);
}

void loop()  {
  digitalWrite(enA, HIGH);
  digitalWrite(enB, HIGH);
  analogWrite(pwmA, in1A);
  analogWrite(0, in2A);
  analogWrite(pwmB, in1B);
  analogWrite(0, in2B);
}
```

ANEXO A.2: PRUEBA DE FUNCIONAMIENTO DEL QTR-8RC

```
#include <QTRSensors.h>
/*
*************************************************************
*         Comprobación funcionamiento sensores IR           *
*************************************************************

 irS[0]  DIG17
 irS[1]  DIG18
 irS[2]  DIG16
 irS[3]  DIG15
 irS[4]  DIG14
 irS[5]  DIG9
 LEDON   DIG19


   irS[0]  irS[1]  irS[2]  irS[3]  irS[4]  irS[5]
    ###     ###     ###     ###     ###     ###
    ###     ###     ###     ###     ###     ###
*/

#define NUM_SENSORS   6               //número de sensores usados
#define TIMEOUT       2500            //2500 us para apagar cada sensor
#define LEDON         9               //el emisor se controla con el pin 19


//Configuración de pines
QTRSensorsRC qtrrc((unsigned char[]) {14, 15, 16, 17, 18, 19},
NUM_SENSORS, TIMEOUT, LEDON);          //configuración de pines y de tiempo para la
```

```
                                        //lectura de los sensores

//Configuración de los sensores IR
unsigned int irS[NUM_SENSORS];          //vector donde se guardan las lecturas de los
sensores
long irMax[] = {1, 1, 1, 1, 1, 1};      //vector para almacenar valores máximos
long irMin[] =
{5000, 5000, 5000, 5000, 5000, 5000};   //vector para almacenar valores mínimos
long irM[] = {0, 0, 0, 0, 0, 0};        //vector donde se guardan los valores mapeados

long tiempo;                            //variable para el tiempo

int i;

/*
===============
=    SETUP    =
===============
*/
void setup()
{
  Serial.begin(9600);

  pinMode(13, OUTPUT);

  //se espera de un segundo para comenzar la calibración
  delay(1000);

  //se enciende el LED 13 para indicar que se están calibrando los sensores
  digitalWrite(13, HIGH);

  //obtención de los valores mínimos y máximos de cada sensor
  tiempo = millis();
  while (millis() - tiempo < 4000)
  {
    qtrrc.read(irS);
    minmax(irS);
  }

  //se apaga el LED 13 para indicar que se ha terminado la calibración de los sensores
  digitalWrite(13, LOW);

  //se obtienen por pantalla los valores máximos y mínimos para cada sensor
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(irMax[i]);
    Serial.print(' ');
  }
  Serial.println(' ');
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(irMin[i]);
    Serial.print(' ');
  }
  Serial.println(' ');
}

/*
===============
=    LOOP     =
===============
*/
void loop()
{
  //lectura de los sensores y almacenamiento de sus valores
  qtrrc.read(irS);

  //se mapea cada valor según su máximo y su mínimo entre 0 y 9 para ver si sigue la
línea
  for (i = 0; i < NUM_SENSORS; i++)
  {
    irM[i] = (map(irS[i], irMin[i], irMax[i], 0, 9));
  }
  //filtro por si hay valores menores o mayores que los medidos durante la calibración
  for (i = 0; i < NUM_SENSORS; i++)
```

```
  {
    if (irM[i] < 0)
    {
      irM[i] = 0;
    }
    if (irM[i] > 9)
    {
      irM[i] = 9;
    }
  }

  /*se obtiene por pantalla el valor mapeado de cada sensor, para sabes si se detecta
  bien o no la línea.
  */
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(irM[i]);
    Serial.print(' ');
  }
  Serial.println(' ');

  delay(100);
}


/*
===============
=  FUNCIONES  =
===============
*/
void minmax(unsigned int val[NUM_SENSORS])    //función para detectar el máximo y el
mínimo de los sensores
{
    for (i = 0; i < NUM_SENSORS; i++)
    {
     if (val[i] > irMax[i])
     {
       irMax[i] = val[i];
     }
     if (val[i] < irMin[i])
     {
       irMin[i] = val[i];
     }
    }
}
```

ANEXO A.3: CONFIGURACIÓN DEL MÓDULO BLUETOOTH

```
/*****************************************************************
*        Cambio de la configuracióon del módulo bluetooth        *
*                  mediante comandos AT.                         *
*****************************************************************/

//Variable para control de configuración
boolean conf = false;
int espera = 1000;

void setup(){
   pinMode(13,OUTPUT);     // Pin13 para control del proceso
   Serial.begin(57600);    // Velocidad del modulo bluetooth
   digitalWrite(13, LOW);  // Apagamos el LED 13
}

void loop(){
  if (conf){
    // Parpadeo para verificar su funcionamiento
    digitalWrite(13, HIGH);   // Enciende el LED
    delay(espera);            // Espera
    digitalWrite(13, LOW);    // Apaga el LED
    delay(espera);            // Espera
  }
```

```
  else{
    digitalWrite(13, HIGH); // Empiezael tiempo de espera para reconectar
    Serial.println("Configuracion del modulo bluetooth");
    Serial.println("Conecte el TX del modulo BT al RX del arduino y el RX del modulo BT
al TX del arduino");
    delay(15000);           // Espera de 15 segundos (se puede cambiar)
    digitalWrite(13, LOW);  // Tiempo de espera agotado para reconectar

    Serial.print("AT");     // Empieza la comunicación por el modulo BT
    delay(espera);          // Espera para el envio de comandos

    Serial.print("AT+NAMEBTarduino"); // Nombre del dispositivo
    delay(espera);                    // Espera para el envio de comandos

    Serial.print("AT+BAUD7");   // Establecemos la velocidad en 57600
    delay(espera);              // Espera para el envio de comandos

    Serial.print("AT+PIN4242");   // Establecemos el pin de asociacion
    delay(espera);                // Espera para el envio de comandos

    //En este punto debe estar configurado
    digitalWrite(13, HIGH); // Todo ha funcionado segun lo esperado
    conf = true;     // Para no volver a configurarlo, salvo que se resetee
  }
}
```

ANEXO A.4: PRUEBA DE FUNCIONAMIENTO DEL MÓDULO BLUETOOTH

```
/*****************************************************
*               Prueba modulo bluetooth              *
*****************************************************/
void setup()
{
  //Se configura el pin 13 como salida para poder manipular el LED
   pinMode(13,OUTPUT);
   //Velocidad del módulo bluetooth con la que se haya configurado
   Serial.begin(57600);
}

void loop()
{
   //Se ejecutará el código mientras esté disponible el puerto Serie
   while (Serial.available())
   {
       //Variable enviada desde el móvil
       char dato= Serial.read();
       //Comprobación del dato
       switch(dato)
       {
         //Si se recibe una 'w' se enciende el LED 13 y se envía para mostrar en
Blueterm LED encendido
         case 'w':
         {
           digitalWrite(13,HIGH);
           Serial.println("LED encendido");
           break;
         }
         //Si se recibe una 'e' se apaga el LED 13 y se envía para mostrar en Blueterm
LED apagado
         case 'e':
         {
           digitalWrite(13,LOW);
           Serial.println("LED apagado");
           break;
         }
         //Si se recibe una 'r' se enciende y se apaga el LED y se muestrar en Blueterm
LED intermitente
         case 'r':
```

```
          {
            digitalWrite(13,HIGH);
            delay(200);
            digitalWrite(13,LOW);
            delay(200);
            digitalWrite(13,HIGH);
            delay(100);
            digitalWrite(13,LOW);
            delay(100);
            Serial.println("LED intermitente");
            break;
          }
        }
    }
}
```

ANEXO A.5: PRUEBA DE FUNCIONAMIENTO EN EL ROBOT DEL QTR-8RC

```
#include <QTRSensors.h>
/*
*************************************************************
*         Comprobación funcionamiento sensores IR           *
*************************************************************

 irS[0]  DIG17
 irS[1]  DIG18
 irS[2]  DIG16
 irS[3]  DIG15
 irS[4]  DIG14
 irS[5]  DIG9
 LEDON   DIG19


   irS[0]  irS[1]  irS[2]  irS[3]  irS[4]  irS[5]
    ###     ###     ###     ###     ###     ###
    ###     ###     ###     ###     ###     ###
*/

#define NUM_SENSORS   6                 //número de sensores usados
#define TIMEOUT       2500              //2500 us para apagar cada sensor
#define LEDON         19                //el emisor se controla con el pin 19


//Configuración de pines
QTRSensorsRC qtrrc((unsigned char[]) {12, 14, 11, 15, 18, 17},
NUM_SENSORS, TIMEOUT, LEDON);           //configuración de pines y de tiempo para la
                                        //lectura de los sensores

//Configuración de los sensores IR
unsigned int irS[NUM_SENSORS];          //vector donde se guardan las lecturas de los
sensores
long irMax[] = {1, 1, 1, 1, 1, 1};      //vector para almacenar valores máximos
long irMin[] =
{5000, 5000, 5000, 5000, 5000, 5000};   //vector para almacenar valores mínimos
long irM[] = {0, 0, 0, 0, 0, 0};        //vector donde se guardan los valores mapeados

long tiempo;                            //variable para el tiempo

int i;

/*
===============
=    SETUP    =
===============
*/
void setup()
{
  Serial.begin(9600);
```

```
  pinMode(13, OUTPUT);

  //se espera de un segundo para comenzar la calibración
  delay(1000);

  //se enciende el LED 13 para indicar que se están calibrando los sensores
  digitalWrite(13, HIGH);

  //obtención de los valores mínimos y máximos de cada sensor
  tiempo = millis();
  while (millis() - tiempo < 4000)
  {
    qtrrc.read(irS);
    minmax(irS);
  }

  //se apaga el LED 13 para indicar que se ha terminado la calibración de los sensores
  digitalWrite(13, LOW);

  //se obtienen por pantalla los valores máximos y mínimos para cada sensor
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(irMax[i]);
    Serial.print(' ');
  }
  Serial.println(' ');
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(irMin[i]);
    Serial.print(' ');
  }
  Serial.println(' ');
}

/*
===============
=    LOOP     =
===============
*/
void loop()
{
  //lectura de los sensores y almacenamiento de sus valores
  qtrrc.read(irS);

  //se mapea cada valor según su máximo y su mínimo entre 0 y 9 para ver si sigue la
línea
  for (i = 0; i < NUM_SENSORS; i++)
  {
    irM[i] = (map(irS[i], irMin[i], irMax[i], 0, 9));
  }
  //filtro por si hay valores menores o mayores que los medidos durante la calibración
  for (i = 0; i < NUM_SENSORS; i++)
  {
    if (irM[i] < 0)
    {
      irM[i] = 0;
    }
    if (irM[i] > 9)
    {
      irM[i] = 9;
    }
  }

  /*se obtiene por pantalla el valor mapeado de cada sensor, para sabes si se detecta
  bien o no la línea.
  */
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(irM[i]);
    Serial.print(' ');
  }
  Serial.println(' ');

  delay(100);
```

```
}


/*
===============
=  FUNCIONES  =
===============
*/
void minmax(unsigned int val[NUM_SENSORS])    //función para detectar el máximo y el
mínimo de los sensores
{
    for (i = 0; i < NUM_SENSORS; i++)
    {
     if (val[i] > irMax[i])
     {
       irMax[i] = val[i];
     }
     if (val[i] < irMin[i])
     {
       irMin[i] = val[i];
     }
    }
}
```

ANEXO A.6: PRUEBA DE FUNCIONAMIENTO EN EL ROBOT DE LOS LEDS DELANTEROS Y DE LOS MOTORES

```
/*
************************************************************
*                  Prueba motores y LEDs                   *
************************************************************

enableA  DIG08          LEDi   DIG10
enableB  DIG07          LEDd   DIG16
in1A     DIG05
in2A     DIG03
in1B     DIG06
in2B     DIG09


 **                                        **
 **                                        **
LEDi                                      LEDd


       ==== OUTB1                OUTA1 ====
       |                                 |
 MB ===|                                 |=== MA
       |                                 |
       ==== OUTB2                OUTA2 ====

*/


int enableA = 8;                        //pin del enable del puente A
int enableB = 7;                        //pin del enable del puente B
int in1A = 5;                           //pin de la entrada 1 del puente A
int in2A = 3;                           //pin de la entrada 2 del puente A
int in1B = 6;                           //pin de la entrada 1 del puente B
int in2B = 9;                           //pin de la entrada 2 del puente B
int LEDi = 10;                          //pin del LED izquierdo
int LEDd = 16;                          //pin del LED derecho

int vMax = 255;                         //constante para limitar la velocidad de los
motores

int i;
/*
===============
```

```
=    SETUP    =
===============
*/
void setup(){
  Serial.begin(57600);
  //pinMode
  pinMode(in1A, OUTPUT);
  pinMode(in2A, OUTPUT);
  pinMode(in1B, OUTPUT);
  pinMode(in2B, OUTPUT);
  pinMode(enableA, OUTPUT);
  pinMode(enableB, OUTPUT);
  pinMode(LEDi, OUTPUT);
  pinMode(LEDd, OUTPUT);
  //se detienen ambos motores
  motor(0,0);
  //se ponen a cero todas las salidas
  digitalWrite(in1A, LOW);
  digitalWrite(in1B, LOW);
  digitalWrite(in2A, LOW);
  digitalWrite(in2B, LOW);
  //se activan ambos puentes
  digitalWrite(enableA, HIGH);
  digitalWrite(enableB, HIGH);
  //se apagan los dos LEDs
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, LOW);
  //espera de un segundo para poder posicionar el robot en el suelo
  delay(1000);
  //se hace una pequeña secuencia con los LEDs para indicar que ha iniciado el programa
  digitalWrite(LEDi, HIGH);
  digitalWrite(LEDd, LOW);
  delay(500);
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, HIGH);
  delay(500);
  digitalWrite(LEDi, HIGH);
  digitalWrite(LEDd, LOW);
  delay(500);
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, HIGH);
  delay(500);
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, LOW);
  delay(500);
}

void loop(){
  //se lleva la velocidad a los motores
  digitalWrite(LEDi, HIGH);
  digitalWrite(LEDd, HIGH);
  motor(200,200);
  delay(2000);

  digitalWrite(LEDi, HIGH);
  digitalWrite(LEDd, HIGH);
  motor(-200,-200);
  delay(2000);

  digitalWrite(LEDi, HIGH);
  digitalWrite(LEDd, LOW);
  motor(100,200);
  delay(1000);

  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, HIGH);
  motor(200,100);
  delay(1000);

  digitalWrite(LEDi, HIGH);
  digitalWrite(LEDd, LOW);
  motor(-100,100);
  delay(1000);

  digitalWrite(LEDi, LOW);
```

```
  digitalWrite(LEDd, HIGH);
  motor(100,-100);
  delay(1000);
}

/*
*********************************************************
*      Función para mover los motores según el PWM      *
*********************************************************
*/
void motor(int out1, int out2){
  if (out1 >= 0){
    analogWrite(in1A, constrain(abs(out1), 0, vMax));
    analogWrite(in2A, 0);
  }
  else{
    analogWrite(in1A, 0);
    analogWrite(in2A, constrain(abs(out1), 0, vMax));
  }
  if (out2 >= 0){
    analogWrite(in1B, constrain(abs(out2), 0, vMax));
    analogWrite(in2B, 0);
  }
  else{
    analogWrite(in1B, 0);
    analogWrite(in2B, constrain(abs(out2), 0, vMax));
  }
}
```

ANEXO A.7: PRUEBA DE LA CONEXIÓN BLUETOOTH ENTRE ROBOT Y ORDENADOR. ARDUINO

```
#include <QTRSensors.h>
/*
*********************************************************
*                    Prueba bluetooth pc                *
*********************************************************

 irS[0]  DIG17
 irS[1]  DIG18
 irS[2]  DIG15
 irS[3]  DIG11
 irS[4]  DIG14
 irS[5]  DIG12
 LEDON   DIG19


   irS[0]  irS[1]  irS[2]  irS[3]  irS[4]  irS[5]
    ###     ###     ###     ###     ###     ###
    ###     ###     ###     ###     ###     ###
*/

#define NUM_SENSORS   6                         //número de sensores usados
#define TIMEOUT       2500                      //espera 2500 us para apagar cada
sensor
#define LEDON         19                        //el emisor se controla con el pin 19


//Configuración de pines

QTRSensorsRC qtrrc((unsigned char[]) {12, 14, 11, 15, 18, 17},
NUM_SENSORS, TIMEOUT, LEDON);                   //configuración de pines y de tiempo
para la lectura de los sensores


//Configuración de los sensores IR

unsigned int irS[NUM_SENSORS];                  //vector donde se guardan las lecturas
de los sensores
```

```
long irMax[] =
{2500, 2220, 2434, 2500, 2500, 2500};             //vector para almacenar valores máximos
long irMin[] = {326, 160, 268, 268, 228, 304};    //vector para almacenar valores mínimos

long irM[] = {0, 0, 0, 0, 0, 0};                  //vector donde se guardan los valores
mapeados

int dt = 10;

long tiempo = 0;

int i;

/*
===============
=    SETUP    =
===============
*/
void setup(){
  Serial.begin(57600);                            //obligado por el BT

  delay(1000);
}

/*
===============
=    LOOP     =
===============
*/
void loop(){
  //lectura de los sensores y almacenamiento de sus valores
  qtrrc.read(irS);

  //mapeamos cada valor según su máximo y su mínimo entre 0 y 9 para ver si sigue la
línea
  for (i = 0; i < NUM_SENSORS; i++){
    irM[i] = (map(irS[i], irMin[i], irMax[i], 0, 9));
  }

  //filtro por si hay valores menores o mayores que los medidos durante la calibración
  for (i = 0; i < NUM_SENSORS; i++){
    if (irM[i] < 0){
      irM[i] = 0;
    }
    if (irM[i] > 9){
      irM[i] = 9;
    }
  }

  //para obtener resultados
  for (i = 0; i < NUM_SENSORS; i++){
    Serial.print(irM[i]);
    Serial.print(' ');
  }
  Serial.println(' ');

  //se incluye un retraso para que matlab pueda procesar los datos
  delay(dt);

}
```

ANEXO A.8: PRUEBA DE LA CONEXIÓN BLUETOOTH ENTRE ROBOT Y ORDENADOR. MATLAB

```
clear all;
clc
%borrar previos
delete(instrfind({'Port'},{'COM6'}));
```

```
%crear objeto serie
s = serial('COM6','BaudRate',57600,'Terminator','CR/LF');
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%abrir puerto
fopen(s);

%Tiempo que dura la medición en segundos
tmax = 20;

%Inicializar
t = 0;

%Ejecutar bucle cronometrado
tic
%Con tic y toc mide el tiempo que hay entre ellos
while t<tmax
    t = toc;
    %Leer del puerto serie
    puerto = fscanf(s,'%d %d %d %d %d %d')';
    %Dibujar el diagrama de barras
    bar(puerto);
    %Actualizar el diagrama
    drawnow
end
clc;

%% Cerrar y eliminar el puerto creado
fclose(s);
delete(s);
clear s;
```

ANEXO A.9: PRUEBA DE LA CONEXIÓN BLUETOOTH ENTRE ROBOT Y DISPOSITIVO ANDROID

```
/*****************************************************
*               Prueba modulo bluetooh               *
*****************************************************/
int LEDi = 10;
int LEDd = 16;

void setup(){
  //Se configuran los pines de los LEDs como salida para poder manipularlos
   pinMode(LEDi, OUTPUT);
   pinMode(LEDd, OUTPUT);
   //Velocidad del módulo bluetooth con la que se haya configurado
   Serial.begin(57600);
   //se apagan ambos LEDs
   digitalWrite(LEDi, LOW);
   digitalWrite(LEDd, LOW);
}

void loop(){
   //Se ejecutará el código mientras esté disponible el puerto Serie
   while (Serial.available()){
       //Variable enviada desde el móvil
       char dato= Serial.read();
       //Comprobación del dato
       switch(dato){
         //Si se recibe una 'a' se apagan los dos LEDs y se envía para mostrar en
Blueterm LEDs apagados
         case 'a':{
           digitalWrite(LEDi, LOW);
           digitalWrite(LEDd, LOW);
           Serial.println("LEDs apagados");
           break;
         }
         //Si se recibe una 'r' se enciende el LED izquierdo y se envía para mostrar en
Blueterm LED izquierdo
         case 'r':{
```

```
            digitalWrite(LEDi, HIGH);
            digitalWrite(LEDd, LOW);
            Serial.println("LED izquierdo");
            break;
          }
          //Si se recibe una 'v' se enciende el LED derecho y se muestrar en Blueterm LED
derecho
          case 'v':{
            digitalWrite(LEDi, LOW);
            digitalWrite(LEDd, HIGH);
            Serial.println("LED derecho");
            break;
          }
          //Si se recibe una 'd' se encienden los dos LEDs y se muestrar en Blueterm LEDs
encendidos
          case 'd':{
            digitalWrite(LEDi, HIGH);
            digitalWrite(LEDd, HIGH);
            Serial.println("LEDs encendidos");
            break;
          }
         //Si se recibe una 'v' se enciende el LED derecho y se muestrar en Blueterm LED
derecho
         case 'i':{
           digitalWrite(LEDi, HIGH);
           digitalWrite(LEDd, LOW);
           delay(100);
           digitalWrite(LEDi, LOW);
           digitalWrite(LEDd, HIGH);
           delay(100);
           digitalWrite(LEDi, HIGH);
           digitalWrite(LEDd, LOW);
           delay(100);
           digitalWrite(LEDi, LOW);
           digitalWrite(LEDd, HIGH);
           delay(100);
           digitalWrite(LEDi, HIGH);
           digitalWrite(LEDd, LOW);
           delay(100);
           digitalWrite(LEDi, LOW);
           digitalWrite(LEDd, HIGH);
           delay(100);
           digitalWrite(LEDi, HIGH);
           digitalWrite(LEDd, LOW);
           delay(100);
           digitalWrite(LEDi, LOW);
           digitalWrite(LEDd, HIGH);
           delay(100);
           Serial.println("LEDs intermitentes");
         }
        }
    }
}
```

ANEXO A.10: FUNCIONAMIENTO COMO ROBOT SIGUE LÍNEAS

```
#include <QTRSensors.h>
/*
************************************************************
*                    Seguidor de línea                     *
************************************************************

 irS[0]   DIG17          enableA  DIG08          LEDi    DIG10
 irS[1]   DIG18          enableB  DIG07          LEDd    DIG16
 irS[2]   DIG15          in1A     DIG05
 irS[3]   DIG11          in2A     DIG03
 irS[4]   DIG14          in1B     DIG06
 irS[5]   DIG12          in2B     DIG09
 LEDON    DIG19

  **                                            **
```

```
  **                                          **
 LEDi                                        LEDd

   irS[0]  irS[1]  irS[2]  irS[3]  irS[4]  irS[5]
    ###     ###     ###     ###     ###     ###
    ###     ###     ###     ###     ###     ###


       ==== OUTB1                OUTA1 ====
       |                                  |
 MB ===|                                  |=== MA
       |                                  |
       ==== OUTB2                OUTA2 ====


 Posición del sensor i = 10 * i
 Valor a seguir = 25
 ____________________________________________________________
 Posición     50   40   30   20   10   0
 Error       -25  -15   -5    5   15   25   */

#define NUM_SENSORS   6                 //número de sensores usados
#define TIMEOUT       2500              //espera 2500 us para apagar cada sensor
#define LEDON         19                //el emisor se controla con el pin 19


//Asignación de pines

QTRSensorsRC qtrrc((unsigned char[]) {12, 14, 11, 15, 18, 17},
NUM_SENSORS, TIMEOUT, LEDON);           //configuración de pines y de tiempo para la
lectura de los sensores
int enableA = 8;                        //pin del enable del puente A
int enableB = 7;                        //pin del enable del puente B
int in1A = 5;                           //pin de la entrada 1 del puente A
int in2A = 3;                           //pin de la entrada 2 del puente A
int in1B = 6;                           //pin de la entrada 1 del puente B
int in2B = 9;                           //pin de la entrada 2 del puente B
int LEDi = 10;                          //pin del LED izquierdo
int LEDd = 16;                          //pin del LED derecho

//Variables de los sensores IR

unsigned int irS[NUM_SENSORS];          //vector donde se guardan las lecturas de los
sensores
long irP[] = {50, 40, 30, 20, 10, 0};   //vector de pesos
long irMax[] =
{1, 1, 1, 1, 1, 1};                     //vector para almacenar valores máximos
long irMin[] =
{5000, 5000, 5000, 5000, 5000, 5000};   //vector para almacenar valores mínimos
long irM[] = {0, 0, 0, 0, 0, 0};        //vector donde se guardan los valores mapeados


//Parámetros del PID

int kc = 10;                             //parámetros obtenidos experimentalmente
float tc = 0.18;

//pid
float kp = 0.6 * kc;                    //constante de proporcionalidad
float kd = kp * tc / 8;                 //constante de derivación
float ki = 2 * kp / tc;                 //constante de integración

float integ = 0;                        //parte integral del PID
float deriv = 0;                        //parte derivada del PID
long errora = 0;                        //variable donde se guarda el error anterior
float toterror = 0;                     //variable donde se guarda la suma total del
error
/*
float kp = 7;
float kd = 0;
float ki = 15;
*/
//Variables para el cálculo de la posición

int seg = 25;                           //constante con el valor a seguir
```

```
long posicion = 0;                          //variable donde se guarda la posición
long cont = 0;                              //variable para calcular la posición
long error = 0;                             //variable donde se guarda el error
char out = 'C';                             //variable para ver si se ha salido. C =
Centrado; I = Izqda; D = Drcha
char bif = 'C';                             //variable para las bifucarciones. C =
Centrado; I = Izqda; D = Drcha
char bifa = 'C';                            //variable para la bifurcación anterior.
int ven = 0;                                //variable para la cantidad de sensores que ven
línea
int vena = 0;                               //variable para la cantidad de sensores en el
momento anterior

//Variables para el movimiento de los motores

int motorA = 0;                             //velocidad controlada del motor A
int motorB = 0;                             //velocidad controlada del motor B
int velr = 100;                             //velocidad sin errores rápida
int vell = 50;                              //velocidad sin errores lenta
int vel = velr;                             //velocidad sin errores
int vMax = 255;                             //constante para limitar la velocidad de los
motores

//Variables de tiempo

int dt = 20;                                //delay introducido. En ms, por lo que habrá
que pasarlo a segundos
unsigned long tiempo;                       //variable de tiempo

//LEDs

int brillo = 0;                             //encendido de los LEDs delanteros

int i;

/*
===============
=    SETUP    =
===============
*/
void setup(){
  Serial.begin(57600);                      //obligado por el BT
  //pinMode
  pinMode(in1A, OUTPUT);
  pinMode(in2A, OUTPUT);
  pinMode(in1B, OUTPUT);
  pinMode(in2B, OUTPUT);
  pinMode(enableA, OUTPUT);
  pinMode(enableB, OUTPUT);
  pinMode(LEDi, OUTPUT);
  pinMode(LEDd, OUTPUT);
  //se detienen ambos motores
  motor(0,0);
  //se ponen a cero todas las salidas
  digitalWrite(in1A, LOW);
  digitalWrite(in1B, LOW);
  digitalWrite(in2A, LOW);
  digitalWrite(in2B, LOW);
  //se activan ambos puentes
  digitalWrite(enableA, HIGH);
  digitalWrite(enableB, HIGH);
  //se apagan los dos LEDs
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, LOW);
  //espera de un segundo para poder posicionar el siguelineas en el suelo
  delay(1000);
  calibracion();
  delay(1000);
}
/*
===============
=    LOOP     =
===============
*/
void loop(){
```

```
  //lectura de los sensores y almacenamiento de sus valores
  qtrrc.read(irS);
  //mapeamos cada valor según su máximo y su mínimo entre 0 y 9 para ver si sigue la
línea
  for (i = 0; i < NUM_SENSORS; i++){
    irM[i] = (map(irS[i], irMin[i], irMax[i], 0, 9));
  }
  //filtro por si hay valores menores o mayores que los medidos durante la calibración y
  //obtención del número de sensores que ven línea
  ven = 0;
  for (i = 0; i < NUM_SENSORS; i++){
    filtro(irM[i],0,9);
    if (irM[i] > 0){
      ven++;
    }
  }
  //si el robot se ha salido ningún sensor verá línea, por lo que ven = 0
  if (ven > 0){
    si_linea();
  }
  else{
    no_linea();
  }
  //se vuelcan en los motores las velocidades obtenidas
  motor(motorB, motorA);
  //LEDs delanteros
  /*
  if (motorA == motorB){
    digitalWrite(LEDi, HIGH);
    digitalWrite(LEDd, HIGH);
  }
  else if (motorA > motorB){
    digitalWrite(LEDi, LOW);
    digitalWrite(LEDd, HIGH);
  }
  else{
    digitalWrite(LEDi, HIGH);
    digitalWrite(LEDd, LOW);
  }*/

  if (bif == 'I'){
    digitalWrite(LEDi, HIGH);
    digitalWrite(LEDd, LOW);
  }
  else if (bif == 'D'){
    digitalWrite(LEDi, LOW);
    digitalWrite(LEDd, HIGH);
  }
  else{
    digitalWrite(LEDi, HIGH);
    digitalWrite(LEDd, HIGH);
  }
  //para obtener resultados
  resultados();
  //delay obligado por la transmisión de datos y del PID
  delay(dt);
}

/*
*********************************************************
*        Función para el cálculo de la posición         *
*********************************************************
-----------------------
Cálculo de la posición
-----------------------
Para calcular la posición del siguelineas se va a emplear la siguiente fórmula

irM[0] * irP[0] + irM[1] * irP[1] + irM[2] * irP[2] + irM[3] * irP[3] + irM[4] * irP[4]
+ irM[5] * irP[5]
-------------------------------------------------------------------------------------------
-----------------
                              irM[0] + irM[1] + irM[2] + irM[3] + irM[4] + irM[5]

De esta forma si la línea se encuentra sobre el sensor 2 se obtiene 20, si está entre el
 4 y el 5 se obtiene 45, etc...
```

```
*/
void calculo_posicion(){
  posicion = 0;
  cont = 0;
  for (i = 0; i < NUM_SENSORS; i++)
  {
    posicion = posicion + irM[i] * irP[i];
    cont = cont + irM[i];
  }
  posicion = posicion / cont;
}

/*
*********************************************************
*          Función para calibrar los sensores           *
*********************************************************
*/
void calibracion(){
  digitalWrite(LEDi, HIGH);
  digitalWrite(LEDd, HIGH);
  //obtención de los valores mínimos y máximos de cada sensor
  motor(50, -50);
  tiempo = millis();
  while (millis() - tiempo < 2000){
    qtrrc.read(irS);
    minmax(irS);
  }
  motor(0, 0);
  //terminada la calibración se coloca el siguelíneas sobre la línea
  motor(-25, 25);
  while (posicion > seg || posicion < seg)
  {
    qtrrc.read(irS);
    for (i = 0; i < NUM_SENSORS; i++)
    {
      irM[i] = map(irS[i], irMin[i], irMax[i], 0, 9);
    }
    for (i = 0; i < NUM_SENSORS; i++)
    {
      filtro(irM[i], 0, 9);
    }
    calculo_posicion();
    //LEDs
    if (brillo == 1){
      digitalWrite(LEDi, HIGH);
      digitalWrite(LEDd, HIGH);
      brillo = 0;
    }
    else{
      digitalWrite(LEDi, LOW);
      digitalWrite(LEDd, LOW);
      brillo = 1;
    }
  }
  motor(0, 0);
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, LOW);
}

/*
*********************************************************
* Función para asegurarnos que no se pasan los valores *
*********************************************************
*/
void filtro (int input, int inputmin, int inputmax){
  if (input < inputmin){
    input = inputmin;
  }
  if (input > inputmax){
    input = inputmax;
  }
}

/*
*********************************************************
```

```
*          Función para calcular los máximos y         *
*               mínimos de cada sensor                 *
*******************************************************
*/
void minmax(unsigned int val[NUM_SENSORS]){
  for (i = 0; i < NUM_SENSORS; i++){
    if (val[i] > irMax[i]){
      irMax[i] = val[i];
    }
    if (val[i] < irMin[i]){
      irMin[i] = val[i];
    }
  }
}

/*
*******************************************************
*      Función para mover los motores según el PWM     *
*******************************************************
*/
void motor(int out1, int out2){
  if (out1 >= 0){
    analogWrite(in1A, constrain(abs(out1), 0, vMax));
    analogWrite(in2A, 0);
  }
  else{
    analogWrite(in1A, 0);
    analogWrite(in2A, constrain(abs(out1), 0, vMax));
  }
  if (out2 >= 0){
    analogWrite(in1B, constrain(abs(out2), 0, vMax));
    analogWrite(in2B, 0);
  }
  else{
    analogWrite(in1B, 0);
    analogWrite(in2B, constrain(abs(out2), 0, vMax));
  }
}

/*
*******************************************************
*   Función por si el robot se ha salido del trazado   *
*******************************************************
*/
void no_linea(){
  switch(out){
    case 'I':                           //caso en el que la última vez que se vió línea
esta estaba a la izquierda
      error = -seg;                     //se fuerza a girar a la izquierda
      motorA = vMax;
      motorB = -vMax;
      break;
    case 'D':                           //caso en el que la última vez que se vió línea
esta estaba a la derecha
      error = seg;                      //se fuerza a girar a la derecha
      motorA = -vMax;
      motorB = vMax;
      break;
    case 'C':                           //caso en el que la última vez que se vió línea
esta estaba en el centro
      error = 0;                        //se fuerza a ir marcha atrás
      motorA = -vMax;
      motorB = -vMax;
      break;
    default:
      error = 0;
      break;
  }
}

/*
*******************************************************
*                          PID                        *
*******************************************************
----------------------
```

```
     Control PID
----------------------
Los signos con los que afectan el control PD a cada motor son distintos ya que para vari
ar la posición del siguelineas
lo que se pretende es qe este gire
*/
void pid(){
  integ = integ + error * dt / 1000;
  deriv = (error - errora) * 1000 / dt;       //el 1000 es el factor para pasar de ms a
segundos

  motorA = vel + error * kp + integ * ki + deriv * kd;
  motorB = vel - error * kp - integ * ki - deriv * kd;

  errora = error;                             //se almacena el error para poder aplicar
la parte derivativa
}

/*
*********************************************************
*   Función para mostrar los resultados por pantalla    *
*********************************************************
*/
void resultados(){
  /*for (i = 0; i < NUM_SENSORS; i++){
    Serial.print(irM[i]);
    Serial.print(' ');
  }*/
  //Serial.print(posicion);
  //Serial.print(' ');
  //Serial.println(seg);
  Serial.print(error);
  Serial.print(' ');
  Serial.print(0);
  Serial.print(' ');
  Serial.print(constrain(motorA,-vMax,vMax));
  Serial.print(' ');
  Serial.println(constrain(motorB,-vMax,vMax));
  //Serial.println(bif);
  /*Serial.write(byte(constrain(abs(motorA),0,vMax)-128));
  if (motorA >= 0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
  Serial.write(byte(constrain(abs(motorB),0,vMax)-128));
  if (motorB >= 0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
  Serial.write(byte(posicion));
  switch (bif){
    case 'I':
      Serial.write(byte(1));
      break;
    case 'C':
      Serial.write(byte(2));
      break;
    case 'D':
      Serial.write(byte(3));
      break;
  }*/
  /*Serial.write(byte(seg));
  for (i = 1; i < NUM_SENSORS; i++){
    Serial.write(byte(irM[i]));
  }*/
}

/*
*********************************************************
*     Función para cuando el robot siga el trazado      *
*********************************************************
```

```
*/
void si_linea(){
  //Marca de bifurcaciones
  //if ((((irM[0] && irM[2]) > 0) && (irM[1] == 0)) || (((irM[5] && irM[3]) > 0) &&
(irM[4] == 0)) || ((irM[0] && irM[2] && irM[3] && irM[5] > 0) && ((irM[1] == 0) &&
(irM[4] == 0))))
  if ((((irM[0] > irM[1]) && (irM[2] > irM[1])) || ((irM[5] > irM[4]) && (irM[3] >
irM[4]))) && (vena < 3))
  //if ((((irM[0] && irM[2]) > 0) && (irM[1] == 0)) || (((irM[5] && irM[3]) > 0) &&
(irM[4] == 0)))
  {
    si_marca();
  }
  else if ((((irM[0] > irM[1]) && (irM[2] > irM[1])) || ((irM[5] > irM[4]) && (irM[3] >
irM[4])))){  //evitar que lade cuando ve una marca de bifurcación
    irM[0] = 0;
    irM[1] = 0;
    irM[4] = 0;
    irM[5] = 0;
    calculo_posicion();
  }
  else{
    calculo_posicion();
  }
  //Actualizar las marcas de bifurcación
  if ((vena < 4) && (ven < 3)){
    bif = bifa;
  }
  //Bifurcaciones
  if ((ven > 3) && (bif == 'I') && (vena > 3)){
    seg = 0;      //obligamos a que gire a la izquierda
    vel = vell;   //se frena el coche
  }
  else if ((ven > 3) && (bif == 'D') && (vena > 3)){
    seg = 50;     //obligamos a que gire a la derecha
    vel = vell;   //se frena el coches
  }
  //seguir recto
  else{
    seg = 25;     //valor por defecto
    vel = velr;   //velocidad normal
  }
  //Cálculo del error
  error = seg - posicion;
  //PID
  pid();
  /*
  Se va guardando la última posición en la que se vio línea. Si se vio a la izquierda de
 la parte central querrá decir que el
  error es negativo y si por el contrario se vio a la derecha querrá decir que el error
es positivo.
  */
  if (error < 0){
    out = 'I';
  }
  else if (error > 0){
    out = 'D';
  }
  else{
    out = 'C';
  }
  vena = ven;
}

/*
*********************************************************
*      Función para detectar marcas de bifurcación      *
*********************************************************

   irS[0]  irS[1]  irS[2]  irS[3]  irS[4]  irS[5]
    ###     ###     ###     ###     ###     ###
    ###     ###     ###     ###     ###     ###

   |||||           ||||||||||||                        //Marca a la izquierda
   |||||           ||||||||||||
```

```
                    ||||||||||||||           |||||        //Marca a la derecha
                    ||||||||||||||           |||||

*/

void si_marca(){
  if ((irM[0] > irM[1]) && (irM[2] > irM[1])){
    bifa = 'I';
    posicion = seg;
  }
  else if ((irM[5] > irM[4]) && (irM[3] > irM[4])){
    bifa = 'D';
    posicion = seg;
  }
  else{
    bifa = 'C';
    posicion = seg;
  }
}
```

ANEXO A.11: OBTENCIÓN DE LOS PARÁMETROS DEL CONTROLADOR PID

```
clear;
clc;

%borrar previos
delete(instrfind({'Port'},{'COM6'}));
%crear objeto serie
s = serial('COM6','BaudRate',57600,'Terminator','CR/LF');
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%abrir puerto
fopen(s);

% parámetros de medidas
tmax = 15; % tiempo de captura en s
rate = 50; % resultado experimental

% preparar la figura
f = figure('Name','Captura');
a = axes('XLim',[0 tmax],'YLim',[-1.2 1.2]);
l1 = line(nan,nan,'Color','r','LineWidth',2);
l2 = line(nan,nan,'Color','b','LineWidth',1);
l3 = line(nan,nan,'Color','y','LineWidth',2);
l4 = line(nan,nan,'Color','g','LineWidth',2);

xlabel('Tiempo (s)')
ylabel('Error')
title('Captura del error en tiempo real con Arduino')
grid on
hold on

% inicializar
v1 = zeros(1,tmax*rate);
v2 = zeros(1,tmax*rate);
v3 = zeros(1,tmax*rate);
v4 = zeros(1,tmax*rate);
i = 1;
t = 0;

% ejecutar bucle cronometrado
tic
while t<tmax
    t = toc;
    % leer del puerto serie
    puerto = fscanf(s,'%f %f %f %f')';
    v1(i)=puerto(1)/25;
```

```
    v2(i)=puerto(2)/25;
    v3(i)=puerto(3);
    v4(i)=puerto(4);
    % dibujar en la figura
    x = linspace(0,i/rate,i);
    set(l1,'YData',v1(1:i),'XData',x);
    set(l2,'YData',v2(1:i),'XData',x);
    set(l3,'YData',v3(1:i),'XData',x);
    set(l4,'YData',v4(1:i),'XData',x);
    drawnow
    % seguir
    i = i+1;
end


%% Cerrar el puerto
fclose(s);
delete(s);
clear s;
```

ANEXO A.12: CONTROL REMOTO DEL ROBOT. ARDUINO

```
/*

************************************************************
*               Control robot desde Android                *
************************************************************

 enableA  DIG08          LEDi   DIG10
 enableB  DIG07          LEDd   DIG16
 in1A     DIG05
 in2A     DIG03
 in1B     DIG06
 in2B     DIG09

   **                                          **
  **                                          **
 LEDi                                        LEDd

   irS[0]  irS[1]  irS[2]  irS[3]  irS[4]  irS[5]
    ###     ###     ###     ###     ###     ###
    ###     ###     ###     ###     ###     ###


       ____ OUTB1               OUTA1 ___
       |                                |
 MB ===|                                |=== MA
       |                                |
       ____ OUTB2               OUTA2 ___


*/
//Configuración de pines

int enableA = 8;          //pin del enable del puente A
int enableB = 7;          //pin del enable del puente B
int in1A = 5;             //pin de la entrada 1 del puente A
int in2A = 3;             //pin de la entrada 2 del puente A
int in1B = 6;             //pin de la entrada 1 del puente B
int in2B = 9;             //pin de la entrada 2 del puente B
int LEDi = 10;            //pin del LED izquierdo
int LEDd = 16;            //pin del LED derecho


//Configuración para el movimiento de los motores

int vMax = 255;           //constante para limitar la velocidad de los motores
                          //se limita para que como mucho vayan al 90%
int i;
```

```
int giro = 0;             //variable donde se guardará el giro
int vel = 0;              //variable donde se guardará la velocidad
int girom = 0;            //giro que se lleva al motor
int velm = 0;             //velocidad que se lleva al motor

/*
===============
=    SETUP    =
===============
*/
void setup(){
  Serial.begin(57600);                  //obligado por el BT
  //pinMode
  pinMode(in1A, OUTPUT);
  pinMode(in2A, OUTPUT);
  pinMode(in1B, OUTPUT);
  pinMode(in2B, OUTPUT);
  pinMode(enableA, OUTPUT);
  pinMode(enableB, OUTPUT);
  pinMode(LEDi, OUTPUT);
  pinMode(LEDd, OUTPUT);
  //se detienen ambos motores
  motor(0,0);
  //se ponen a cero todas las salidas
  digitalWrite(in1A, LOW);
  digitalWrite(in1B, LOW);
  digitalWrite(in2A, LOW);
  digitalWrite(in2B, LOW);
  //se activan ambos puentes
  digitalWrite(enableA, HIGH);
  digitalWrite(enableB, HIGH);
  //se apagan los dos LEDs
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, LOW);
  //espera de un segundo para poder posicionar el siguelineas en el suelo
  delay(1000);
}

/*
===============
=    LOOP     =
===============
*/
void loop(){
  delay(20);
  digitalWrite(LEDi, LOW);
  digitalWrite(LEDd, LOW);
  //motor(0,0);
  //Mientras el puerto serie del modulo bluetooth esta disponible
   while (Serial.available()>0){
      //Guardamos en la variable dato el valor leido por el modulo bluetooth
       digitalWrite(LEDi, HIGH);
       digitalWrite(LEDd, HIGH);
       giro = Serial.read() - 10;  //el -10 es cosa de java. Fallaba el paso de numeros
negativos
       vel = Serial.read() - 10;
       //filtros
       filtro(vel, -10, 10);
       filtro(giro, -10, 10);

       //mapeados
       if (vel >= 0){  //hay que cambiar el sentido a vel
         velm = map(vel, 0, 10, 0, -255);
       }
       else{
         velm = map(vel, -10, 0, 255, 0);
       }
       if (giro >= 0){
         girom = map(giro, 0, 10, 0, 50);
       }
       else{
         girom = map(giro, -10, 0, -50, 0);
       }
       //motor
```

```
        motor(velm - girom, velm + girom);
    }
    enviar();
}


void enviar(){
  Serial.write(byte(constrain(abs(velm-girom),0,vMax)-128));
  if (velm-girom >=0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
  Serial.write(byte(constrain(abs(velm+girom),0,vMax)-128));
  if (velm+girom >=0){
    Serial.write(byte(1));
  }
  else{
    Serial.write(byte(2));
  }
}

void filtro (int input, int inputmin, int inputmax)
{
  if (input < inputmin)
  {
    input = inputmin;
  }
  if (input > inputmax)
  {
    input = inputmax;
  }
}

void motor(int out1, int out2)    //función para mover los motores según el pwm que se
desee
{
  if (out1 >= 0)
  {
    analogWrite(in1A, constrain(abs(out1), 0, vMax));
    analogWrite(in2A, 0);
  }
  else
  {
    analogWrite(in1A, 0);
    analogWrite(in2A, constrain(abs(out1), 0, vMax));
  }
  if (out2 >= 0)
  {
    analogWrite(in1B, constrain(abs(out2), 0, vMax));
    analogWrite(in2B, 0);
  }
  else
  {
    analogWrite(in1B, 0);
    analogWrite(in2B, constrain(abs(out2), 0, vMax));
  }
}
```

ANEXO A.13: CONTROL REMOTO DEL ROBOT. ANDROID

```
package processing.test.mandar_datos;

import processing.core.*;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
```

```
import android.content.Intent;
import java.util.ArrayList;
import java.util.UUID;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Method;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.hardware.SensorEventListener;
import android.os.Bundle;
import android.view.WindowManager;

public class mandar_datos extends PApplet {

//variables de las barras
BarraInfo pwmA;
BarraInfo pwmB;

//fuentes
String[] fontList;
PFont androidFont;

//variables de configuración del acelerometro
SensorManager mSensorManager;
MySensorEventListener accSensorEventListener;
Sensor acc_sensor;
float[] acc_values;             //matriz donde se guardarán los resultados

//variables de la comunicación bluetooth
private static final int REQUEST_ENABLE_BT = 3;
ArrayList<BluetoothDevice> dispositivos;//matriz con los dispositivos encontrados por el
teléfono
BluetoothAdapter bttelefono;                            //representa el bluetooth del teléfono
BluetoothDevice dispositivo;                            //representa el bluetooth de los
dispositivos ajenos al teléfono
BluetoothSocket socket;                                         //establece la conexión entre
teléfono y robot
InputStream ins;                                                //variable para recibir datos
del robot (byte)
OutputStream ons;                                               //variable para enviar
datos al robot (byte)

//variables del menú
int estado;
String error;

//variables para el cálculo de valor
byte[] valor={0,0};                                             //valor que se enviará
por bluetooth   {menu,giro,velocidad}
byte giro = 0;                                                  //variable del giro. El
punto medio será 4
byte vel = 0;                                                   //variable de la
velocidad. El punto medio será 5
float ant[] ={0,0};                                             //variable del valor del
acelerómetro anterior. Se calculará la media de las dos ultiams mediciones {vel,giro}
byte[] ardu = {0,0,0,0};                                //valor recibido por el robot

int i;

//variables del fondo
float l;

//variables del circulito
float ay;
float ax;
////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////
BroadcastReceiver receptor = new BroadcastReceiver(){           //un BroadcastReceiver sirve
para recibir eventos
```

```
    public void onReceive(Context context, Intent intent){
        String accion = intent.getAction();                                    //recibe el
evento que se envía a la aplicación
        if (BluetoothDevice.ACTION_FOUND.equals(accion)){       //la acción es que ha encontrado
un dispositivo bluetooth
            BluetoothDevice dispositivo =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            println(dispositivo.getName() + " " + dispositivo.getAddress());
            dispositivos.add(dispositivo);
        }
        else if (BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(accion)){     //el teléfono
está buscando disopositivos
          estado = 0;
        }
        else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(accion)){    //el teléfono ha
terminado de buscar
          estado = 1;
        }

    }
};
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void setup() {
  size(displayWidth, displayHeight, JAVA2D);                    //se obtienen las medidas de la
pantalla y se define que solo habrán

               //objetos en 2D
  frameRate(25);
        //refresco de pantalla

  //para que la pantalla esté girada
  orientation(LANDSCAPE);
  //si queremos que la posición de la pantalla sea vertical:
  //orientation(PORTRAIT);

  //fuentes
  fontList = PFont.list();
  androidFont = createFont(fontList[0], 32, true);
  textFont(androidFont);
  stroke(255);
  textAlign(CENTER);

  //distancia vertical máxima
  l = (float) (0.98 * height);

  //barras
  pwmA = new BarraInfo();
  pwmA.inicio = 0;
  pwmA.fin = 255;
  pwmA.x = (int) (0.9 * width);
  pwmA.y = (int) (0.1 * height);
  pwmA.anchura = width/60;
  pwmA.altura = (int) (0.8 * height);
  pwmA.pasoLinea = height/60;
  pwmA.pasoNumero = height/12;
  pwmA.recorteX = width/20;
  pwmA.recorteY = width/60;
  pwmA.digitos = 3;
  pwmA.decimales = -1;
  pwmA.etiqueta = "Motor derecho";

  pwmB = new BarraInfo();
  pwmB.inicio = 0;
  pwmB.fin = 255;
  pwmB.x = (int) (0.1 * width);
  pwmB.y = (int) (0.1 * height);
  pwmB.anchura = width/60;
  pwmB.altura = (int) (0.8 * height);
  pwmB.pasoLinea = height/60;
  pwmB.pasoNumero = height/12;
  pwmB.recorteX = width/20;
```

```
  pwmB.recorteY = width/60;
  pwmB.digitos = 3;
  pwmB.decimales = -1;
  pwmB.etiqueta = "Motor izquierdo";
}

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void draw() {                                                              //para
ir pasando de un estado a otro
  switch(estado){
    case 0:
      listaDispositivos("BUSCANDO DISPOSITIVOS", color(255, 0, 0));
      break;
    case 1:
      listaDispositivos("DISPOSITIVOS DISPONIBLES", color(51, 181, 229));
      break;
    case 2:
      conectaDispositivo();
      break;
    case 3:
      //dibujar el fondo
      background(0);
      strokeWeight(1);
      fill(31,91,76);
      stroke(92,214,166);
      ellipse(width/2,height/2,(float)(0.98*height),(float) (0.98*height));
      noFill();
      for (i = 1; i*50 < l/2; i++){
          ellipse(width/2,height/2,100*i,100*i);
      }
      muestraDatos();
        //mostrar los resultados por pantalla
      break;
    case 4:
      muestraError();
      break;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void onCreate(Bundle bundle) {                                  //función para evitar
que se apague la pantalla
super.onCreate(bundle);
getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void onStart(){
        //función que se ejecuta al iniciarse la aplicación
  super.onStart();
        //incluirlo siempre al princpio de onStart
  bttelefono = BluetoothAdapter.getDefaultAdapter();    //detectar como está el bluetooth
  if (bttelefono != null){                                                        //si el
teléfono no dispone de bluetooth devolverá null
    if (!bttelefono.isEnabLED()){                                          //si el
bluetooth no está activado, se activa. Permisos
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);     //de
bluetooth
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }
    else{
      empieza();
    }
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void onStop(){
        //función para finalizar la aplicación
  if(socket != null){                                                             //si
todavía no se ha cortado la comunicación se intenta
```

```
    try{
        //cerrar el medio de comunicación
      socket.close();
    }
    catch(IOException ex){
      println(ex);
    }
  }
  super.onStop();
        //incluirlo siempre al final del onStop
}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
public void onActivityResult (int requestCode, int resultCode, Intent data){     //se obtiene el
resultado de una operación
  if(resultCode == RESULT_OK){
                                //con resultCode se obtiene si la operación
    empieza();
                                                //ha sido exitosa o no
  }
  else{
    estado = 4;
    error = "No se ha activado el bluetooth";
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
public void mouseReleased(){                    //función que se llama cada vez que se toca la
pantalla
  switch(estado){
    case 0:
      break;
    case 1:
      compruebaEleccion();
      break;
    case 3:
      onResume();
      break;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
public void empieza(){
    dispositivos = new ArrayList<BluetoothDevice>();
    for (BluetoothDevice dispositivo : bttelefono.getBondedDevices()){
        dispositivos.add(dispositivo);          //añade los diferentes dispositivos a la matriz
de dispositivos disponibles
    }
    estado = 1;                                                     //pasar a la pantalla de
elegir dispositivos -> listaDispositivos
}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
public void listaDispositivos(String texto, int c){     //función para mostrar en pantalla los
dispositivos
  background(0);
  textFont(androidFont);
  fill(c);
  textSize(height/15);
  text(texto,width/2, 30);
  if(dispositivos != null){
    for(int indice = 0; indice < dispositivos.size(); indice++){
      BluetoothDevice dispositivo = (BluetoothDevice) dispositivos.get(indice);
      fill(0,255,255);
      int posicion = height/6 * (1 + indice);
      if(dispositivo.getName() != null){
          textSize(height/18);
          text(dispositivo.getName(),width/2, posicion);//nombre del dispositivo
      }
      fill(92, 214, 166);
      textSize(height/20);
```

```
      text(dispositivo.getAddress(),width/2, posicion + height/16);     //dirección mac del
dispositivo
      fill(255);
      line(width/4, posicion + height/12, width*3/4, posicion + height/12);
    }
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void compruebaEleccion(){
  int elegido = (mouseY - 50) / 55;
  if(elegido < dispositivos.size()){
    dispositivo = (BluetoothDevice) dispositivos.get(elegido);
    println(dispositivo.getName());
    estado = 2;                                          //ir a conectaDispositivo
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void conectaDispositivo(){       //sirve para conectar el teléfono con el dipositivo.
Cuidado con el UUID. Los números
  try{                                                              //ahí puestos son para el
bluetooth, pero se han tenido que cambiar los primeros (puerto serie)
        socket = dispositivo.createRfcommSocketToServiceRecord(UUID.fromString("00001101-0000-
1000-8000-00805F9B34FB"));
       Method m = dispositivo.getClass().getMethod("createRfcommSocket", new Class[] {
int.class });
    socket = (BluetoothSocket) m.invoke(dispositivo, 1);
        socket.connect();
    ins = socket.getInputStream();  //por aquí se recibirán datos del robot en el móvil
    ons = socket.getOutputStream(); //por aquí se enviarán datos del móvil al robot
    estado = 3;                                          //ir a muestraDatos
  }
  catch(Exception ex){
    estado = 4;
    error = ex.toString();
    println(error);
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void muestraDatos(){                     //función donde se realizarán las principales
operaciones y se mostrarán por
                                                              //pantalla los resultados
  //mandar datos del acelerometro
  ax = (ant[1] + acc_values[1]) / 2;
  ay = (ant[0] + acc_values[0]) / 2;
  giro = (byte) (ax + 10);                      //el +10 es para evitar que sea negativo
  ant[1] = acc_values[1];                       //se almacena el valor del acelerómetro para
calcular la media

  vel = (byte) (ay + 10);
  ant[0] = acc_values[0];                       //se almacena el valor del acelerómetro para
calcular la media

  valor[1]=vel;
  valor[0]=giro;

  //dibujar el circulo
  strokeWeight(2);
  stroke(2,52,77);
  fill(0,255/* - 180 / 15 * sqrt(ax*ax+ay*ay)*/,255/* - 180 / 15 * sqrt(ax*ax+ay*ay)*/,10 +
sqrt(ax*ax+ay*ay)*245/15);
  ellipse(constrain(map(ax, -10, 10, width/2-l/2+width/40, width/2+l/2-width/40), width/2-
l/2+width/20, width/2+l/2-width/20),
                  constrain(map(ay, -10, 10, height/2-l/2+width/40, height/2+l/2-width/40),
height/2-l/2+width/20, height/2+l/2-width/20),
                  width/20, width/20);
  //enviar datos al robot desde el teléfono*/
  try{
      ons.write(valor,0,valor.length);  //(variable,bites vacíos,longitud)
```

```
          ons.flush();                                    //evitar que se envíen datos que no se
desean
  }
  catch(Exception ex){
          estado = 4;
          error = ex.toString();
          println(error);
  }
  //recibir datos del robot en el teléfono
  try{
    while(ins.available() > 0){
      ins.read(ardu,0,ardu.length);             //(variable,desde donde,longitud)
    }
  }
  catch(Exception ex){
    estado = 4;
    error = ex.toString();
    println(error);
  }

  pwmA.valor = ardu[0] + 128;
  pwmA.signo = ardu[1];
  pwmB.valor = ardu[2] + 128;
  pwmB.signo = ardu[3];

  //pwmA.dibuja();
  //pwmB.dibuja();
}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
public void onResume() {                //función que sirve para volver a retomar una actividad,
en este caso el acelerómetro
  super.onResume();                                     //incluir siempre al principio del
onResume
  //se define el SensorManager (te permite acceder al sensor)
  mSensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
  //se define un SensorEventListener (recibe la información del sensor cuando este varía. Lega
desde el SensorManager)
  accSensorEventListener = new MySensorEventListener();
  //se coge el sensor tipo acelerómetro
  acc_sensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  //se guarda el SensorEventListener junto con su sensor y su Sensor Manager. Ajustar el delay
según las necesidades
  mSensorManager.registerListener(accSensorEventListener, acc_sensor,
SensorManager.SENSOR_DELAY_GAME);
} //tipos de delay en orden ascendente: SENSOR_DELAY_FASTEST; SENSOR_DELAY_GAME;
SENSOR_DELAY_NORMAL; SENSOR_DELAY_UI
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
public void onPause() {
  //desvincular el SensorEventListener antes de salir
  mSensorManager.unregisterListener(accSensorEventListener);
  super.onPause();      //incluir siempre al final del onPause
}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
public void muestraError() {                            //función para mostrar errores. Se
mostrará el error en el centro de la pantalla
                    //con fondo rojo
  background(255, 0, 0);
  fill(255, 255, 0);
  textFont(androidFont);
  textAlign(CENTER);
  translate(width / 2, height / 2);
  textSize(16);
  text(error, 0, 0);
}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
//class del sensor
class MySensorEventListener implements SensorEventListener {
  public void onSensorChanged(SensorEvent event) {
```

```
    int eventType = event.sensor.getType();
    if(eventType == Sensor.TYPE_ACCELEROMETER) {
      acc_values = event.values;
    }
  }
  public void onAccuracyChanged(Sensor sensor, int accuracy) {
    //el sdk dice que hay que incluirlo, pero solo si varía la precisión
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
class BarraInfo {
  float inicio; // primer valor del rango
  float fin; // último valor del rango
  int x; // posición x del conjunto
  int y; // posición y del conjunto
  int anchura; // anchura de las muescas y de la barra
  int altura; // altura del conjunto
  int pasoLinea; // cada cuantos píxeles se dibuja una muesca
  int pasoNumero; // cada cuantos píxeles se dibuja un valor de referencia del rango
  int recorteX; // ajuste de la posición X del texto de los valores de referencia
  int recorteY; // ajuste de la posición Y del texto de los valores de referencia
  int digitos; // Número de digitos que aparecerán en el texto de los valores de referencia
  int decimales; // Número de decimales que aparecerán en el texto de los valores de referencia
  byte signo;
  String etiqueta;
  int valor; // Valor real

  BarraInfo() {
    valor = 0;
  }

  void dibuja() {
    int indice;
    int longitud;
    textSize(height/33);
    textAlign(LEFT, CENTER);
    stroke(92,214,166);
    fill(92,214,166);
    text(nfs(map(y, y, y + altura, fin, inicio), digitos, decimales), x - recorteX, y + recorteY
- pasoLinea);
    text(nfs(map(y + altura, y, y + altura, fin, inicio), digitos, decimales), x - recorteX, y +
altura + recorteY - pasoLinea);
    for(indice = y; indice <= y + altura; indice++) {
      if(indice % pasoLinea == 0) {
        line(x, indice, x + anchura, indice);
      }
      if(indice % pasoNumero == 0) {
        text(nfs(map(indice, y, y + altura, fin, inicio), digitos, decimales), x - recorteX,
indice + recorteY - pasoLinea);
      }
    }
    if (signo == 1){
            stroke(31,91,76);
            fill(92,214,166);
    }
    else if (signo == 2){
        stroke(91,31,31);
            fill(255,155,155);
    }
    longitud = (int) (map(valor, inicio, fin, 0, altura));
    rect(x + anchura * 2, y + altura - longitud, anchura, longitud);
    textAlign(CENTER,CENTER);
    fill(92,214,166);
    text(etiqueta, x + anchura / 2, (float) (0.5 * y));
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
  public int sketchWidth() { return displayWidth; }
  public int sketchHeight() { return displayHeight; }
  public String sketchRenderer() { return JAVA2D; }
```

```
}
```

ANEXO A.14: TELEMETRÍA

```
package processing.test.mandar_datos;

import processing.core.*;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import java.util.ArrayList;
import java.util.UUID;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Method;
import android.os.Bundle;
import android.view.WindowManager;

public class mandar_datos extends PApplet {

//fuentes
String[] fontList;
PFont androidFont;

//variables de la comunicación bluetooth
private static final int REQUEST_ENABLE_BT = 3;
ArrayList<BluetoothDevice> dispositivos;//matriz con los dispositivos encontrados por el
teléfono
BluetoothAdapter bttelefono;                              //representa el bluetooth del teléfono
BluetoothDevice dispositivo;                              //representa el bluetooth de los
dispositivos ajenos al teléfono
BluetoothSocket socket;                                           //establece la conexión entre
teléfono y robot
InputStream ins;                                                  //variable para recibir datos
del robot (byte)
OutputStream ons;                                                         //variable para enviar
datos al robot (byte)

//variables del menú
int estado;
String error;

//variables para el cálculo de valor
byte[] ardu = {0,0,0,0,0};                        //valor recibido por el robot

int[] pwmA;                                                               //pwm del motor A
int[] pwmB;                                                               //pwm del motor B
int[] posicion;                                                   //posicion del robot
int i;

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
BroadcastReceiver receptor = new BroadcastReceiver(){             //un BroadcastReceiver sirve
para recibir eventos
    public void onReceive(Context context, Intent intent){
        String accion = intent.getAction();                                       //recibe el
evento que se envía a la aplicación
        if (BluetoothDevice.ACTION_FOUND.equals(accion)){         //la acción es que ha encontrado
un dispositivo bluetooth
            BluetoothDevice dispositivo =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            println(dispositivo.getName() + " " + dispositivo.getAddress());
            dispositivos.add(dispositivo);
        }
```

```
        else if (BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(accion)){      //el teléfono
está buscando disopositivos
          estado = 0;
        }
        else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(accion)){     //el teléfono ha
terminado de buscar
          estado = 1;
        }

    }
};
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
public void setup() {
  size(displayWidth, displayHeight, JAVA2D);                    //se obtienen las medidas de la
pantalla y se define que solo habrán

               //objetos en 2D
  frameRate(25);
        //refresco de pantalla

  //para que la pantalla esté girada
  orientation(LANDSCAPE);
  //si queremos que la posición de la pantalla sea vertical:
  //orientation(PORTRAIT);

  //fuentes
  fontList = PFont.list();
  androidFont = createFont(fontList[0], 32, true);
  textFont(androidFont);
  stroke(255);
  //definición del tamaño de la matriz a rellenar
  pwmA = new int[width];
  pwmB = new int[width];
  posicion = new int[width];
}

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
public void draw() {                                                                    //para
ir pasando de un estado a otro
  switch(estado){
    case 0:
      listaDispositivos("BUSCANDO DISPOSITIVOS", color(255, 0, 0));
      break;
    case 1:
      listaDispositivos("DISPOSITIVOS DISPONIBLES", color(51, 181, 229));
      break;
    case 2:
      conectaDispositivo();
      break;
    case 3:
      //dibujar el fondo
      background(0);
          strokeWeight(1);
      fill(31,91,76);
      stroke(92,214,166);
      rectMode(CORNERS);
      rect(0,(float) (0.01*height),width,(float) (0.33*height));
      rect(0,(float) (0.34*height),width,(float) (0.66*height));
      rect(0,(float) (0.67*height),width,(float) (0.99*height));
      textSize(height/33);
      textAlign(CENTER, CENTER);
      fill(253,254,31);
      stroke(253,254,31);
      text("Motor izquierdo",width/2,(float) (height*0.03));
      text("Motor derecho",width/2,(float) (height*0.36));
      text("Línea",width/2,(float) (height*0.69));
      textSize(height/44);
      textAlign(LEFT, CENTER);
      text("255",(float) (0.01*width),(float) (0.03*height));
      text("0",(float) (0.01*width),(float) (0.15*height));
```

```
      text("-255",(float) (0.01*width),(float) (0.31*height));

      text("255",(float) (0.01*width),(float) (0.36*height));
      text("0",(float) (0.01*width),(float) (0.48*height));
      text("-255",(float) (0.01*width),(float) (0.64*height));

      text("50",(float) (0.01*width),(float) (0.69*height));
      text("25",(float) (0.01*width),(float) (0.81*height));
      text("0",(float) (0.01*width),(float) (0.97*height));

      line(0,(float) (0.17*height),width,(float) (0.17*height));
      line(0,(float) (0.5*height),width,(float) (0.5*height));
      line(0,(float) (0.83*height),width,(float) (0.83*height));
      fill(92,214,166);
      stroke(92,214,166);
      muestraDatos();
        //mostrar los resultados por pantalla
      break;
    case 4:
      muestraError();
      break;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void onStart(){
        //función que se ejecuta al iniciarse la aplicación
  super.onStart();
        //incluirlo siempre al princpio de onStart
  bttelefono = BluetoothAdapter.getDefaultAdapter();    //detectar como está el bluetooth
  if (bttelefono != null){                                                    //si el
teléfono no dispone de bluetooth devolverá null
    if (!bttelefono.isEnabLED()){                                          //si el
bluetooth no está activado, se activa. Permisos
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);     //de
bluetooth
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }
    else{
      empieza();
    }
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void onCreate(Bundle bundle) {                              //función para evitar
que se apague la pantalla
          super.onCreate(bundle);
          getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        }
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void onStop(){
        //función para finalizar la aplicación
  if(socket != null){                                                       //si
todavía no se ha cortado la comunicación se intenta
    try{
        //cerrar el medio de comunicación
      socket.close();
    }
    catch(IOException ex){
      println(ex);
    }
  }
  super.onStop();
        //incluirlo siempre al final del onStop
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void onActivityResult (int requestCode, int resultCode, Intent data){     //se obtiene el
resultado de una operación
```

```
  if(resultCode == RESULT_OK){
                                //con resultCode se obtiene si la operación
    empieza();
                                                    //ha sido exitosa o no
  }
  else{
    estado = 4;
    error = "No se ha activado el bluetooth";
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void mouseReleased(){                    //función que se llama cada vez que se toca la
pantalla
  switch(estado){
    case 0:
      break;
    case 1:
      compruebaEleccion();
      break;
    case 3:
      onResume();
      break;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void empieza(){
    dispositivos = new ArrayList<BluetoothDevice>();
    for (BluetoothDevice dispositivo : bttelefono.getBondedDevices()){
        dispositivos.add(dispositivo);          //añade los diferentes dispositivos a la matriz
de dispositivos disponibles
    }
    estado = 1;                                                         //pasar a la pantalla de
elegir dispositivos -> listaDispositivos
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void listaDispositivos(String texto, int c){     //función para mostrar en pantalla los
dispositivos
  background(0);
  textFont(androidFont);
  textAlign(CENTER,CENTER);
  fill(c);
  textSize(height/15);
  text(texto,width/2, 30);
  if(dispositivos != null){
            for(int indice = 0; indice < dispositivos.size(); indice++){
              BluetoothDevice dispositivo = (BluetoothDevice) dispositivos.get(indice);
              fill(0,255,255);
              int posicion = height/6 * (1 + indice);
              if(dispositivo.getName() != null){
                  textSize(height/18);
                  text(dispositivo.getName(),width/2, posicion);//nombre del dispositivo
              }
              fill(92, 214, 166);
              textSize(height/20);
              text(dispositivo.getAddress(),width/2, posicion + height/16);     //dirección mac
del dispositivo
              fill(255);
              line(width/4, posicion + height/12, width*3/4, posicion + height/12);
            }
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void compruebaEleccion(){
  int elegido = (mouseY - 50) / 55;
  if(elegido < dispositivos.size()){
    dispositivo = (BluetoothDevice) dispositivos.get(elegido);
    println(dispositivo.getName());
    estado = 2;                                         //ir a conectaDispositivo
```

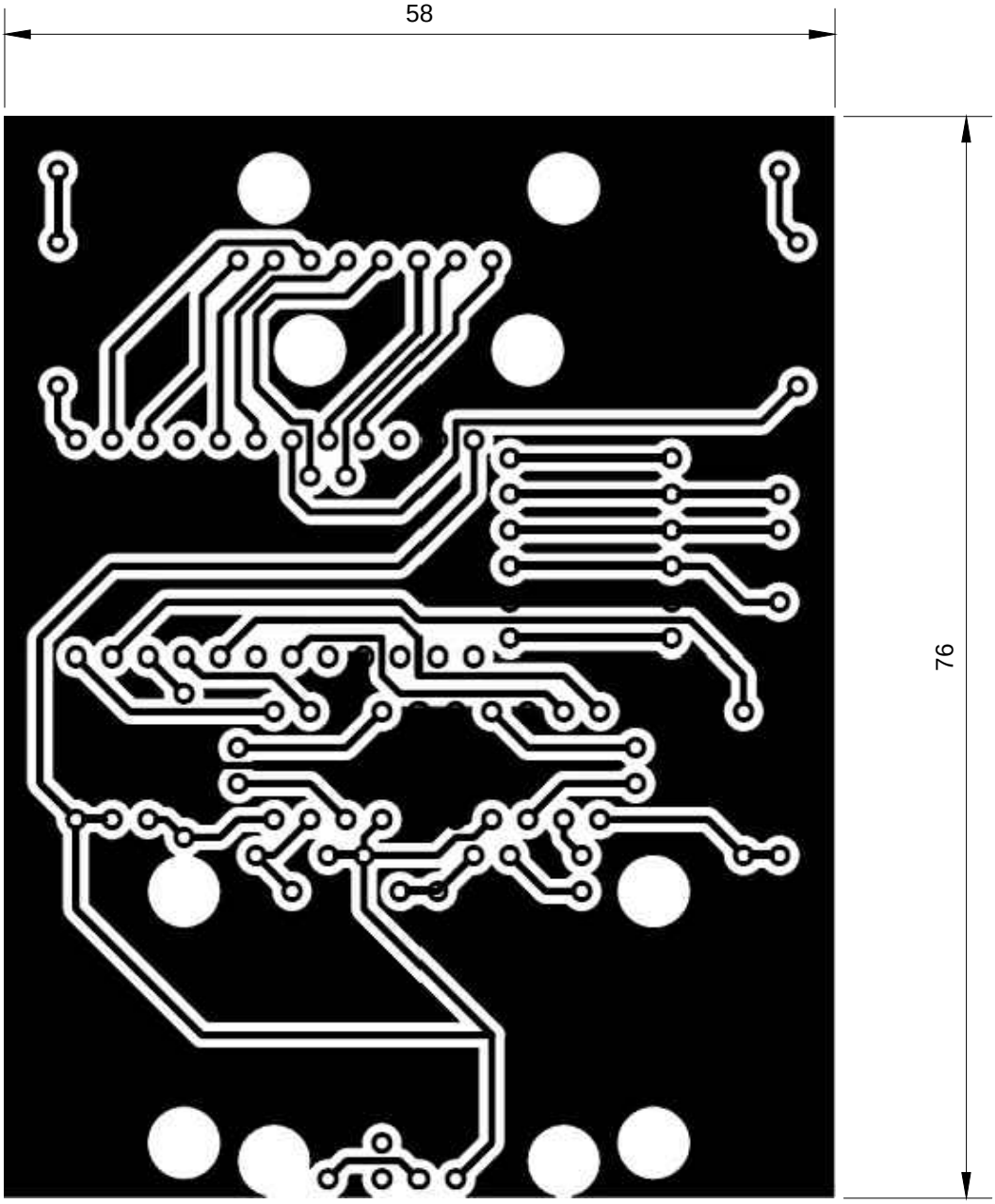
```
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void conectaDispositivo(){       //sirve para conectar el teléfono con el dipositivo.
Cuidado con el UUID. Los números
  try{                                                          //ahí puestos son para el
bluetooth, pero se han tenido que cambiar los primeros (puerto serie)
        socket = dispositivo.createRfcommSocketToServiceRecord(UUID.fromString("00001101-0000-
1000-8000-00805F9B34FB"));
        Method m = dispositivo.getClass().getMethod("createRfcommSocket", new Class[] {
int.class });
    socket = (BluetoothSocket) m.invoke(dispositivo, 1);
        socket.connect();
    ins = socket.getInputStream();  //por aquí se recibirán datos del robot en el móvil
    ons = socket.getOutputStream(); //por aquí se enviarán datos del móvil al robot
    estado = 3;                                             //ir a muestraDatos
  }
  catch(Exception ex){
    estado = 4;
    error = ex.toString();
    println(error);
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void muestraDatos(){                     //función donde se realizarán las principales
operaciones y se mostrarán por
                                                            //pantalla los resultados


  try{                                                      //recibir datos del robot en el
teléfono
    while(ins.available() > 0){
      ins.read(ardu,0,ardu.length);
    }
  }
  catch(Exception ex){
    estado = 4;
    error = ex.toString();
    println(error);
  }

  for (i=1;i<width;i++){
          pwmA[i-1]=pwmA[i];
          pwmB[i-1]=pwmB[i];
          posicion[i-1]=posicion[i];
  }
  //Añadir valores
  if (ardu[1]==1){
          pwmA[i-1]=ardu[0]+128;
  }
  else if (ardu[1]==2){
          pwmA[i-1]=-1*(ardu[0]+128);
  }
  if (ardu[3]==1){
          pwmB[i-1]=ardu[2]+128;
  }
  else if (ardu[3]==2){
          pwmB[i-1]=-1*(ardu[2]+128);
  }
  posicion[i-1]=ardu[4];
  for (i=1;i<width;i++){
          point(i,map(pwmA[i],-255,255,(float)(0.33*height),(float)(0.01*height)));
          line(i,map(pwmA[i],-255,255,(float)(0.33*height),(float)(0.01*height)),i,map(pwmA[i-
1],-255,255,(float)(0.33*height),(float)(0.01*height)));
          point(i,map(pwmB[i],-255,255,(float)(0.66*height),(float)(0.34*height)));
          line(i,map(pwmB[i],-255,255,(float)(0.66*height),(float)(0.34*height)),i,map(pwmB[i-
1],-255,255,(float)(0.66*height),(float)(0.34*height)));
          point(i,map(posicion[i],0,50,(float)(0.99*height),(float)(0.67*height)));
```

```
line(i,map(posicion[i],0,50,(float)(0.99*height),(float)(0.67*height)),i,map(posicion[i-
1],0,50,(float)(0.99*height),(float)(0.67*height)));
  }
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
public void muestraError()                              //función para mostrar errores. Se
mostrará el error en el centro de la pantalla
{                                                                                  //con fondo rojo
  background(255, 0, 0);
  fill(255, 255, 0);
  textFont(androidFont);
  textAlign(CENTER);
  translate(width / 2, height / 2);
  textSize(16);
  text(error, 0, 0);
}
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////
  public int sketchWidth() { return displayWidth; }
  public int sketchHeight() { return displayHeight; }
  public String sketchRenderer() { return JAVA2D; }
}
```

ANEXO B: PLANOS

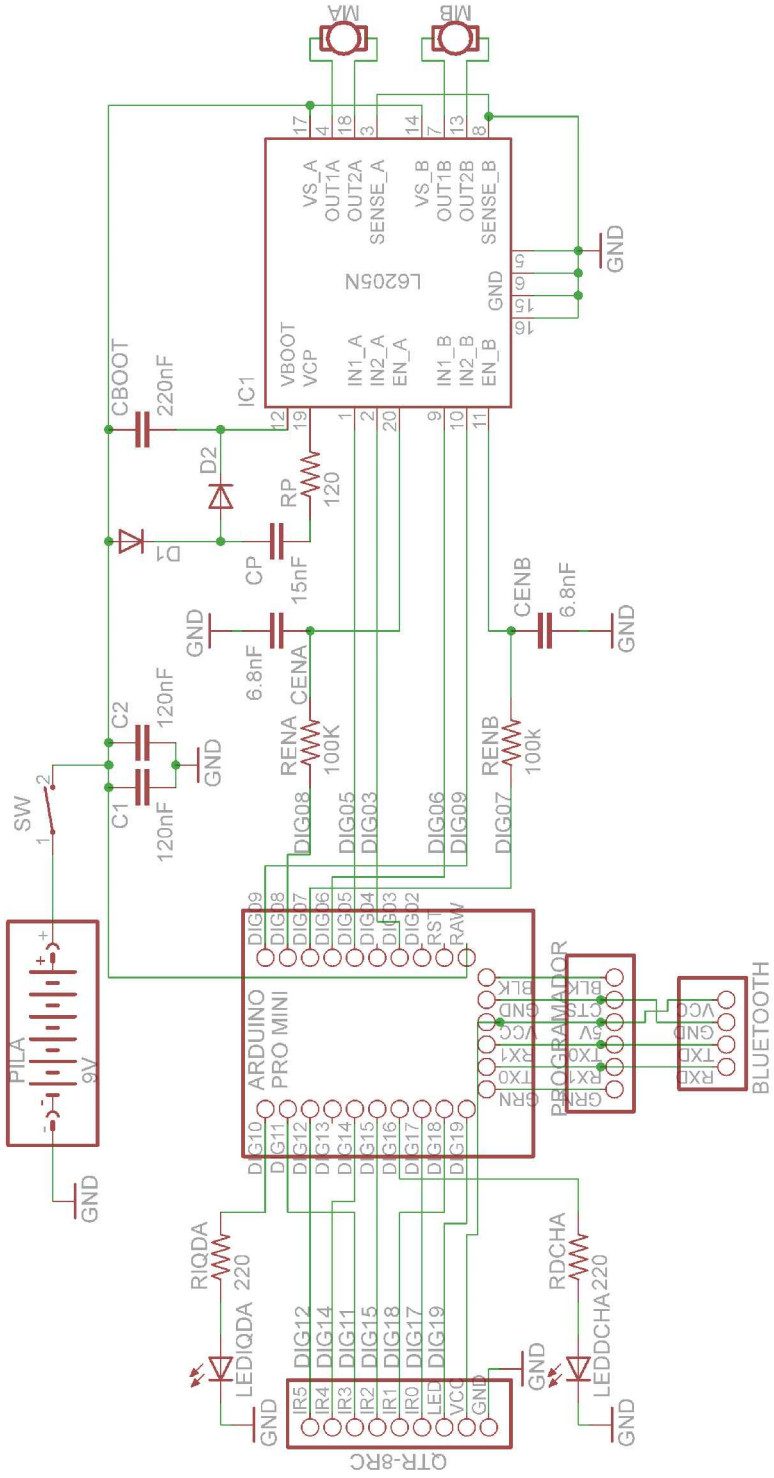
ANEXO B.1: DISEÑO DEL PCB

ANEXO B.2: ESQUEMA ELÉCTRICO DEL ROBOT



Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO INDUSTRIAL**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELECTRICA Y ELECTRONICA**
PROYECTO: **ROBOT RASTREADOR CON INTERFAZ BLUETOOTH/ANDROID**		REALIZADO: **PEREZ MOTOS, JOSE LUIS** FIRMA:

PLANO:	FECHA:	ESCALA:	Nº PLANO:
DISEÑO DEL PCB	**18/2/2013**	**2:1**	**1**



Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO INDUSTRIAL**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELECTRICA Y ELECTRONICA**
PROYECTO: **ROBOT RASTREADOR CON INTERFAZ BLUETOOTH/ANDROID**		REALIZADO: **PEREZ MOTOS, JOSE LUIS** FIRMA:
PLANO: **ESQUEMA ELÉCTRICO DEL ROBOT**	FECHA: **18/2/2013**	ESCALA: / Nº PLANO: **2**