eman ta zabal zazu

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIAL DE BILBAO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

2016 / 2017

INGENIERIA INVERSA DEL MANDO DE PS2 CABLEADO PARA EL CONTROL DE UN ROBOT SEGUIDOR DE LINEA CON DETECCION DE OBSTACULOS POR RF

2. DISEÑO

DATOS DE LA ALUMNA O DEL ALUMNO	DATOS DEL DIRECTOR O DE LA DIRECTORA
NOMBRE: ENDIKA	NOMBRE: OSKAR
APELLIDOS: PÉREZ RAMÍREZ	APELLIDOS: CASQUERO OYARZABAL
	DEPARTAMENTO: INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
FDO.:	FDO.:
FECHA: 18/01/2017	FECHA: 18/01/2017

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. **INTRODUCCIÓN**

El proyecto consiste en un vehículo controlado mediante radiofrecuencia a través de un mando de la plataforma de videojuegos PlayStation II, con opciones de funcionamiento en forma autónoma evitando obstáculos y en modo seguidor de línea. Para el control de todos los componentes del TFG se han utilizado dos placas Arduino con su respectiva programación, una para el vehículo y otra para el mando. En este documento se explicarán los pasos necesarios para llevar a cabo el proyecto, los problemas surgidos en su desarrollo, así como las soluciones encontradas, con el fin de ofrecer una explicación práctica del prototipo.

El proyecto está dividido en dos partes diferenciadas. El coche y el mando.

El coche, cuyo chasis es el de PrintBot Renacuajo de BQ, dispone de dos ruedas (que dotaan de movimiento al vehículo) controladas por un servomotor de rotación continua cada una, dos sensores infrarrojos (que actúan como detectores de línea) situados en la parte inferior delantera del coche, un sensor de proximidad ultrasónico situado en la parte superior delantera del coche que gira a izquierda y derecha por medio de un servomotor, también se necesita un receptor de radiofrecuencia para recibir las órdenes enviadas por el mando, todos estos elementos se encuentran conectados y controlados por la placa Freaduino UNO.

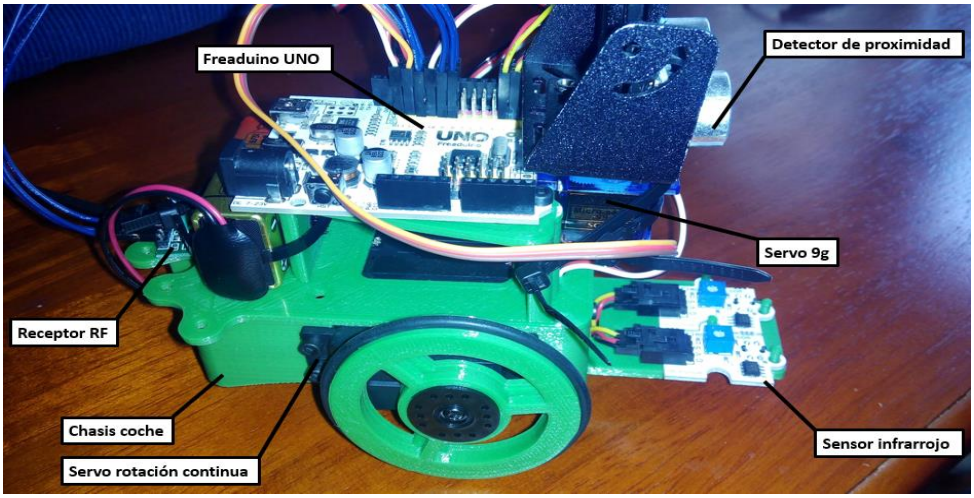

Figura 1. Coche.

El mando, como ya antes se ha mencionado, es el de la plataforma de videojuegos PlayStation II, que se ha modificado para su funcionamiento inalámbrico y para poder situar en su interior diferentes elementos como una batería Lipo, recargable mediante un enchufe y protegido el circuito mediante un interruptor, el emisor de radiofrecuencia RF433, una antena, y la placa Arduino Pro Mini 3,3V a la que están conectados y controlados por ella todos los elementos.

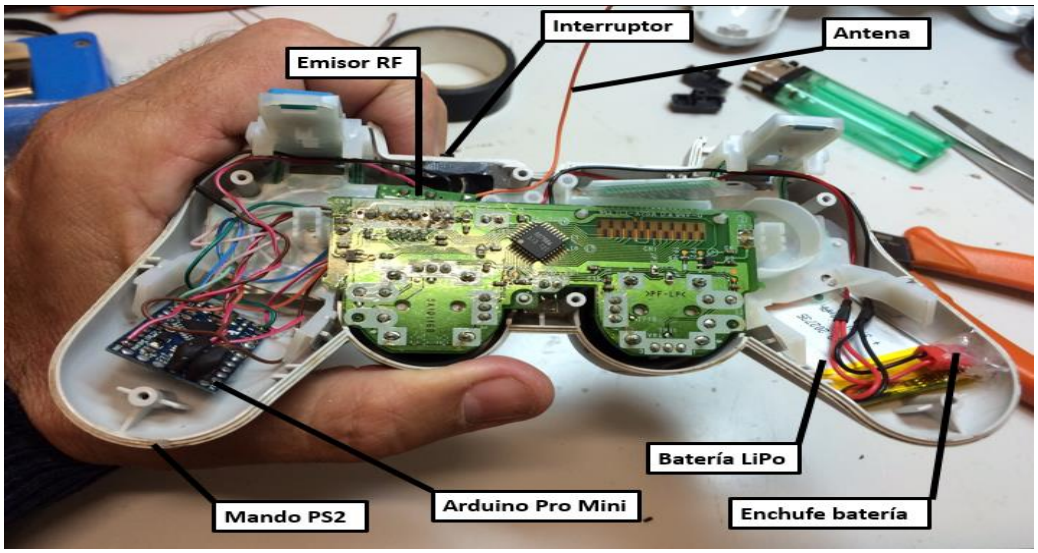

Figura 2.Mando.

2. COMPONENTES

2.1. COMPONENTES DEL VEHÍCULO

2.1.1. FREADUINO UNO

Freaduino Uno es compatible 100% con el IDE de Arduino, y los Shields diseñados para el mismo.

Una de las ventajas de esta placa es la inclusión de los headers para conexiones externas, que incluyen +Vcc y Gnd. Estos headers están claramente señalados al pin que corresponden, y brindan una gran ayuda el momento de conectar servomotores o sensores.

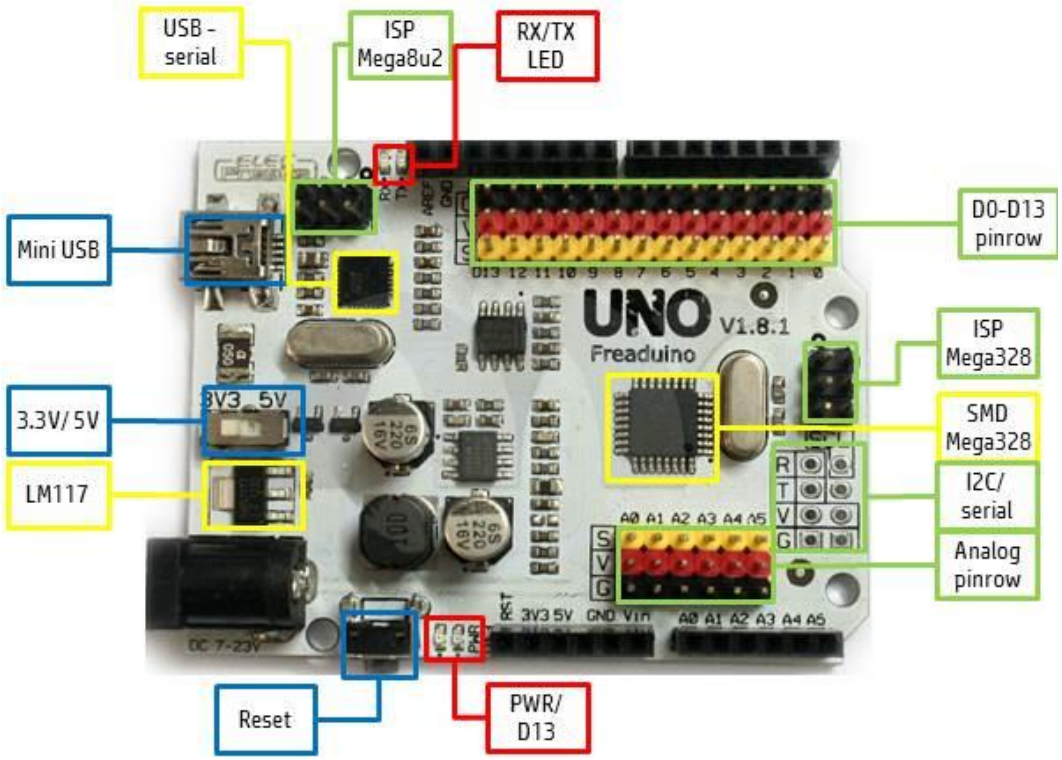

Figura 3. Freaduino UNO.

Extraída de: http://www.mathias-wilhelm.de/arduino/assets/boards/Slide1cr.jpg

Otra función adicional, que podrá resultar útil, es la inclusión de un selector de voltaje de trabajo. Todas las tarjetas Arduino tienen una salida de voltaje de referencia de 3.3v, pero el microcontrolador siempre funcionará a 5Vcc internamente. El Freaduino Uno posee un selector de voltaje de operación que permite que todo el microcontrolador opere a 3.3v. Esto resulta útil para interconectar dispositivos que requieran este voltaje de funcionamiento, eliminando el requerimiento de componentes adicionales para la conexión.

Características:

- Modelo: Freaduino UNO.
- Voltaje de alimentación: de 7 a 23V.
- Voltaje de operación: 3.3V/5V seleccionable.
- Corriente de salida máx.: 800mA (3.3V) / 2000mA (5V).
- Microcontrolador: ATMega 328 SMD.
- Memoria Flash: 32Kb.
- Memoria Ram: 2Kb.
- Memoria EEPROM: 1Kb.
- Tipo de conector USB: mini USB.
- E/S Digitales: 14.
- E/S Analógicas: 6.

- E/S PWM: 6 (compartidas con las E/S digitales).
- Pines dedicados para IIC/UART: Si.
- Rieles +Vcc y Gnd para E/S: Si.

A la hora de cargar el programa en la placa se debe tener en cuenta que en el caso de que se encuentren conectados a esta elementos que requieran un alto consumo de corriente para su funcionamiento, no será suficiente con la conexión de la placa al ordenador a través del USB, además la placa deberá estar conectada a la red eléctrica a través de un transformador.

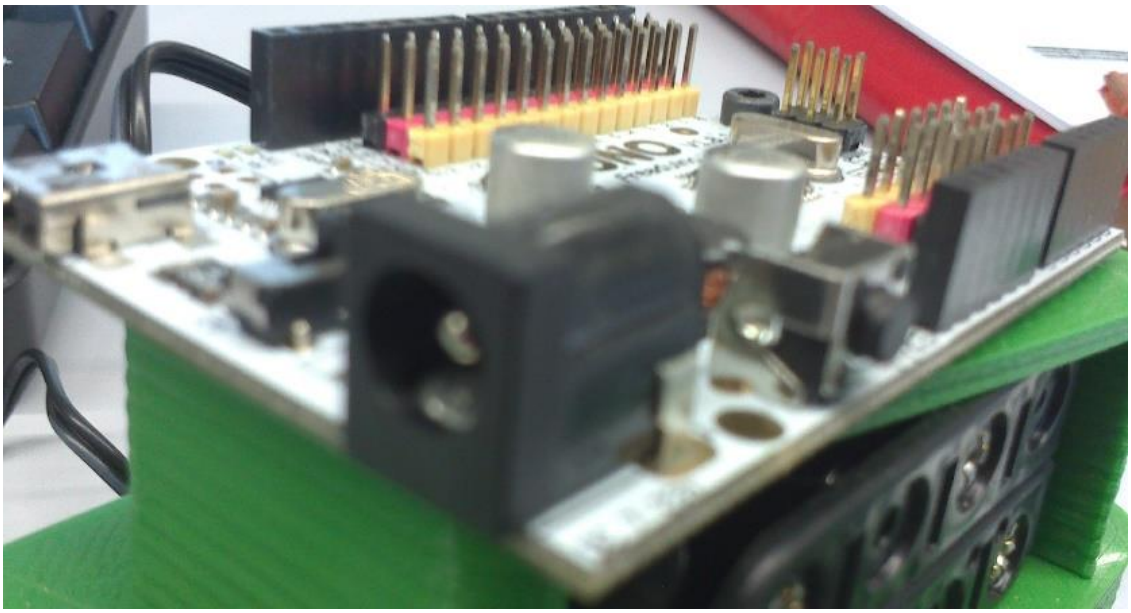

Figura 4. Entrada alimentación transformador.

Conexionado:

Tabla 1. Conexiones Freaduino UNO.

Freaduino UNO	Dispositivos	
Pinrow 11 V	Receptor RF	Pin VCC
Pinrow 11 G	Receptor RF	Pin GND
Pinrow 11 S	Receptor RF	Pin DATA
Pinrow 8 V	HC-SR04	Pin VCC
Pinrow 8 G	HC-SR04	Pin GND
Pinrow 8 S	HC-SR04	Pin Trigger
Pinrow 9 S	HC-SR04	Pin Echo
Pinrow 12 V	Servo 9g	Pin VCC
Pinrow 12 G	Servo 9g	Pin GND
Pinrow 12 S	Servo 9g	Pin Señal
Pinrow 10 V	Servo rueda izda	Pin VCC
Pinrow 10 G	Servo rueda izda	Pin GND
Pinrow 10 S	Servo rueda izda	Pin Señal
Pinrow 13 V	Servo rueda dcha	Pin VCC
Pinrow 13 G	Servo rueda dcha	Pin GND
Pinrow 13 S	Servo rueda dcha	Pin Señal
Pinrow 7 V	Sensor IR izda	Pin VCC
Pinrow 7 G	Sensor IR izda	Pin GND
Pinrow 7 S	Sensor IR izda	Pin Señal
Pinrow 6 V	Sensor IR dcha	Pin VCC
Pinrow 6 G	Sensor IR dcha	Pin GND
Pinrow 6 S	Sensor IR dcha	Pin Señal
Pin VCC	Batería LiPo 11.1V	Pin Vcc
Pin GND	Batería LiPo 11.1V	Pin Gnd

2.1.2. RECEPTOR RF433MHZ

Los módulos de comunicación por radiofrecuencia RF433MHz emisor y receptor nos van a permitir conectar dos Arduinos y que estos puedan comunicarse unidireccionalmente.

Para hacerlos funcionar existen varías librarías (VirtualWire y RadioHead). En este proyecto se ha utilizado la librería RadioHead. Esta librería se encarga de gestionar las funciones más básicas de estos módulos de radiofrecuencia: envío de información, comprobación de los paquetes de datos…

Características:

- Modelo: MX-05V.
- Voltaje de operación: 5V.
- Consumo: 4mA.
- Frecuencia de recepción: 433Mhz.
- Modulación: ASK/OOK.
- Dimensiones: 30x14mm.

Conexionado:

Tabla 2. Conexiones receptor RF.

Receptor RF	Freaduino UNO
Pin VCC	Pinrow 11 V
Pin GND	Pinrow 11 G
Pin DATA	Pinrow 11 S

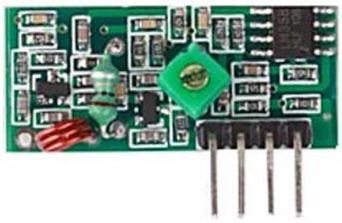

Figura 5. Receptor RF.

Extraída de: http://mla-s1-p.mlstatic.com/modulo-rf-transmisor-y-receptor-433-mhz-arduino-robotica-pic-20271-MLA20186866356_102014-F.jpg

2.1.3. CHASIS

PrintBot Renacuajo es un kit completo para montaje de un vehículo de dos ruedas, cuyo chasis está compuesto por una unidad central donde se encuentran los motores que accionan las ruedas, un compartimento en la parte media en el que se aloja la batería que alimenta la placa, un espacio en la parte superior donde va atornillada la placa microcontroladora, y por último un saliente en la parte delantera del vehículo utilizado para colocar los sensores IR.

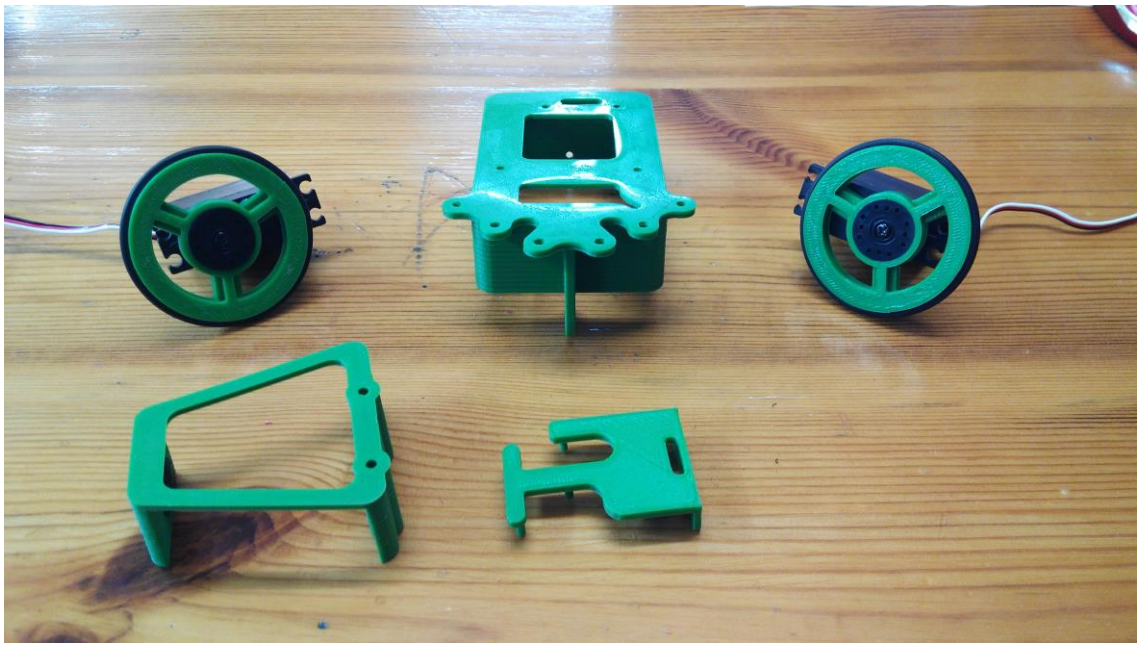

Figura 6. Despiece del coche.

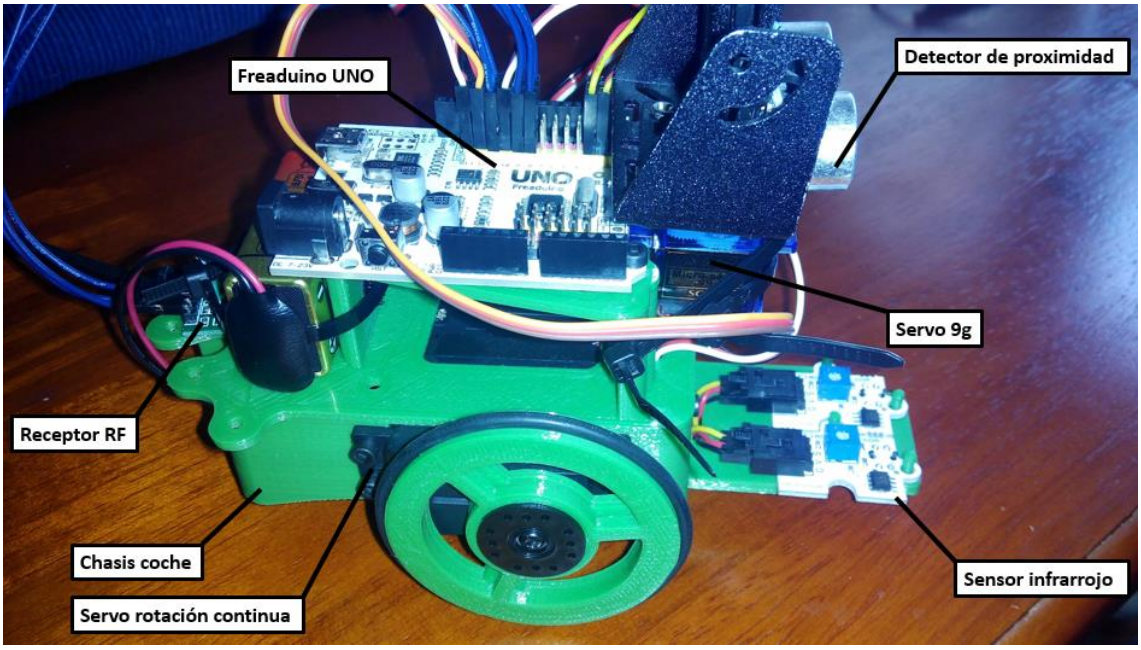

Figura 7.Coche montado.

2.1.4. DETECTOR DE PROXIMIDAD (HC SR04)

Es un sensor ultrasónico que no sólo puede detectar si un objeto se presenta, como un sensor PIR (Passive Infrared Sensor), sino que también puede sentir y transmitir la distancia al objeto, esto lo consigue enviando un ultrasonido (inaudible para el oído humano por su alta frecuencia) a través de uno de la pareja de cilindros que compone el sensor (un transductor) y espera a que dicho sonido rebote sobre un objeto y vuelva, retorno captado por el otro cilindro. Ofrece una excelente detección sin contacto (remoto) con elevada precisión y lecturas estables, en concreto tiene un rango de distancias sensible entre 3cm y 3m con una precisión de 3mm. El funcionamiento no se ve afectado por la luz solar o el material negro como telémetros ópticos (aunque acústicamente materiales suaves como telas pueden ser difíciles de detectar). La velocidad del sonido en el aire (a una temperatura de 20 °C) es de 343 m/s. (por cada grado centígrado que sube la temperatura, la velocidad del sonido aumenta en 0,6 m/s).

Características:

- Modelo: HC SR04.
- Voltaje de operación: 5V.
- Corriente de operación: 15mA.
- Frecuencia de operación: 40Hz.
- Distancia máxima de detección: 4m.
- Distancia mínima de detección: 2cm.
- Ángulo de apertura: 15º.
- Precisión: 3mm.

- Dimensiones: 45x20x15mm.

Conexionado:

Tabla 3. Conexiones sensor de ultrasonidos.

HC-SR04	Freaduino UNO
Pin VCC	Pinrow 8 V
Pin GND	Pinrow 8 G
Pin Trigger	Pinrow 8 S
Pin Echo	Pinrow 9 S

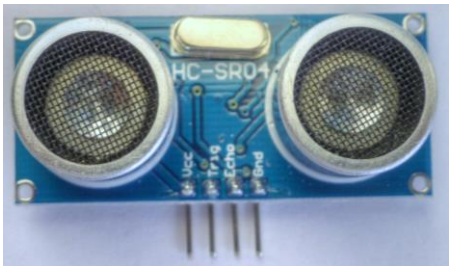

Figura 8. Sensor de ultrasonidos.

Extraída de:

https://upload.wikimedia.org/wikipedia/commons/2/20/HC_SR04_Ultrasonic_sensor_1480322_3_4_HDR_Enhancer.jpg

2.1.5. SERVO 9G

El servo SG90 Tower Pro un servo miniatura de gran calidad y diminutas dimensiones. Funciona con la mayoría de tarjetas electrónicas de control con microcontroladores y además con la mayoría de los sistemas de radio control.

El servo SG90 tiene un conector universal tipo “S” que encaja perfectamente en la mayoría de los receptores de radio control incluyendo los Futaba, JR, GWS, Cirrus, Hitec y otros. Los cables en el conector están distribuidos de la siguiente forma: Rojo=Alimentación (+), Cafe = Alimentación (–) o tierra, Naranja= Señal PWM (en este caso pin digital 12).

Este servo es el encargado de hacer girar a izquierda y derecha el sensor ultrasónico, ambos se encuentran en la parte superior delantera del vehículo.

NOTA IMPORTANTE: hay que tener en cuenta que al llevar el servomotor a una posición extrema de giro pueden surgir problemas si esta posición fuerza el servo a girar más allá de sus posibilidades (aparece un pequeño ruido), entonces no llegará la corriente necesaria a los demás dispositivos.

Características:

- Modelo: Micro Servo Tower-pro.
- Voltaje de operación: 3-7.2V.
- Temperatura de funcionamiento: -30 ℃ ~ 60 ℃.
- Velocidad: 0.10 sec/60° a 4.8V.
- Torque: 1.8 Kg-cm a 4.8V.

- Ángulo de rotación: 180°.
- Ancho de pulso: 500-2400 µs.
- Longitud de cable de conector: 24.5cm.
- Peso: 9g.
- Dimensiones: 32x30x12mm.

Conexionado:

Tabla 4. Conexiones servo 9g.

Servo 9g	Freaduino UNO
Pin VCC	Pinrow 12 V
Pin GND	Pinrow 12 G
Pin Señal	Pinrow 12 S

Figura 9. Servo 9g.

Extraída de: https://upload.wikimedia.org/wikipedia/commons/d/d3/Micro_servo.jpg

2.1.6. SERVOMOTOR DE ROTACIÓN CONTINUA (RUEDAS)

Se trata de un servomotor modificado para que pueda llegar a realizar giros completos de 360 grados, además también es posible el control de su velocidad así como de su sentido de giro.

No necesitan driver, simplemente una señal PWM, son sumamente poderosos para su tamaño, no consumen mucha energía y tienen una gran precisión.

NOTA IMPORTANTE: durante el desarrollo del TFG se estropeó uno de los servomotores, ese hecho ocasionó que dicho servomotor consumiera para sí gran parte de la corriente que alimenta los demás elementos del coche dejando por lo tanto inutilizados a los demás componentes.

Características:

- Modelo: SM-S4303R.
- Voltaje de operación: 4.8V – 6V.
- Velocidad de operación: 0.13seg – 60o.
- Fuerza: 39.2 oz-in.
- Dimensiones: 39.5x20x35.6mm.
- Peso: 42g.

Conexionado:

Tabla 5. Conexiones servo rueda.

Servo rueda izda	Freaduino UNO
Pin VCC	Pinrow 10 V
Pin GND	Pinrow 10 G
Pin Señal	Pinrow 10 S

Tabla 6. Conexiones Servo rueda.

Servo rueda dcha	Freaduino UNO
Pin VCC	Pinrow 13 V
Pin GND	Pinrow 13 G
Pin Señal	Pinrow 13 S

2.1.7. SENSOR INFRARROJO (OCTOPUS HUNT SENSOR)

"Octopus Hunt Sensor" es un sensor basado en el elemento sensitivo fotoeléctrico TCRT5000. Es capaz de detectar la señal infrarroja reflejada, de este modo se utiliza como detector de línea. Dispone de una sensibilidad ajustable mediante el potenciómetro azul, y de tres pines: dos pines de entrada 5V y GND y un pin de salida de señal digital.

Características:

- Modelo: EF04002.
- Voltaje de alimentación: 5V.
- Corriente de operación de tubo emisor: 10mA.
- Corriente de operación de tubo receptor: 0.1mA.
- Conversor fotoeléctrico: TCRT5000.
- Tipo de interfaz: analógica.
- Dimensiones: 31.4x23.4m.

Conexionado:

Tabla 7. Conexiones Sensor IR izquierdo.

Sensor IR izquierdo	Freaduino UNO
Pin VCC	Pinrow 7 V
Pin GND	Pinrow 7 G
Pin Señal	Pinrow 7 S

Tabla 8. Conexiones Sensor IR derecho.

Sensor IR derecho	Freaduino UNO
Pin VCC	Pinrow 6 V
Pin GND	Pinrow 6 G
Pin Señal	Pinrow 6 S

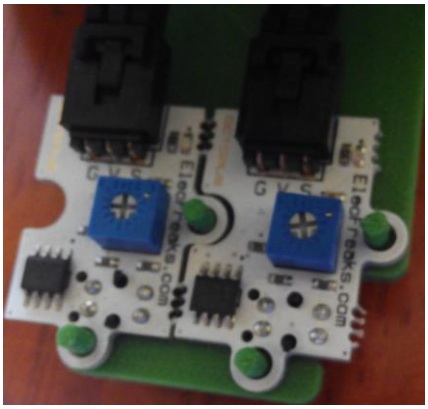

Figura 10. Sensor infrarrojo

2.1.8. BATERÍA COCHE (LIPO)

La batería LiPo será la encargada de alimentar todos los componentes del coche a través de la placa Fraduino UNO. Como la batería proporciona 11.1V de tensión nominal y 800mA la placa se alimenta mediante la conexión del polo positivo (cable rojo) de la batería y del polo negativo de la batería (cable negro) a la entrada de la fuente que admite tensiones entre 7V y 23V.

De las baterías disponibles en el mercado, estas ofrecen una mayor densidad de energía, así como una tasa de descarga bastante superior, además tienen un tamaño más reducido respecto a las de otros componentes. Esta batería proporciona una tensión nominal de 11.1V y una corriente de 800mA.

Las baterías LiPo disponen también de un circuito de protección en su parte superior, en el proceso de descarga, cuando llega a un voltaje mínimo, la función del circuito protector consistirá en apagar la batería.

Para recargar la batería se utiliza un adaptador al que se le puede conectar un cable USB.

Figura 11. Entrada Fuente Freaduino.

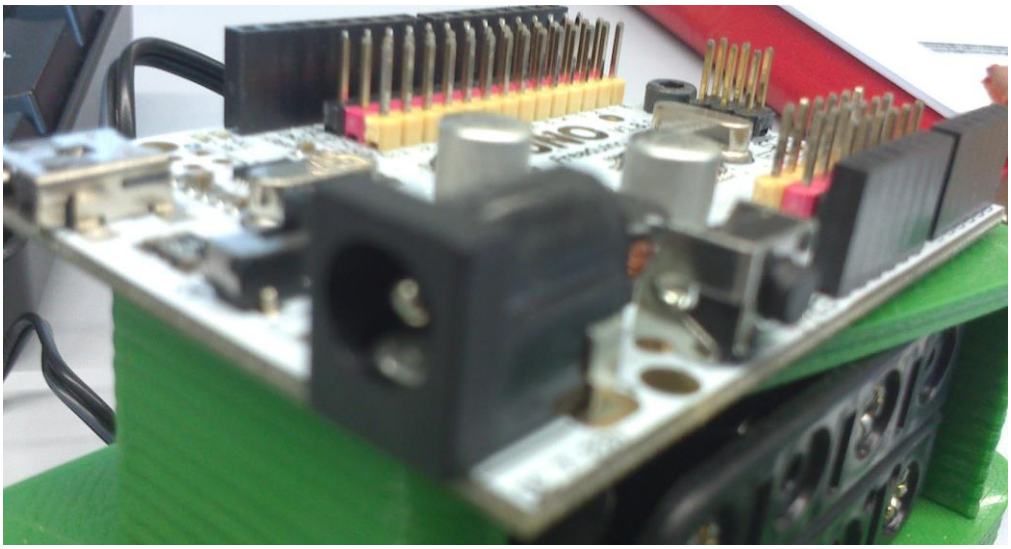

Figura 12. Entrada fuente Freaduino.

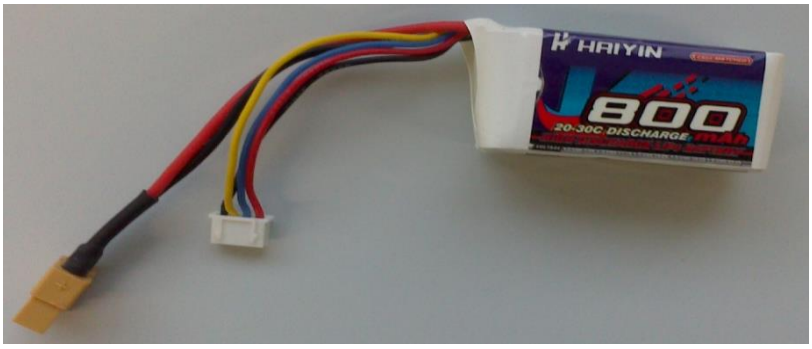

Figura 13. Batería LiPo 11.1V.

2.2. COMPONENTES DEL MANDO

2.2.1. ARDUINO PRO MINI 3,3V

La placa Arduino Pro Mini 3,3V interconecta y controla todos los componente del mando. Un UCTRL miniatura de gran calidad y diminutas dimensiones que permite insertarlo dentro del mando.

Arduino puede tomar información del entorno a través de sus entradas analógicas y digitales, puede controlar luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador.

Existen dos versiones de Arduino Pro Mini. Una de 3,3V cuya velocidad de trabajo es de 8MHz y otra de 5V cuyo reloj trabaja a 16MHz. En este proyecto se utiliza el modelo de 3,3V:

- La placa no dispone de conexión USB ni de conectores.
- El Arduino Pro Mini es una tarjeta de desarrollo basada en el ATmega328. Cuenta con 14 pines de entradas/salidas digital (de las cuales 6 se puede usar como salidas PWM), 6 entradas analógicas.
- Los pines 0 y 1 están reservados para la comunicación serie, por lo tanto no pueden ser utilizados como entradas y salidas.
- Mediante el botón de reset se reinicia la placa volviendo a iniciar la rutina de programación.

- La placa dispone de dos LEDs. Uno de ellos es el LED de encendido y el otro el LED de funcionamiento, este último LED puede ser utilizado a través del pin 13 como salida de un programa, normalmente para realizar pruebas.

- La placa dispone de 8 entradas analógicas (A0-A7), de las cuales 4 (A0-A3) están situadas en la parte inferior, y las otras 4 (A4-A7) en la parte superior del reverso de la placa. Estas entradas son muy importantes para la utilización de los sensores, la información proveniente se mide en una escala de 0 a 1024 (resolución de 10 bits). Estos pines también puedes ser configurados como salidas, aunque no es muy recomendable.

- Los seis pines de la parte derecha de la placa se pueden conectar a un conector de 6 pines para emplear un cable FTDI o una tarjeta FTDI (como el FTDI Basic Breakout) para suministrar voltaje USB y establecer comunicación con el circuito.

- Para poner en funcionamiento la placa existen dos modos:

    - El primero es el mencionado en el anterior punto, mediante el intermediario TTL_USB y de ahí se conecta al ordenador.

    - El segundo, según el modelo se utiliza cuando la tensión aplicada no es exactamente la tensión de trabajo, en lugar de alimentar la placa por el pin VCC, se alimentará por RAW, ya que la placa tiene un regulador interno de tensión y gracias a esto puede soportar una tensión de hasta 12V.

- En cuanto a la memoria, tiene un microcontrolador ATmega328 de 32kB de memoria para guardar el código de programación, una SRAM de 2kB y una EEPROM de 1kB.

- Los pines 0(RX) y 1(TX) son utilizados para transmitir y recibir la comunicación serie. Los pines SPI 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) sirven para la

comunicación SPI. Los pines A4 (SDA) and A5 (SCL) sirven para la comunicación I2C usando la librería Wire.

Características:

- Modelo: Arduino Pro Mini 3.3V.
- Voltaje de alimentación: de 3.3V a 12V.
- Voltaje de operación: 3.3V.
- Intensidad máxima por E/S 40mA.
- Microcontrolador: ATmega328.
- Memoria Flash: 32KB (de los cuales 2KB están reservados por el gestor de arranque).
- SRAM: 2KB.
- EEPROM: 1KB.
- Velocidad de Reloj: 8 MHz.
- Pines digitales de E/S: 14 (6 de los cuales tienen salida PWM).
- Pines de entrada analógica: 8.
- Dimensiones: 18x33mm.

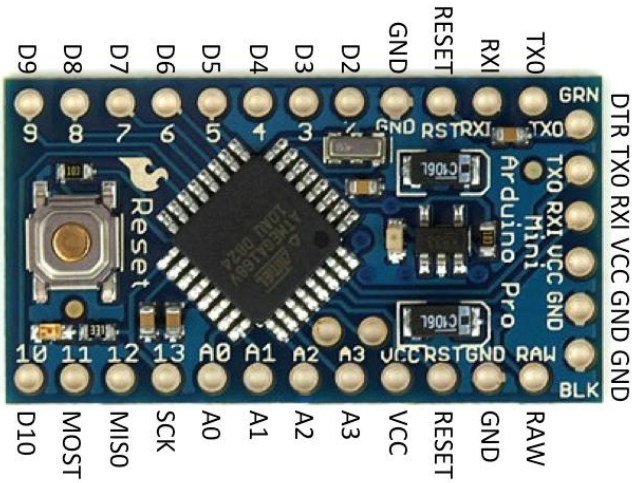

Figura 14. Pines Arduino Pro Mini.

Extraída de: https://3.bp.blogspot.com/-8JBi23wdV9s/VQx5BoqjLRI/AAAAAAAADeE/VfbuZKVR6gg/w800-h800/tumblr_lxbpcgXIii1qljpwd.png

La placa Arduino Pro Mini no dispone de circuito conector USB, esto quiere decir que para poder programar en la placa habrá que utilizar un adaptador serie USB-TTL, este adaptador podrá ser un cable de conversión FTDI-TTL-USB o una placa electrónica USB-TTL, en este proyecto se utilizará la segunda opción.

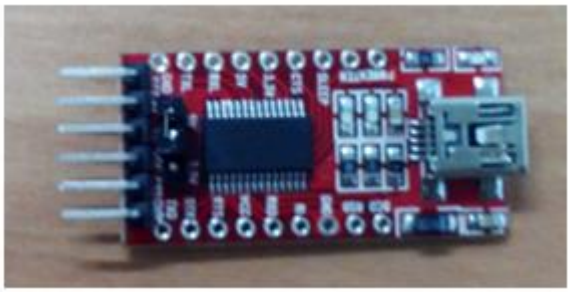

Figura 15. Adaptador serie.

Conexionado:

Tabla 9. Conexiones Arduino Pro Mini.

Arduino Pro Mini 3.3V	Dispositivos	
Pin13	Mando PS2	Pin1. Data
Pin11	Mando PS2	Pin2. Comando
GND	Mando PS2	Pin4. Tierra
RAW	Mando PS2	Pin5. Vcc. 3,3V
Pin10	Mando PS2	Pin6. Attention(send)
Pin9	Mando PS2	Pin7. Reloj
Pin VCC	Emisor RF	Pin VCC
Pin GND	Emisor RF	Pin GND
Pin digital 12	Emisor RF	Pin DATA
Pin RAW	Batería LiPo 3.7V	Pin Vcc
Pin GND	Batería LiPo 3.7V	Pin Gnd

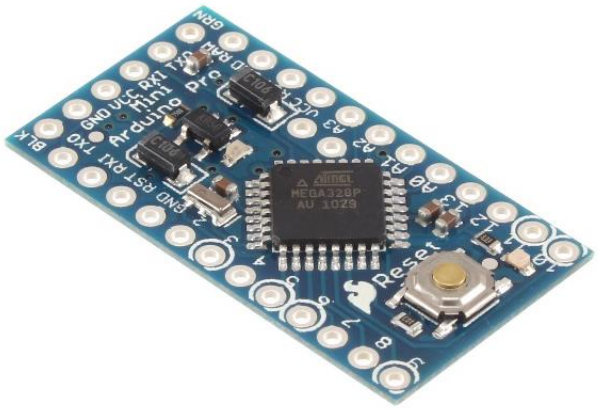

Figura 16. Arduino Pro Mini 3.3V.

Extraída de: https://upload.wikimedia.org/wikipedia/commons/0/01/Arduino_Pro_Mini_(2).jpg

2.2.2. BATERÍA MANDO (LIPO)

La batería LiPo será la encargada de alimentar todos los componentes del mando a través de la placa Arduino Pro Mini 3,3V. Como la batería proporciona 3,7V de tensión nominal la placa se alimenta mediante la conexión del polo positivo (cable rojo) de la batería al pin RAW, a su vez, el polo negativo de la batería (cable negro) se conecta al pin GND de la placa. Teniendo en cuenta el lugar donde va situada la batería y su trabajo a desempeñar se ha elegido una de 100mA de capacidad. De esta manera se consigue la completa autonomía del mando sin necesidad de cables.

Para recargar la batería se utiliza un adaptador al que se le puede conectar un cable USB, este adaptador conexiona con el cable de la batería gracias a un pequeño hueco abierto en lateral del mando.

Las baterías LiPo disponen también de un circuito de protección en su parte superior, ya que esta batería proporciona una tensión de 3,7V, en el proceso de descarga cuando llega a 3V la función del circuito protector consiste en apagar la batería.

Figura 17. Batería LiPo 3.7V.

Además, se ha realizado un circuito paralelo con el objetivo de proteger la placa de sobretensiones durante el proceso de recarga de la batería, a través de un interruptor (el cual conecta/desconecta la batería del Arduino) situado en la parte superior del mando mediante la apertura de un pequeño hueco en la parte superior del mando.

2.2.2.1. DIAGRAMA DEL CIRCUITO PARALELO

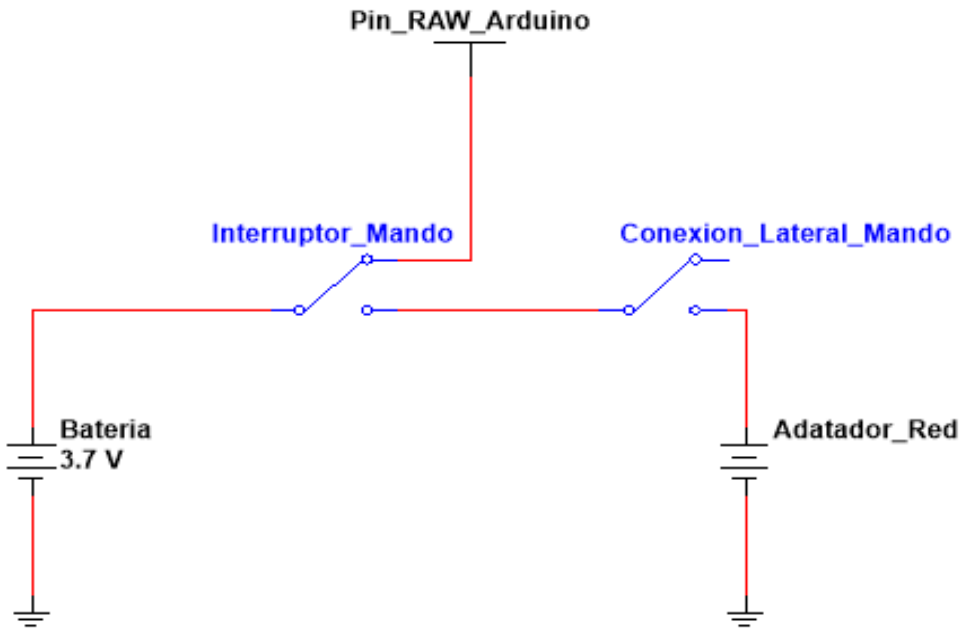

Figura 18. Mando en funcionamiento.

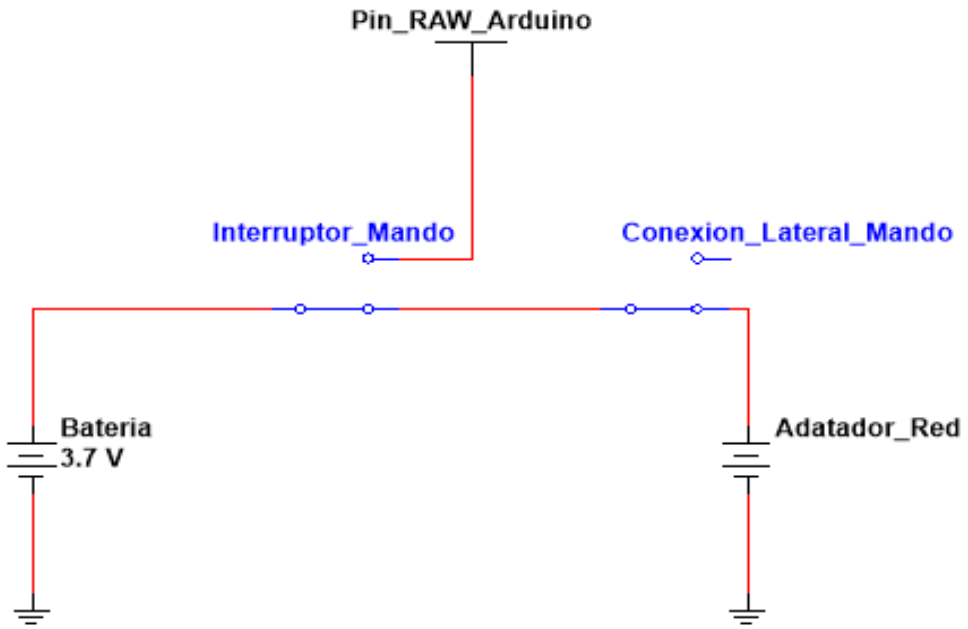

Figura 19. Batería cargando.

2.2.3. EMISOR RF433

Los módulos de comunicación por radiofrecuencia RF433MHz emisor y receptor nos van a permitir conectar dos Arduinos y que estos puedan comunicarse unidireccionalmente.

Para hacerlos funcionar se necesita la librería RadioHead. Esta librería se encarga de gestionar las funciones más básicas de estos módulos de radiofrecuencia: envío de información, comprobación de los paquetes de datos…

Características técnicas:

- Modelo: MX-FS-03V
- Voltaje de alimentación: 3.5-12V.
- Alcance: 20-200 metros dependiendo de los diferentes voltajes de alimentación. Mediante la inclusión de una antena también es posible aumentar el alcance.
- Tasa de transmisión: 4KB/S.
- Potencia de transmisión: 10mW.
- Frecuencia de transmisión: 433Mhz.
- Modulación: ASK/OOK.
- Dimensiones: 19x19mm.

Conexionado:

Tabla 10. Conexiones emisor RF.

Emisor RF	Arduino Pro Mini 3.3V
Pin VCC	Pin VCC
Pin GND	Pin GND
Pin DATA	Pin digital 12

Una vez cargados los programas y en ejecución, conectando el monitor serie de la aplicación Arduino con la placa receptora a 9600 baudios, se debería ver el mensaje que se está emitiendo desde la placa emisora.

Figura 20. Emisor RF.

Extraída de: http://mla-s1-p.mlstatic.com/modulo-rf-transmisor-y-receptor-433-mhz-arduino-robotica-pic-20271-MLA20186866356_102014-F.jpg

2.2.4. MANDO

De acuerdo con el trabajo desarrollado, las conexiones que se deben realizar entre los pines del mando y Arduino (en este caso Arduino Pro Mini 3,3V) son las siguientes:

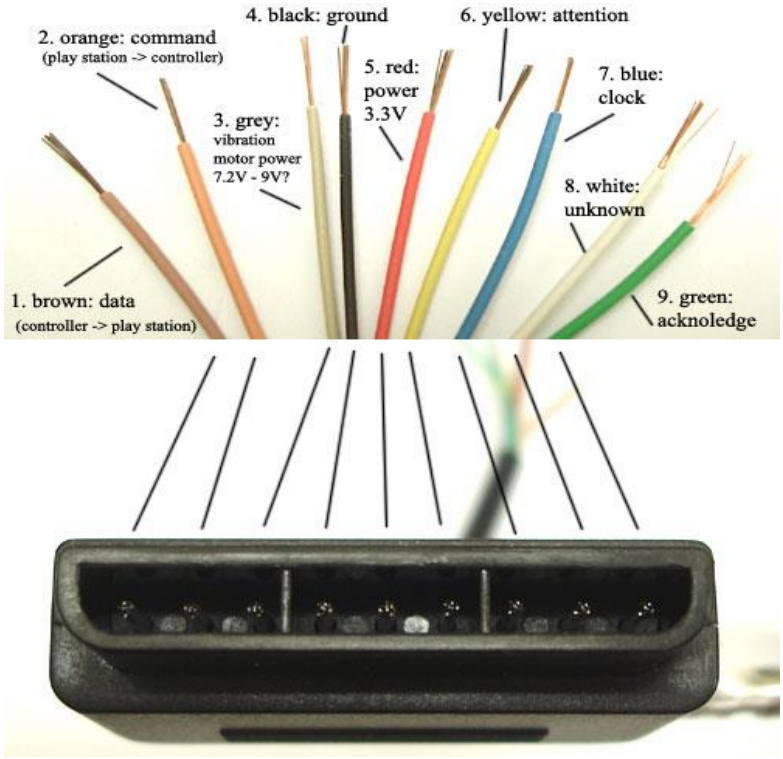

Figura 21. Conexiones mando PS2.

Extraída de: http://store.curiousinventor.com/guides/PS2

Tabla 11. Conexiones mando.

Mando PS2	Arduino Pro Mini 3.3V
Pin1. Data	Pin13
Pin2. Comando	Pin11
Pin4. Tierra	GND
Pin5. Vcc. 3,3V	RAW
Pin6. Attention(send)	Pin10
Pin7. Reloj	Pin9
Pin8	Sin conexión
Pin9	Sin conexión

Aunque el objetivo del trabajo es el funcionamiento del mando de la plataforma de videojuegos PS2 sin el cable, es decir, insertando para ello todos los elementos necesarios en su interior, primero se ha trabajado con el cable y con todos los componentes en el exterior para verificar su correcto funcionamiento.

Figura 22. Funcionamiento del mando con cable.

Una vez todo comprobado, se ha procedido a cortar el cable y realizar la inserción de los componentes en el interior del mando, para ello ha sido necesario abrir el mando, extraer los motores que dispone en sus laterales y en los espacios libres colocar la batería LiPo 100mA, el Arduino Pro Mini 3,3V y el emisor de radiofrecuencia. Además, se ha pelado el cable general del mando y los cables de su interior se han soldado a la placa de Arduino según las conexiones indicadas en la tabla. Ya que la batería se

encuentra en el interior del mando y una vez cerrado este no existe posibilidad de trabajar con ella, se ha abierto un hueco en el lateral del mando que posibilita la carga de batería. Además se ha realizado un circuito paralelo con el objetivo de proteger la placa de sobretensiones durante el proceso de recarga de la batería, a través de un interruptor (el cual conecta/desconecta la batería del Arduino) situado en la parte superior del mando mediante la apertura de un pequeño hueco en la parte superior del mando. Por otra parte la antena soldada al emisor RF433 se extrae por el hueco dejado por el antiguo cable.

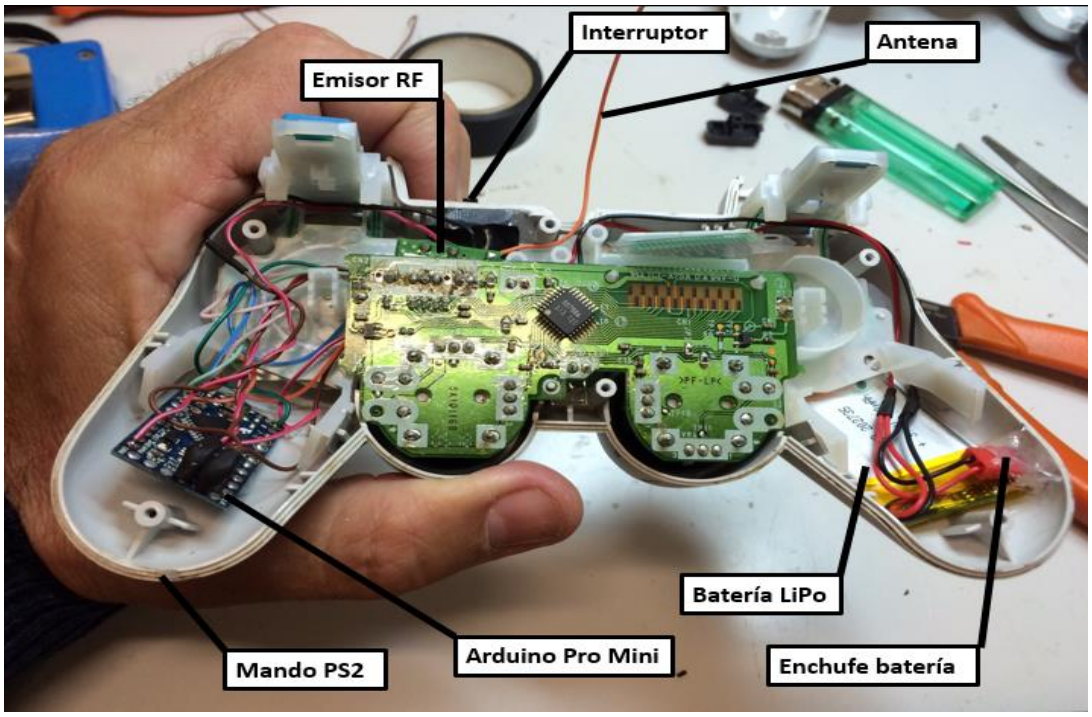

Figura 23. Interior del mando

Figura 24. Interruptor y conexión carga batería.

3. PROGRAMACIÓN

En la sección de programación se dan las explicaciones necesarias para entender el código de programación en Arduino utilizado para el desarrollo del proyecto. El proyecto se encuentra dividido en dos partes, la parte emisora de órdenes o mando y el coche, es decir, la parte controlada o receptora de órdenes. En ambas se programa en Arduino, pero para el mando se utiliza la placa Arduino Pro Mini y para el coche la placa Freaduino UNO.

3.0. EJEMPLO GRÁFICO DEL PROGRAMA

En esta sección se muestra unos ejemplos gráficos que ayudaran a comprender el funcionamiento del programa. Estos ejemplos son el diagrama de flujo general de los programas emisor y receptor unificados y una visión jerárquica de las funciones que componen el programa.

NOTA IMPORTANTE: Hay que tener en cuenta en el diagrama de flujo general que para el funcionamiento de los modos seguidor de línea y funcionamiento en modo autónomo (andar solo) se deben de mantener presionados los botones (SELECT y START).

3.0.1. DIAGRAMA DE FLUJO GENERAL.

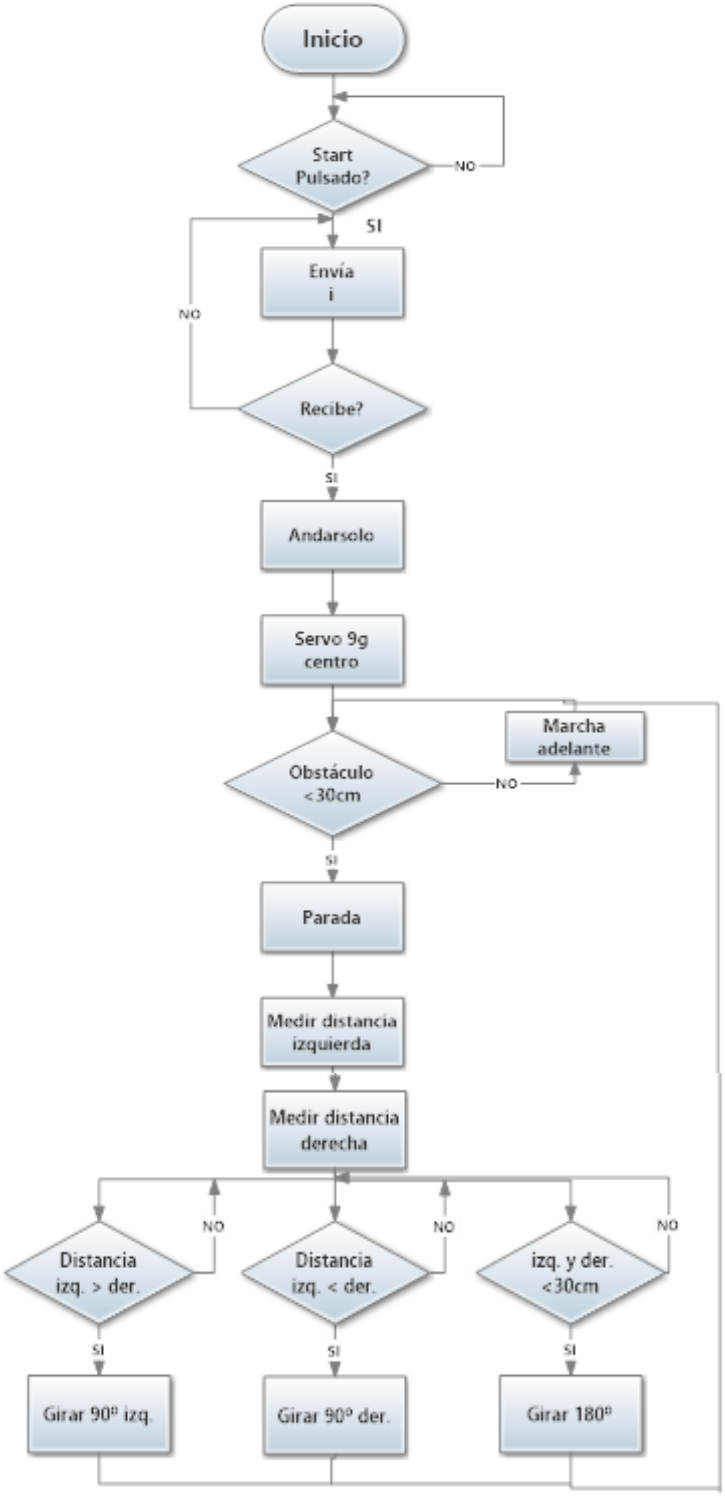

Figura 25. Diagrama de flujo parte 1.

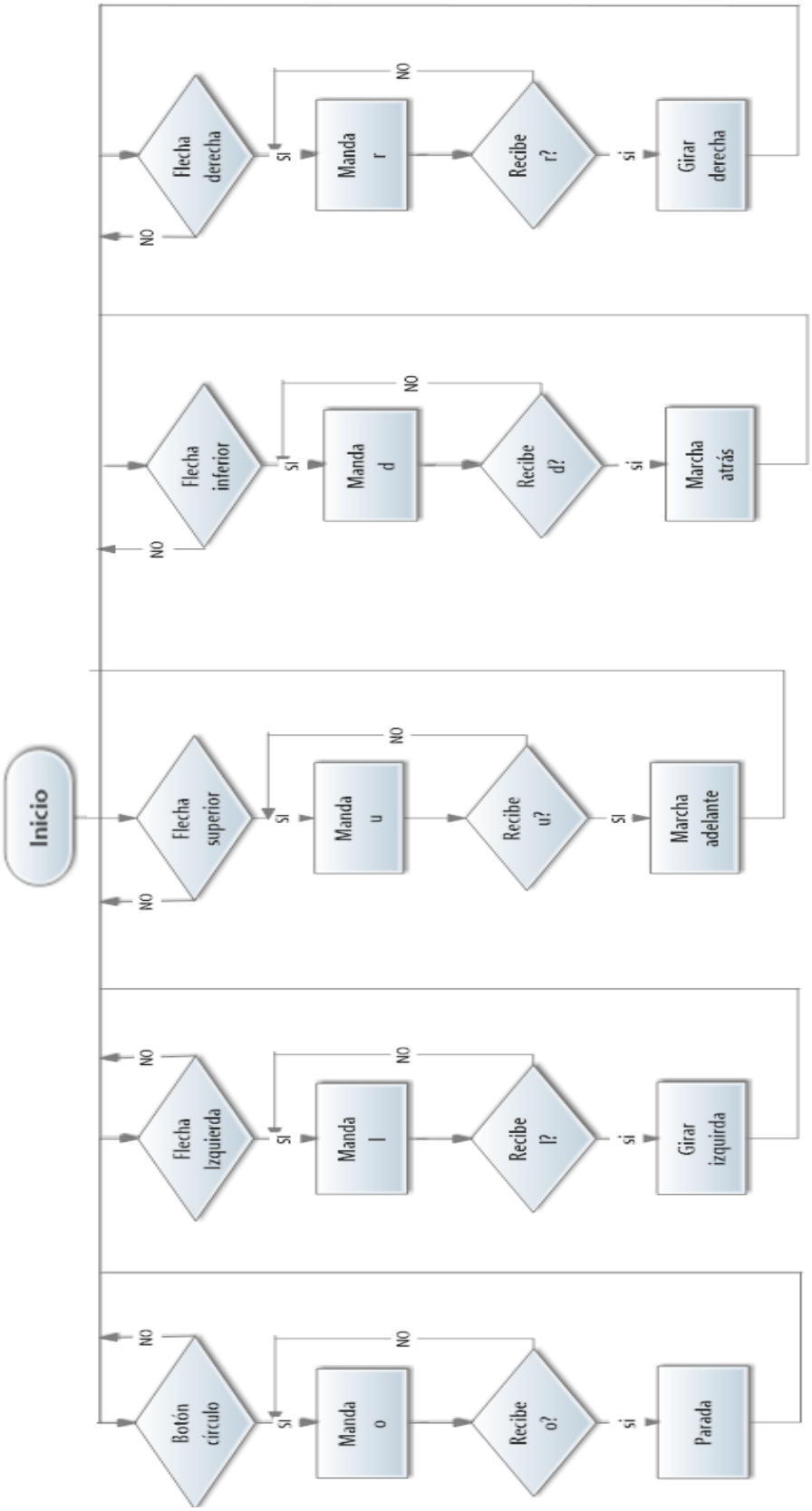

Figura 26. Diagrama de flujo parte 2.

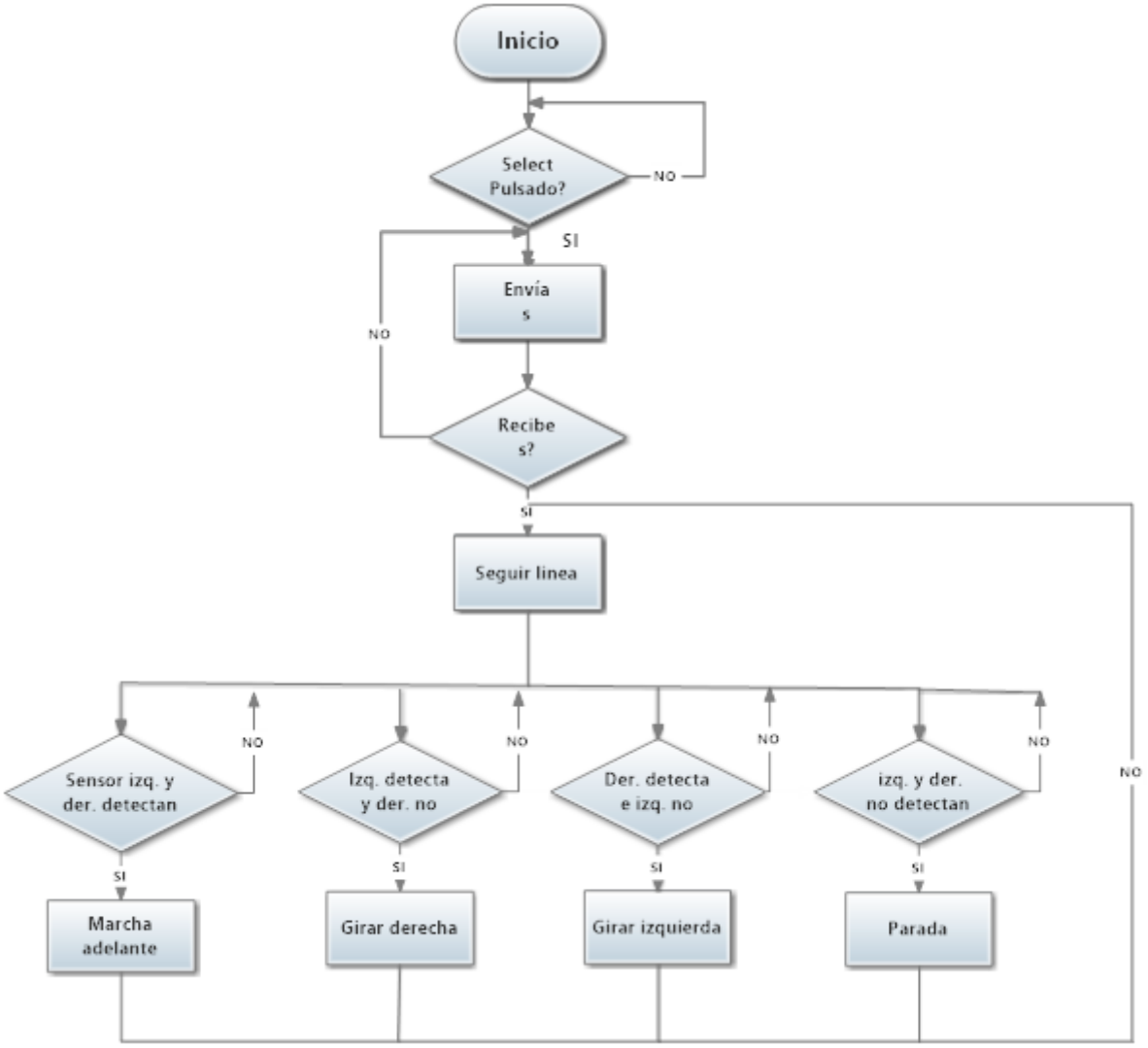

Figura 27. Diagrama de flujo parte 3

3.0.2. JERARQUÍA DE LAS FUNCIONES.

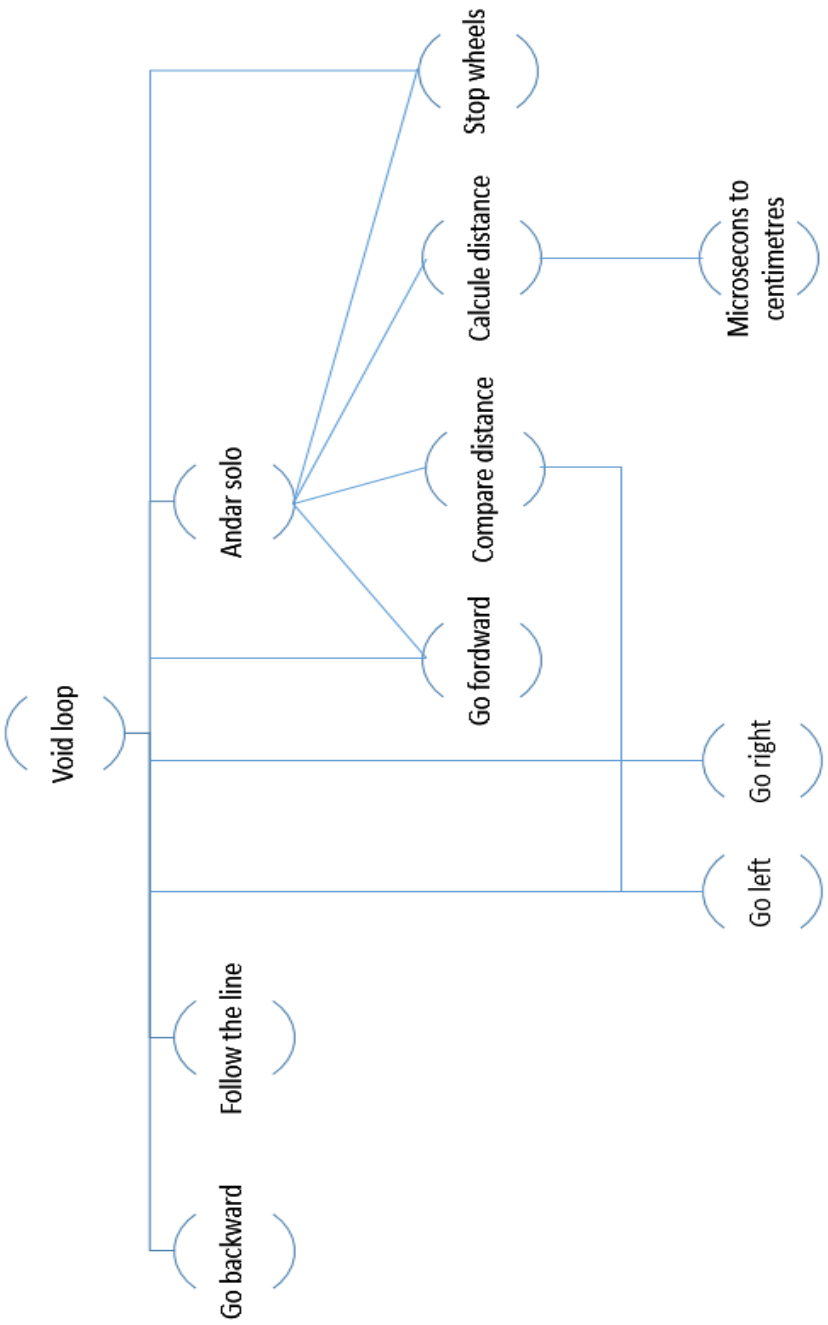

Figura28.Funciones del programa.

Instalación de una librería:

Las librerías ofrecen una funcionalidad mayor al programa. Para insertar una librería se han de seguir los siguientes pasos: **Programa > Importar Librería**.

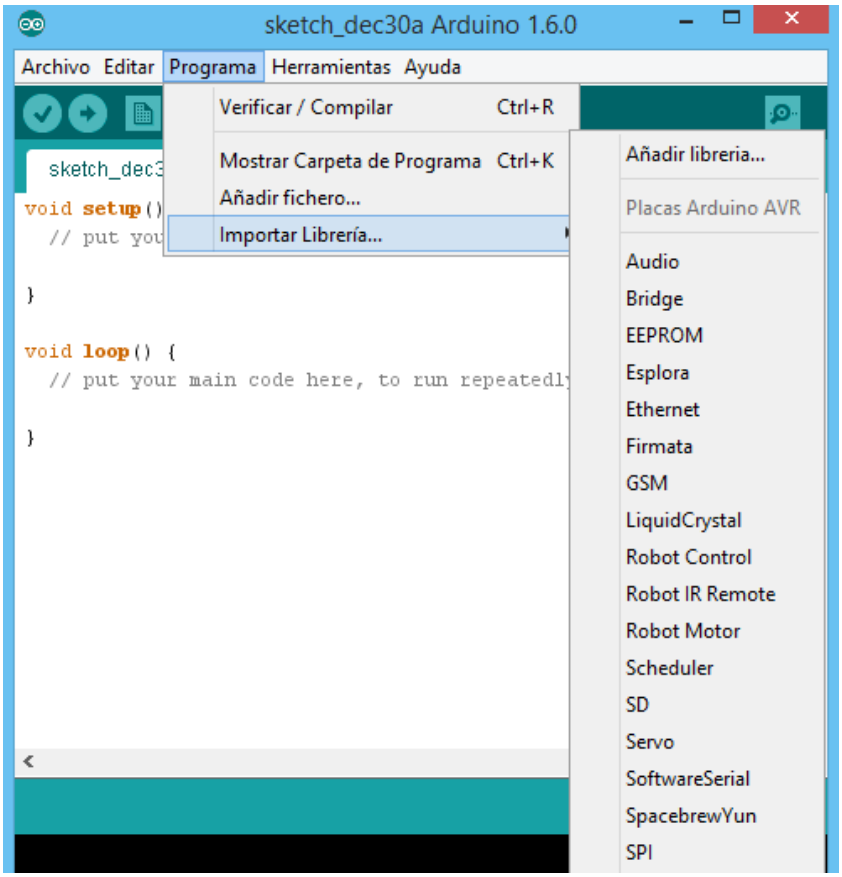

Figura 29. Instalación de librería.

En el caso de que se quiera utilizar una librería existente basta con seleccionar una de las aparece en la lista. Sin embargo, si se desea añadir una librería nueva a la lista, es decir, una que no se encuentra en el software de Arduino, habrá que introducirla de la siguiente manera: **C:\Users\Usuario\Documents\Arduino**

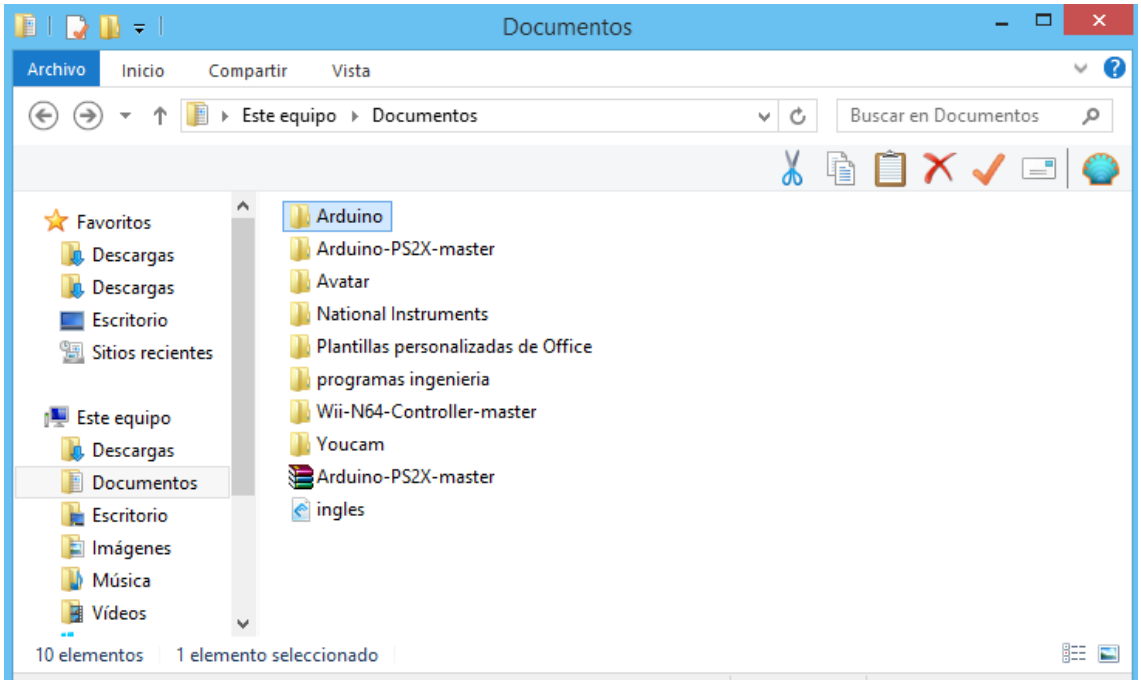

Figura 30. Instalación de librería.

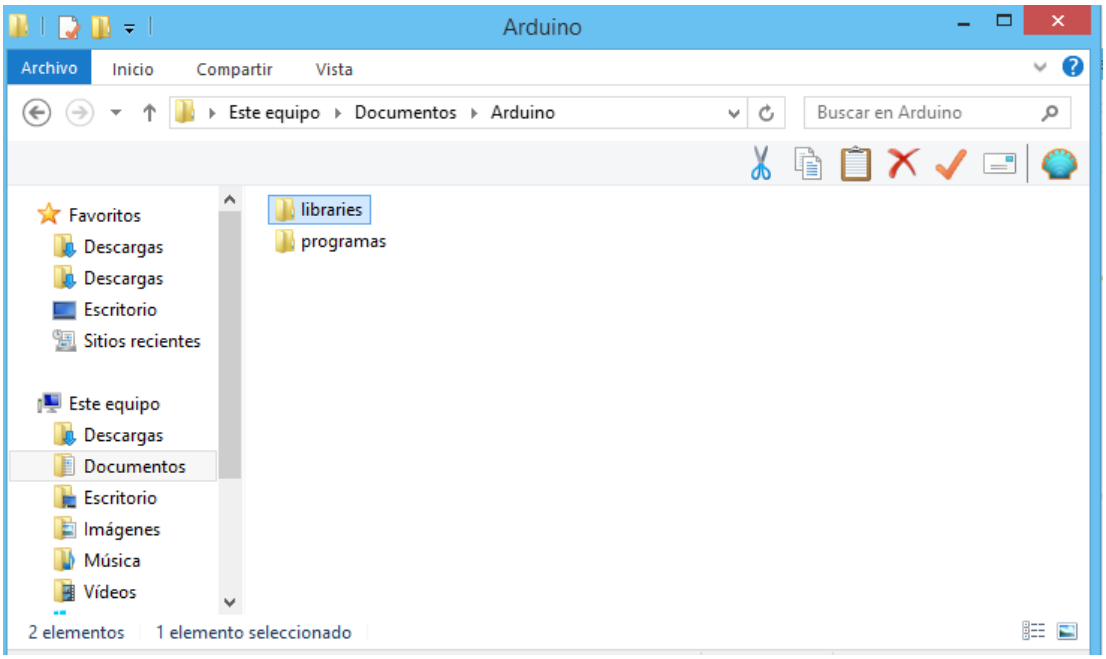

Figura 31. Instalación de librería.

En esta dirección se copiarán las carpetas que contienen a las nuevas librerías de usuario. Una vez realizado este proceso, las librerías introducidas aparecerán en la parte inferior de la lista antes mencionada.

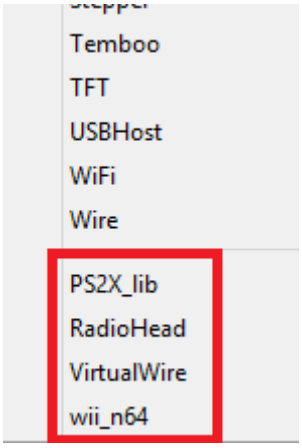

Figura 32. Instalación de librería.

NOTA IMPORTANTE: las librerías de usuario no se deben insertar donde se ha instalado el programa Arduino, sino en la carpeta Documentos del usuario.

3.1. PROGRAMACIÓN DEL MANDO

3.1.1. LIBRERÍAS UTILIZADAS.

Al inicio del código aparecen las librerías utilizadas en este proyecto.

```
#include <PS2X_lib.h>
#include <RH_ASK.h>
#include <SPI.h>
```

Figura 33. Librerías utilizadas.

La librería PS2X permite la comunicación entre el mando de la plataforma de videojuegos PlayStationII y Arduino. Esta librería está creada por Bill Porter y se puede encontrar en el siguiente enlace de internet:

- http://www.billporter.info/2010/06/05/playstation-2-controller-arduino-library-v1-0/

La librería RH_ASK permite la utilización (transmisión recepción de mensajes) de los comunicadores de radiofrecuencia RF433Mhz junto con Arduino. Está disponible para su descarga en el siguiente enlace de internet:

- http://www.airspayce.com/mikem/arduino/RadioHead/

La librería SPI es la única utilizada que viene por defecto instalada en Arduino. El protocolo SPI proviene de las siglas en inglés “Serial Peripheral Interface”, y es un estándar de comunicaciones usado principalmente en la transferencia de información entre circuitos integrados en circuitos electrónicos. Se trata de un bus serie de datos para la transferencia síncrona y bidireccional de información. En toda comunicación por SPI deberá haber al menos un dispositivo actuando como maestro (Arduino), y uno o más actuando como esclavos.

3.1.2. MAPEO DE LOS PINES FÍSICOS EN EL CÓDIGO ARDUINO

```
RH_ASK driver;
#define PS2_DAT        13
#define PS2_CMD        11
#define PS2_SEL        10
#define PS2_CLK        9
#define pressures   true
//#define pressures   false
//#define rumble      true
#define rumble      false
PS2X ps2x; // create PS2 Controller Class

int error = 0;
byte type = 0;
byte vibrate = 0;
```

Figura 34. Mapeo pines físicos

En las primeras líneas de código se definen los pines a los que estará conectado el mando de la PS2, a continuación se habilita el medidor de presión de los botones del mando y se deshabilita la función de vibración del mando. Por último se inicializan a cero variables utilizadas en el programa para el control del mando.

A la hora de hacer funcionar el mando con la librería pueden surgir algún problema, consultar la siguiente página:

- http://www.billporter.info/2011/03/27/arduino-playstation-2-controller-library-troubleshooting-guide/

En nuestro caso el problema en cuestión fue el siguiente:

There's a value in the 'PS2X_lib.h' file that governs the speed of the bus to the controller. It's called 'CTRL_CLK' and you can find it by looking for this line:

#define CTRL_CLK 4

The PlayStation 2 talks to its controllers at 500kHz, or a value of '2' in my library. Arduino tends to have issues setting a value that low, so by default I have it set at 4. **You can try using 2 instead, and I'd also try using some higher number for a slower bus speed. Go from 2-20 and even 200 to see if you can get the controller talking**. Remember to save the .h file every time you edit it, and re-compile the sketch.

Figura 35. Resolución problema.

Figura 7. Indicación del error y posible solución. http://www.billporter.info/2011/03/27/arduino-playstation-2-controller-library-troubleshooting-guide/

```
89  #include <math.h>
90  #include <stdio.h>
91  #include <stdint.h>
92  #ifdef __AVR__
93    // AVR
94    #include <avr/io.h>
95    #define CTRL_CLK        20
96    #define CTRL_BYTE_DELAY 3
97  #else
98    // Pic32...
99    #include <pins_arduino.h>
100   #define CTRL_CLK       20
101   #define CTRL_CLK_HIGH   5
102   #define CTRL_BYTE_DELAY 4
103 #endif
```

Figura 36. Solución aplicada.

3.1.3. FUNCIÓN DE INICIALIZACIÓN (ENTRADAS Y SALIDAS)

```
void setup(){
 Serial.begin(9600);      // Debugging only
 if (!driver.init())
 Serial.println("init failed");
 //setup pins and settings: GamePad(clock, command, attention, data, Pressures?, Rumble?) check for error
 error = ps2x.config_gamepad(PS2_CLK, PS2_CMD, PS2_SEL, PS2_DAT, pressures, rumble);

 if(error == 0){
    Serial.print("Found Controller, configured successful ");
    Serial.print("pressures = ");
         if (pressures)
           Serial.println("true ");
         else
         Serial.println("false");
         Serial.print("rumble = ");
         if (rumble)
           Serial.println("true)");
         else
           Serial.println("false");
    Serial.println("Try out all the buttons, X will vibrate the controller, faster as you press harder;");
    Serial.println("holding L1 or R1 will print out the analog stick values.");
    Serial.println("Note: Go to www.billporter.info for updates and to report bugs.");
  }
  else if(error == 1)
    Serial.println("No controller found, check wiring, see readme.txt to enable debug. visit www.billporter.info


  else if(error == 2)
    Serial.println("Controller found but not accepting commands. see readme.txt to enable debug

  else if(error == 3)
    Serial.println("Controller refusing to enter Pressures mode, may not support it. ");

//  Serial.print(ps2x.Analog(1), HEX);

  type = ps2x.readType();
  switch(type) {
    case 0:
      Serial.print("Unknown Controller type found ");
      break;
    case 1:
      Serial.print("DualShock Controller found ");
      break;
    case 2:
      Serial.print("GuitarHero Controller found ");
      break;
        case 3:
      Serial.print("Wireless Sony DualShock Controller found ");
      break;
   }
}
```

Figura 37. Función de inicialización.

Primero se inicializa el puerto serie a 9600bps para que el programa pueda imprimir por pantalla en el puerto serie aquellos mensajes que se han establecido previamente.

Luego se comprueba todas las funciones del mando conectado, en caso de que alguna de ellas no responda correctamente se mandarán los correspondientes mensajes de error y sus soluciones.

3.1.4. FUNCIÓN PRINCIPAL (LOOP)

Esta parte del código se ejecutará una y otra vez. Su principal función es recoger las órdenes dadas mediante el mando (pulsación de botones), asignar a cada orden una letra indicativa y proceder a enviar dicha letra:

Tabla 12. Letra por botón

Botones del mando	Letra transmitida
Botón START	I
Botón SELECT	S
Botón flecha superior	U
Botón flecha derecha	R
Botón flecha izquierda	L
Botón flecha inferior	D
Botón triángulo	T
Botón círculo	O
Botón aspa	X
Botón cuadrado	C

```
void loop() {

  if(error == 1) //skip loop if no controller found
    return;

  else { //DualShock Controller!!!!
    ps2x.read_gamepad(false, vibrate); //read controller and set large motor to spin at 'vibrate' speed

    if(ps2x.Button(PSB_START)){         //will be TRUE as long as button is pressed
       Serial.println("Start is being held");
       const char *msg = "i";
       driver.send((uint8_t *)msg, strlen(msg));
       driver.waitPacketSent();
       delay(200);
      }

    if(ps2x.Button(PSB_SELECT)){
      Serial.println("Select is being held");
      const char *msg = "s";
      driver.send((uint8_t *)msg, strlen(msg));
      driver.waitPacketSent();
      delay(200);
     }

  if(ps2x.Button(PSB_PAD_UP)) {      //will be TRUE as long as button is pressed
    Serial.print("Up held this hard: ");
    Serial.println(ps2x.Analog(PSAB_PAD_UP), DEC);
    const char *msg = "u";
    driver.send((uint8_t *)msg, strlen(msg));
    driver.waitPacketSent();
    delay(200);
   }

  if(ps2x.Button(PSB_PAD_RIGHT)){
     Serial.print("Right held this hard: ");
     Serial.println(ps2x.Analog(PSAB_PAD_RIGHT), DEC);
     const char *msg = "r";
     driver.send((uint8_t *)msg, strlen(msg));
     driver.waitPacketSent();
     delay(200);
    }

   if(ps2x.Button(PSB_PAD_LEFT)){
     Serial.print("LEFT held this hard: ");
     Serial.println(ps2x.Analog(PSAB_PAD_LEFT), DEC);
     const char *msg = "l";
     driver.send((uint8_t *)msg, strlen(msg));
     driver.waitPacketSent();
     delay(200);
    }
```

```
  if(ps2x.Button(PSB_PAD_DOWN)){
    Serial.print("DOWN held this hard: ");
    Serial.println(ps2x.Analog(PSAB_PAD_DOWN), DEC);
    const char *msg = "d";
    driver.send((uint8_t *)msg, strlen(msg));
    driver.waitPacketSent();
    delay(200);
   }

  if(ps2x.Button(PSB_TRIANGLE)) {
    Serial.println("Triangle pressed");
    const char *msg = "t";
    driver.send((uint8_t *)msg, strlen(msg));
    driver.waitPacketSent();
    delay(200);
   }

 if(ps2x.ButtonPressed(PSB_CIRCLE))  {              //will be TRUE if button was JUST pressed
    Serial.println("Circle just pressed");
    const char *msg = "o";
    driver.send((uint8_t *)msg, strlen(msg));
    driver.waitPacketSent();
    delay(200);
   }


   if(ps2x.NewButtonState(PSB_CROSS)) {             //will be TRUE if button was JUST pressed OR released
      Serial.println("X just changed");
      const char *msg = "x";
      driver.send((uint8_t *)msg, strlen(msg));
      driver.waitPacketSent();
      delay(200);
    }

   if(ps2x.ButtonReleased(PSB_SQUARE))  {           //will be TRUE if button was JUST released
      Serial.println("Square just released");
      const char *msg = "c";
      driver.send((uint8_t *)msg, strlen(msg));
      driver.waitPacketSent();
      delay(200);
     }
  }
  delay(50);
}
```

Figura 38. Función principal.

Para los botones triángulo, círculo, aspa y cuadrado, bastará con una sola presión, mientras que para accionar el resto de botones será necesario mantenerlos pulsados. Para cambiar esta configuración solo habrá que modificar la instrucción (como ejemplo se muestra el botón START):

- "(ps2x.Button(PSB_START))"
- "(ps2x.ButtonPressed(PSB_START))".

A continuación se describe el bloque genérico de código utilizado para el envío de una orden, en este caso se analiza el botón SELECT.

```
if(ps2x.Button(PSB_SELECT)){
  Serial.println("Select is being held");
  const char *msg = "s";
  driver.send((uint8_t *)msg, strlen(msg));
  driver.waitPacketSent();
  delay(200);
 }
```

Figura 39. Bloque genérico de envío de letra por botón.

Al detectar que el botón SELECT ha sido pulsado, el programa imprime por pantalla un mensaje indicando que dicho botón está siendo pulsado y asigna una letra específica para cada botón (en este caso la letra s) que inmediatamente es enviada por el transmisor RF que espera que el mensaje enviado se haya acabado de transmitir.

3.2. PROGRAMACIÓN DEL COCHE

3.2.1. LIBRERÍAS UTILIZADAS.

Al inicio del código aparecen las librerías utilizadas en este proyecto.

```
#include <RH_ASK.h>
#include <SPI.h>
#include <Servo.h>

RH_ASK driver;
```

Figura 40. Librerías utilizadas.

La librería RH_ASK permite la utilización (transmisión recepción de mensajes) de los comunicadores de radiofrecuencia RF433Mhz junto con Arduino, en este caso no se utiliza la librería VirtualWire ya que no es compatible con la librería Servo porque ambas utilizan el mismo TIMER 1. La librería RadioHead está disponible para su descarga en el siguiente enlace de internet:

- http://www.airspayce.com/mikem/arduino/RadioHead/

La librería SPI es la única utilizada que viene por defecto instalada en Arduino. El protocolo SPI proviene de las siglas en inglés “Serial Peripheral Interface”, y es un estándar de comunicaciones usado principalmente en la transferencia de información entre circuitos integrados en circuitos electrónicos. Se trata de un bus serie de datos para la transferencia síncrona y bidireccional de información. En toda comunicación por SPI deberá haber al menos un dispositivo actuando como maestro (Arduino), y uno o más actuando como esclavos.

La librería Servo facilita el trabajo con los servomotores del proyecto. Se utilizará para dar movimiento tanto a las ruedas (izquierda y derecha) como al girador del ultrasonido detector de presencia.

3.2.2. DEFINICIÓN DE VARIABLES

En esta parte del código se definen los pines del Freaduino correspondientes a los dos servos que controlan las dos ruedas así como los correspondientes a los dos sensores infrarrojos y los pines del sensor ultrasonido (detector de proximidad).

También se definen los valores que habrá que proporcionar a los servomotores de rotación continua para que giren en un sentido u otro o permanezcan inmóviles.

Por último se declaran variables necesarias para los servomotores y los sensores infrarrojos.

```
/* Pin definition of the board to be used */
#define pinLeftWheel            10
#define pinRightWheel           13
#define pinSensorIRLeft         7   /*   Left infrared sensor     */
#define pinSensorIRRight        6   /*   Right infrared sensor    */
#define echoPin 9
#define trigPin 8


/* Definition of the values that can take continuous rotation servo,
 that is, the wheels */
#define wheelStopValue            90
#define leftWheelFordwardValue    0
#define leftWheelBackwardsValue   180
#define rightWheelFordwardValue   180
#define rightWheelBackwardsValue  0


/* Size of the received data buffer */
#define bufferSize 1


/* Default delay */
#define defaultDelay        10


 /* Variable that controls the current state of the program */
 int currentState;
 int leftDistance, rightDistance; //distances on either side

 /* A object from the Servo class is created for each servo */
 Servo leftWheel;                      /*  Values from 0 to 180  */
 Servo rightWheel;                     /*  Values from 0 to 180  */
 Servo panMotor;

 /* Variables of the line follower mode */
 int rightIR;
 int leftIR;
 int BLACK = 0;
 int WHITE = 1;
```

Figura 41. Definición de variables.

3.2.3. DEFINICIÓN DE FUNCIONES

3.2.3.1. MOVIMIENTO RUEDAS

En este apartado se definen las funciones a las que se llamarán posteriormente en el programa principal. La existencia de funciones servirá de ayuda para hacer el programa más modular, de forma que se pueda visualizar, comprender y mantener mejor el programa principal.

```
void stopWheels() {
  leftWheel.write(wheelStopValue);
  delay(defaultDelay);
  rightWheel.write(wheelStopValue);
  delay(defaultDelay);
}

void goForwards() {
  leftWheel.write(leftWheelFordwardValue);
  delay(defaultDelay);
  rightWheel.write(rightWheelFordwardValue);
  delay(defaultDelay);
}

void goBackwards() {
  leftWheel.write(leftWheelBackwardsValue);
  delay(defaultDelay);
  rightWheel.write(rightWheelBackwardsValue);
  delay(defaultDelay);
}

void goLeft() {
  leftWheel.write(wheelStopValue);
  delay(defaultDelay);
  rightWheel.write(rightWheelFordwardValue);
  delay(defaultDelay);
}
```

Figura 42. Definición de funciones.

El tipo de función Void únicamente ejecutará las líneas de código de las que está formada, no devolverá ningún valor al programa principal.

Las funciones arriba mencionadas actuarán sobre los servomotores de rotación continua de las ruedas, transmitiendo las órdenes de parar ruedas, avanzar, retroceder y girar a la izquierda y a la derecha. Los giros a izquierda y derecha se realizan moviendo una rueda adelante y manteniendo la otra rueda parada, mientras que las funciones avanzar y retrocedes se realizan accionando ambas ruedas.

A continuación se describe el bloque genérico de código utilizado para el control de las ruedas (en este caso el parado).

```
void stopWheels() {
  leftWheel.write(wheelStopValue);
  delay(defaultDelay);
  rightWheel.write(wheelStopValue);
  delay(defaultDelay);
}
```

Figura 43. Bloque genérico control ruedas.

Llamada en el programa la función determinada (en este caso la función stopWheels), transmite el valor previamente establecido al servo indicado gracias a la librería Servo. A continuación espera un tiempo determinado previamente y hará lo mismo con el otro servo correspondiente a la rueda derecha y espera igual que antes un tiempo previamente determinado.

3.2.3.2. SEGUIR DE LÍNEA

La función followTheLine es la utilizada para lograr que el coche pueda seguir un línea negra, para ello el programa se sirve de la información de los sensores infrarrojos para accionar en según qué caso la rueda izquierda o derecha

```
void followTheLine() {
  /* Read the state of the sensors */
  rightIR = digitalRead(pinSensorIRLeft);
  leftIR = digitalRead(pinSensorIRRight);
  if (rightIR == BLACK) {
    leftWheel.write(leftWheelFordwardValue);
    delay(defaultDelay);
  }
  else {
    leftWheel.write(wheelStopValue);
    delay(defaultDelay);
  }
  if (leftIR == BLACK) {
    rightWheel.write(rightWheelFordwardValue);
    delay(defaultDelay);
    }
  else {
    rightWheel.write(wheelStopValue);
    delay((defaultDelay));
  }
}
```

Figura 44. Función modo seguidor de línea.

3.2.3.3. COMPARA DISTANCIAS

La función compareDistance compara las distancias medidas a izquierda y derecha, en el caso de que una de ellas sea mayor girará el coche hacia ese lado (el de mayor distancia) durante 0,5seg (aproximadamente 90o). Si ambas distancias son igualmente no válidas, el coche procederá a dar media vuelta.

```
void compareDistance() {
  if (leftDistance>rightDistance) { //if left is less obstructed
    goLeft();
    delay(500);
  }
  else if (rightDistance>leftDistance) { //if right is less obstructed
    goRight(); //turn right
    delay(500);
  }
  else {//if they are equally obstructed
    leftWheel.write(0);
    rightWheel.write(180); //turn 180 degrees
    delay(1000);
  }
}
```

Figura 45. Función comparar distancia.

3.2.3.4. CONVERSORA

Esta función convierte los microsegundos que ha tardado la onda ultrasónica en ir y volver (del obstáculo al sensor) a centímetros, posteriormente los deberá dividir entre dos para descartar el tiempo de vuelta.

```
long microsecondsToCentimeters(long microseconds) {
  // La velocidad del sonido a 20° de temperatura es 340 m/s o
  // 29 microsegundos por centrimetro.
  // La señal tiene que ir y volver por lo que la distancia a
  // la que se encuentra el objeto es la mitad de la recorrida.
  return microseconds / 29 / 2 ;
}
```

Figura 46. Función conversora.

3.2.3.5. CALCULA DISTANCIA

Esta función activa el sensor ultrasónico y proporciona con la ayuda de la función anterior la distancia en pantalla.

```
long calculaDistancia() {
  long cm = 0;
  long duration = 0;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  cm = microsecondsToCentimeters(duration);
  Serial.print ("Milisegundos: ");
  Serial.print(duration);
  Serial.print("  Distancia estimada: ");
  Serial.print(cm);
  Serial.println(" cm");
  return cm;
}
```

Figura 47. Función calcular distancia.

3.2.3.6. MODO AUTÓNOMO

```
void andarsolo() {
  panMotor.write(65); //set PING))) pan to center
  long cm=0;
  cm = calculaDistancia();
  delay(500);
  if(cm < 30){
    Serial.println("A MENOS DE 30CM----------------------------------------");
    stopWheels();

    panMotor.write(10);
    delay(2000);
    cm = calculaDistancia();
    rightDistance = cm; //scan to the right
    Serial.print("DISTANCIA A LA DERECHA: "); Serial.println(rightDistance);

    panMotor.write(140);
    delay(2000);
    cm = calculaDistancia();
    leftDistance = cm;
    Serial.print("DISTANCIA A LA IZQUIERDA: "); Serial.println(leftDistance);

    panMotor.write(65); //return to center
    compareDistance();
    Serial.println("SALIENDO DE A MENOS DE 30CM----------------------------------------");
  }

   else {
     goForwards();
   }
}
```

Figura 48. Función modo autónomo.

Gracias a esta función el coche será capaz de tener un funcionamiento autónomo evitando obstáculos, para ello, se servirá del sensor de proximidad ultrasónico y de un servomotor unido al sensor que lo hace girar a izquierda y derecha.

En caso de no detectar obstáculo alguno a menos de 30cm en su frontal, el coche continua su avance hacia adelante hasta encontrarse con un obstáculo, entonces, el coche se parará y procede a medir la distancia hasta el obstáculo más cercano tanto a izquierda como a derecha girando el sensor de proximidad con el servomotor para después comparar ambas distancias y en el caso de que una de ellas sea mayor girará el coche hacia ese lado (el de mayor distancia) durante 0,5seg (aproximadamente 90o). Si ambas distancias son igualmente no válidas, el coche procederá a dar media vuelta y continuar su marcha hacia adelante hasta volver a encontrar un obstáculo cercano.

3.2.4. FUNCIÓN INICIALIZADORA (ENTRADAS Y SALIDAS)

En esta parte del programa se asignan los pines correspondientes a los dos servomotores de rotación continua (ruedas), al servomotor central que permite el giro del sensor de proximidad y se da la orden de parar el coche para iniciar de este modo. También se establecen como pin de salida el pin de Arduino conectado al triger del ultrasonido y como pin de entrada el “echo” del ultrasonido, además los pines de la placa conectados a los sensores infrarrojos se establecen como pines de entrada.

Después se inicializa el puerto serie a 9600bps para que el programa pueda imprimir por pantalla en el puerto serie los mensajes requeridos.

```
void setup()
{
  leftWheel.attach(pinLeftWheel);
  rightWheel.attach(pinRightWheel);
  panMotor.attach(12); //attach motors to proper pins
  stopWheels();/* The robot is stopped at the beginning */

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(pinSensorIRLeft, INPUT);
  pinMode(pinSensorIRRight, INPUT);

  Serial.begin(9600);    // Debugging only
  if (!driver.init())
  Serial.println("init failed");
}
```

Figura 49. Función inicializadora.

3.2.5. FUNCIÓN PRINCIPAL

```
void loop()
{

 uint8_t buf[RH_ASK_MAX_MESSAGE_LEN];
 uint8_t buflen = sizeof(buf);

    if (driver.recv(buf, &buflen)){ // Non-blocking
          int i;
          driver.printBuffer("Got:", buf, buflen);
          for (i = 0; i < buflen; i++) {
              if((char)buf[i]=='u') {
                    goForwards();
                    Serial.print(buf[i]);
                    Serial.print(" ");
                }
              Serial.println("");
              if((char)buf[i]=='d') {
                    goBackwards();
                    Serial.print((char)buf[i]);
                    Serial.print(" ");
                }
              Serial.println("");
                if((char)buf[i]=='o') {
                    stopWheels();
                    Serial.print((char)buf[i]);
                    Serial.print(" ");
                }

             Serial.println("");
             if((char)buf[i]=='l'){
                   goLeft();
                   Serial.print((char)buf[i]);
                   Serial.print(" ");
               }
             Serial.println("");
             if((char)buf[i]=='r') {
                   goRight();
                   Serial.print((char)buf[i]);
                   Serial.print(" ");
               }
             Serial.println("");
             if((char)buf[i]=='s') {
                   uint8_t buf[RH_ASK_MAX_MESSAGE_LEN];
                   uint8_t buflen = sizeof(buf);
                   while( !(driver.recv(buf, &buflen)) && (char)buf[i]!='o')
                     followTheLine();

                   Serial.print((char)buf[i]);
                   Serial.print(" ");
               }
```

Figura 50. Función principal.

Esta parte del código se ejecutará una y otra vez. En el momento de recibir un mensaje enviado por el emisor de radiofrecuencia, se procede a identificar su contenido para posteriormente dependiendo del mensaje transmitido (pulsando diferentes botones del mando) asignarle su acción correspondiente:

Tabla 13. . Acciones correspondientes a botón pulsado

Botones del mando	Letra transmitida	Acción correspondiente
Botón START	I	Modo andar solo
Botón SELECT	S	Modo seguidor de línea
Botón flecha superior	U	Marcha adelante
Botón flecha derecha	R	Giro derecha
Botón flecha izquierda	L	Giro izquierda
Botón flecha inferior	D	Marcha atrás
Botón triángulo	T	Sin acción
Botón círculo	O	Ruedas paradas
Botón aspa	X	Sin acción
Botón cuadrado	C	Sin acción

Para los botones triángulo, círculo, aspa y cuadrado, bastará con una sola presión, mientras que para accionar el resto de botones será necesario mantenerlos pulsados.

A continuación se describe el bloque genérico de código utilizado para la recepción de letra (en este caso la letra d):

```
Serial.println("");
if((char)buf[i]=='d') {
      goBackwards();
      Serial.print((char)buf[i]);
      Serial.print(" ");
  }
```

Figura 51. Recepción de letra.

Detectada la recepción de una letra establecida, el programa pasa a llamar a la función que se le ha asignado para el pulso del botón. Acto seguido se imprime por pantalla la letra recibida y un espacio en blanco.