UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Desarrollo de un pequeño robot móvil basado en microcontrolador

Proyecto Final de Carrera

Ingeniería Técnica de Informática de Sistemas

Autor: Javier Cilla Mateu

Director: José Vicente Busquets Mataix

Septiembre de 2014

Resumen

Este proyecto de fin de carrera consiste en la programación y construcción de un pequeño robot móvil autónomo basado en una tarjeta basada en microcontrolador de la familia Arduino. A lo largo del proyecto explicaremos los diferentes requisitos tanto de software como de hardware necesarios para la realización del proyecto, el diseño mismo del robot, las herramientas y el entorno utilizados, el desarrollo de la programación y el hardware y las pruebas realizadas para el perfecto funcionamiento del robot.

El robot que vamos a construir es un pequeño robot móvil que evita los obstáculos. A través de unos sensores infrarrojos, detectará la distancia a los objetos que tiene frente a sí. Cuando se acerque a una determinada distancia, el microcontrolador mandará una señal a los motores para que haga girar al robot y así evitar el obstáculo.

La evasión de obstáculos es una característica fundamental para que se puedan construir otros robots que realicen funciones más complejas sin quedarse atorados en medio de su trabajo por culpa de algún obstáculo que hayan encontrado en el camino, y permite acercarse a la automatización completa del robot sin la necesidad de un control remoto sobre este.

Palabras clave: Arduino, robot móvil, microcontrolador, servomotor, sensor de infrarrojos, detección de obstáculos

Índice

1. Introducción

La robótica surgió por el deseo humano de hacer máquinas a su semejanza y que lo descargaran de trabajo. Hoy en día, los robots más extendidos son los de uso industrial. Éstos suelen ser estáticos y estar instalados en cadenas de montaje donde no paran de realizar siempre una función específica.

La robótica móvil es un campo que está bastante virgen y en auge, ya que hay muchos trabajos que pueden ser realizados por robots pero requieren la capacidad de movimiento que no tienen los robots estáticos.

En este proyecto vamos a centrarnos en uno de los rasgos que deben poseer estos robots móviles para poder realizar su trabajo eficientemente, la evasión de obstáculos.

Hay tareas, como la exploración o recolección en terrenos agrestes, o campos en los que se está trabajando, como la automatización de vehículos, en los que la evasión de obstáculos puede ser determinante.

Aunque rascando sólo su superficie, en este proyecto se trata de dar un paso más para conseguir la automatización de la evasión de obstáculos.

1.1 Descripción de los objetivos del proyecto

La finalidad de este proyecto final de carrera es construir un robot móvil que evite los obstáculos que encuentre a su paso.

Mediante la manufactura del chasis y la programación del microcontrolador Arduino, el robot tendrá la capacidad de evitar todo tipo de obstáculos y así poder continuar moviéndose sin chocar con cualquier objeto que se encuentre en su camino.

1.2 Contenido de la memoria

En esta memoria se explica paso a paso el desarrollo del proyecto final de carrera. Y para ello se ha dividido el contenido en 8 apartados principales, intentando así que la información quede lo más ordenada posible y sea fácil de seguir.

El primer apartado es una introducción que habla del objetivo del proyecto y el principal rasgo del robot que vamos a construir, la evasión de obstáculos. El punto actual se incluye dentro de dicho apartado dedicado a la introducción.

El segundo apartado realiza un análisis de los requisitos tanto a nivel hardware como software que ha de tener nuestro robot.

En el tercer apartado realizaremos el diseño físico del robot y de la programación necesaria para la consecución de nuestros objetivos.

En el cuarto apartado expondremos las herramientas utilizadas para la construcción del robot como para su programación.

El quinto es el más importante de los apartados, ya que en él figura la explicación para construir el robot. Tanto el proceso seguido para cortar las piezas necesarias para crear la estructura y ensamblar los diferentes componentes hardware como la explicación del código realizado para controlar el comportamiento de nuestro autómata móvil.

Una vez desarrollado todo el trabajo, y ya teniendo el robot completamente operativo, pasamos a realizar pruebas de su comportamiento y el de sus componentes. Estos tests y las correcciones realizadas para el correcto funcionamiento del robot se encuentran en el apartado sexto.

En el penúltimo apartado, el séptimo, se indicarán una serie de conclusiones acompañadas de posibles ámbitos de aplicación de este proyecto y también posibles mejoras o ampliaciones que se podrían implementar en un futuro.

Por último, en el octavo apartado se encuentran las referencias utilizadas y el índice de imágenes y código de la memoria.

2. Análisis

Para la construcción del robot hemos de tener en cuenta dos aspectos fundamentales: los requerimientos de hardware y los de software.

2.1 Análisis de requerimientos de Hardware

Si queremos construir un pequeño robot móvil, primero hemos de idear que clase de vehículo queremos construir.

Para ello, aunque podríamos haber elegido entre diferentes tipos de chasis (4 ruedas, ruedas oruga...), hemos elegido un chasis con dos ruedas traseras bidireccionales y una rueda delantera omnidireccional. Esto, además de conferirle una mayor movilidad al robot, supone un ahorro de energía importante, ya que en vez de tener que alimentar cuatro servomotores (chasis de 4 ruedas) o tener unas ruedas más pesadas (chasis oruga) sólo tenemos que alimentar dos servomotores (la rueda omnidireccional es puramente mecánica, no necesita alimentación). Además de los requisitos de diseño, para la construcción del chasis vamos a utilizar metacrilato, que es muy ligero.

Una vez definido el chasis, vamos a hablar de los componentes hardware necesarios. Como ya hemos comentado anteriormente, necesitaremos tres ruedas, dos bidireccionales y una omnidireccional. Para el funcionamiento de las ruedas bidireccionales necesitaremos dos servomotores (uno para cada una).

Para la detección de los obstáculos instalaremos dos sensores de infrarrojos frontales. Estos sensores tienen la función de enviar una señal indicando la distancia a la que se encuentran los objetos que el robot tiene en frente suyo.

También necesitaremos una base para baterías para alimentar el microcontrolador y el resto de componentes del robot.

Por último, el centro alrededor del cual se crea el robot es la placa Arduino UNO. Esta placa es la encargada de distribuir la alimentación por los diferentes componentes hardware que lo necesiten y recibir las señales de los sensores infrarrojos y enviarla a los servomotores. Para recibir y

enviar las señales necesarias, esta placa cuenta con el microcontrolador donde se carga el programa que gestiona las funciones que ha de realizar el robot en general y cada componente en particular.

2.2 Análisis de requerimientos de Software

Habiendo detallado ya los requerimientos de hardware necesarios para la construcción del robot, pasamos ahora a analizar el programa que vamos a tener que implementar para su correcto funcionamiento.

Las funciones principales que tiene que realizar el robot son: detectar los obstáculos, moverse hacia delante y girar para evitar los obstáculos que encuentre a su paso.

La función principal que realiza el robot es la de detección. El robot siempre tiene que estar detectando obstáculos, los haya o no. Para ello el robot siempre tiene que estar leyendo las señales que envían los sensores infrarrojos. Si no encuentra detecta ningún obstáculo, sigue moviéndose hacia adelante y si lo detecta gira hacia un lado u otro, dependiendo del sensor que haya localizado el obstáculo.

Una vez localizado un obstáculo, el robot debe de girar hacia el lado donde contrario al que ha localizado el obstáculo hasta que ninguno de los dos sensores detecta ningún obstáculo.

Cuando el robot tiene el camino libre deberá moverse hacia adelante hasta que vuelva a detectar algo que le bloquee el paso.

3. Diseño

El diseño del robot se compone de dos partes claramente diferenciadas. El diseño físico del robot, que comprende el chasis y los componentes hardware utilizados, y el diseño lógico, en el cual hablaremos de la estructuración del programa realizado para el funcionamiento del robot.

3.1 Diseño físico

Para construir el pequeño robot móvil lo primero que tenemos que tener en cuenta es el diseño de un chasis en el que podamos instalar todos los componentes necesarios para su funcionamiento. Por este motivo no podemos escoger unas dimensiones demasiada pequeñas, ya que no podríamos añadir las distintas piezas que requiere el robot, ni demasiado grande, ya que supondría una pérdida inútil de recursos.

Como ya comentamos en el apartado "2.1 Análisis de requerimientos de Hardware", los componentes necesarios para la construcción del robot son dos servomotores, dos ruedas bidireccionales (conectadas a los servomotores, por lo tanto no influyen en el diseño del chasis), una rueda omnidireccional, una base para baterías, una placa con el microcontrolador y dos sensores de infrarrojos delanteros.

La distribución de las piezas en el chasis es la siguiente:

Los servomotores se instalarán en la parte lateral del chasis.

Los sensores de infrarrojos irán ubicados en la parte frontal externa.

La rueda omnidireccional se añadirá en la parte frontal interna.

La base para baterías y el microcontrolador se instalarán en la base principal del chasis.

El chasis estará compuesto de la base, una pieza frontal, una pieza trasera y dos piezas laterales.

Teniendo ya la distribución de los diferentes componentes realizamos el plano del chasis del robot que queda como sigue:

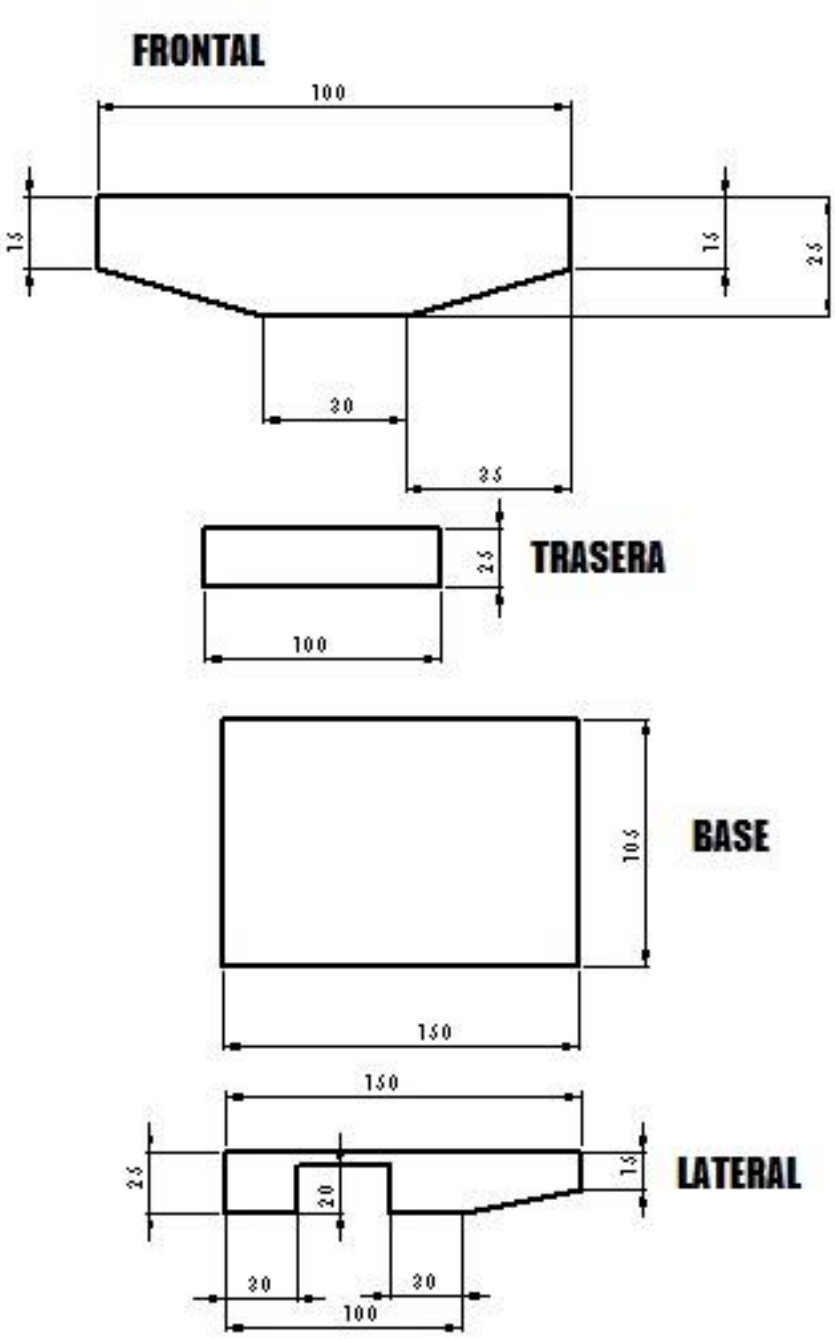


Imagen 1. Piezas del chasis del robot.

Como podemos ver en la Imagen 1(medidas en mm), las piezas laterales tienen un hueco para poder introducir los servomotores.

En el siguiente boceto vemos como quedaría el chasis con las tres ruedas y los sensores montados en él.

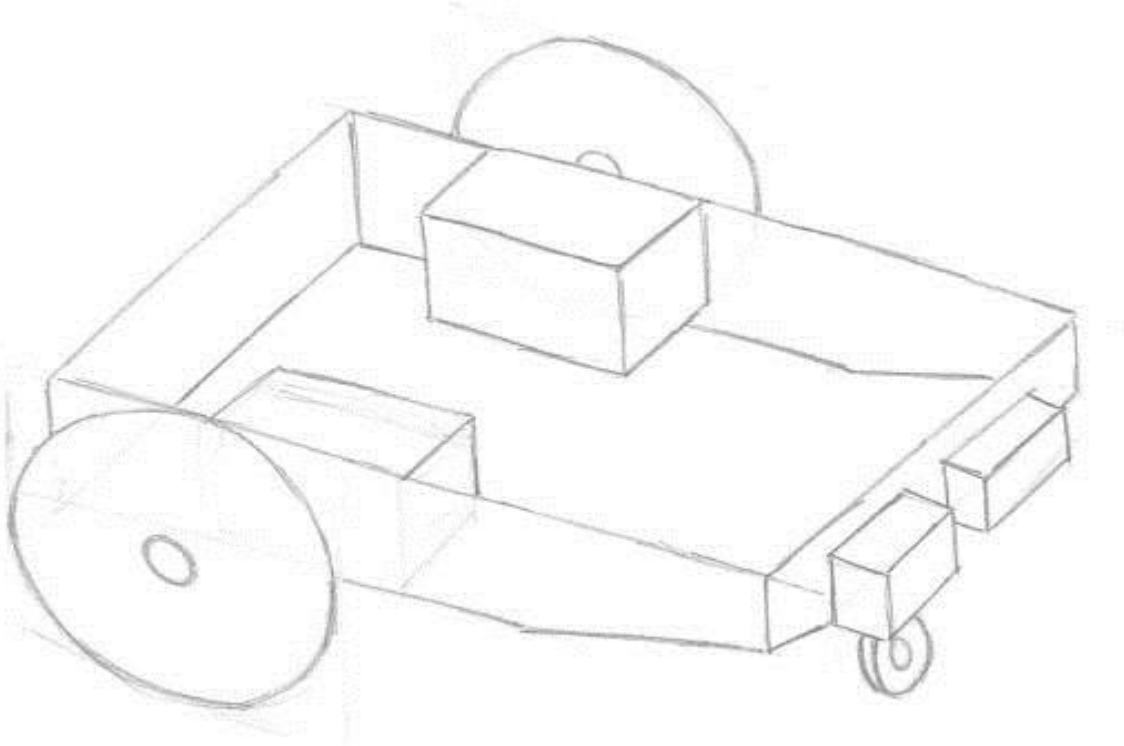


Imagen 2. Chasis con ruedas y sensores.

3.2 Diseño lógico

Una vez analizados los requerimientos del programa que tenemos que crear para conseguir el comportamiento deseado del robot, hemos de proceder a su diseño.

Todo programa realizado utilizando el lenguaje Arduino tiene una función "setup", donde indicamos las señales de entrada y salida, y un bucle principal que contiene las funciones primarias del programa.

A partir de este bucle es desde donde diseñaremos el programa del robot.

Las dos funciones principales son las de "detectar" y "adelante".

La función "detectar" nos dirá si encuentra un obstáculo o no, y si lo hace, cuál es el sensor que detecta el obstáculo.

La función "adelante" gestionará los datos recibidos de la función detectar. Si detectar no encuentra ningún obstáculo, "adelante" llamará a la función "dosmotor" con los parámetros necesarios para que el robot siga su camino hacia adelante. Si encuentra un objeto a la derecha o a la izquierda llamará a la función "derecha" o "izquierda", según dónde esté el obstáculo, que harán girar el robot hasta que no haya ningún obstáculo y pueda seguir la marcha.

Las funciones "derecha" e "izquierda" hacen que las ruedas giren hacia el lado correspondiente y llaman a "detectar" para comprobar si sigue habiendo obstáculos o no. Para que el robot no entre en un bucle de movimiento, seguirá girando hacia el mismo lado hasta que ninguno de los dos sensores detecte obstáculos en el camino. Si no fuera así, el robot podría entrar en un punto muerto, en el que detectara un obstáculo a un lado, comenzara a girar hacia el lado contrario, dejara de detectar el obstáculo y detectara otro en el lado contrario, por lo que giraría en sentido contrario y volvería a detectar el primer obstáculo, lo que conllevaría que girase hacia ambos lados sin parar.

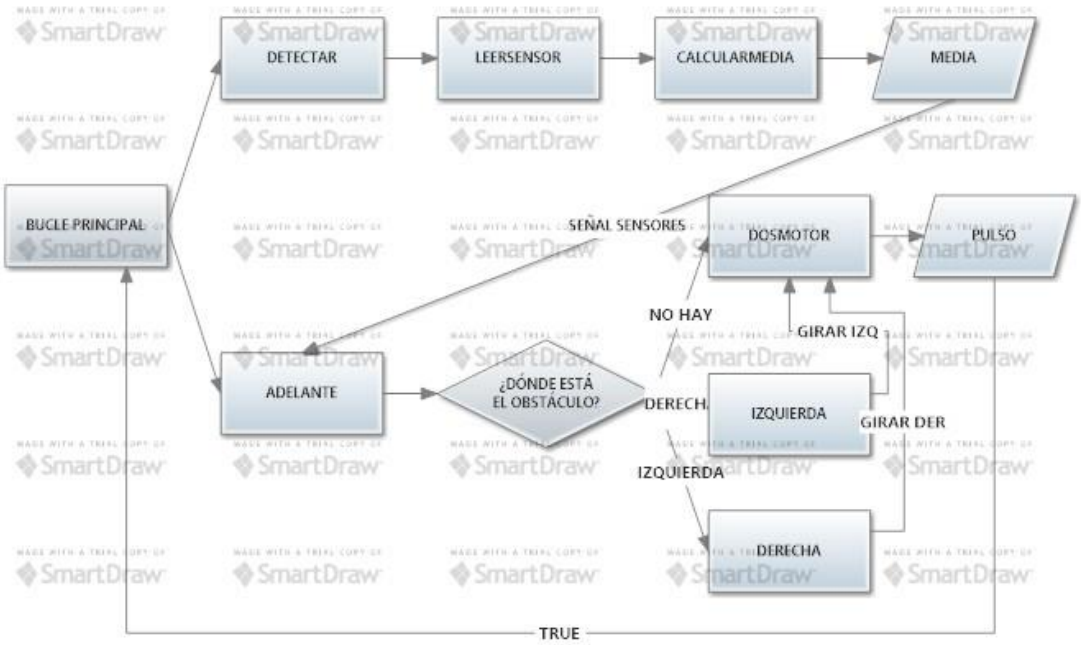


Imagen 3. Diagrama de flujo

4. Herramientas y entorno

Para la realización de este proyecto hemos utilizado diferentes herramientas. A lo largo de este apartado las detallaremos, así como el entorno de programación Arduino.

Como se ha podido comprobar a lo largo de los anteriores puntos (y en los sucesivos), la construcción del robot tiene dos ámbitos completamente diferenciados, hardware y software, por lo tanto, en este apartado se volverá a hacer distinción entre las herramientas utilizadas para la manufacturación del robot y las empleadas para su programación.

4.1 Herramientas utilizadas en la manufacturación del robot

Para construir un robot a partir de cero hemos tenido que realizar diferentes tareas de bricolaje y para cada una de ellas hemos tenido que utilizar diferentes herramientas.

4.1.1 Tornillo de banco

El tornillo de banco es una herramienta utilizada para dar una sujeción eficaz a un material y así poder realizar diferentes funciones sobre el. En nuestro caso lo hemos utilizado para sujetar la plancha de metacrilato a cortar y, una vez cortadas las piezas, para realizar las perforaciones necesarias para ensamblar los diferentes componentes hardware(ref.9).

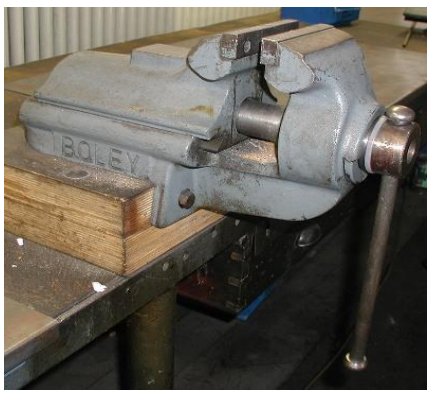


Imagen 4. Tornillo de banco.

4.1.2 Sierra de marquetería

Vamos a comenzar con la primera herramienta que utilizamos, la sierra de marquetería (ref.10).

Debido a las pequeñas dimensiones del robot, y al material utilizado, metacrilato, optamos por utilizar este tipo de sierra ya que nos otorga una mayor precisión y adaptabilidad para poder realizar los cortes más complejos de las piezas diseñadas.

La sierra de marquetería está compuesta de dos partes, el soporte, con un mango de madera, un arco de acero y dos tornillos tensores para colocar la hoja de sierra, que es la segunda parte, y es la que proporciona el corte. En este caso utilizamos una hoja de sierra especializada en cortar metacrilato.

Imagen 5. Sierra de marquetería

4.1.3 Taladro y destornillador.

El taladro es una herramienta utilizada para realizar agujeros cilíndricos en una superficie. El destornillador sirve para apretar y aflojar tornillos u otras piezas que requieren poca fuerza de apriete.

Estas herramientas las hemos utilizado para realizar los agujeros en las piezas de metacrilato del chasis del robot y en la fijación de los diferentes componentes hardware a éste.

Imagen 6. Taladro.

Imagen 7. Destornillador.

4.2 Herramientas y entorno de programación

Una vez se ha hablado de las herramientas utilizadas para la manufacturación de robot, pasamos a hablar de la herramienta utilizada para crear el diagrama de flujo y del entorno de programación utilizado para programar el autómata.

4.2.1 SmartDraw CI 21.2.4.6

SmartDraw es un procesador visual que permite la creación de todo tipo de elementos de comunicación visual: diagramas de flujo, organigramas, árboles genealógicos, mapas mentales, diagramas de bloques, circuitos eléctricos o mecánicos... entre muchos otros elementos

visuales. Para cada situación dispone de una interfaz y elementos diferentes que lo hace muy adaptable(ref.6).

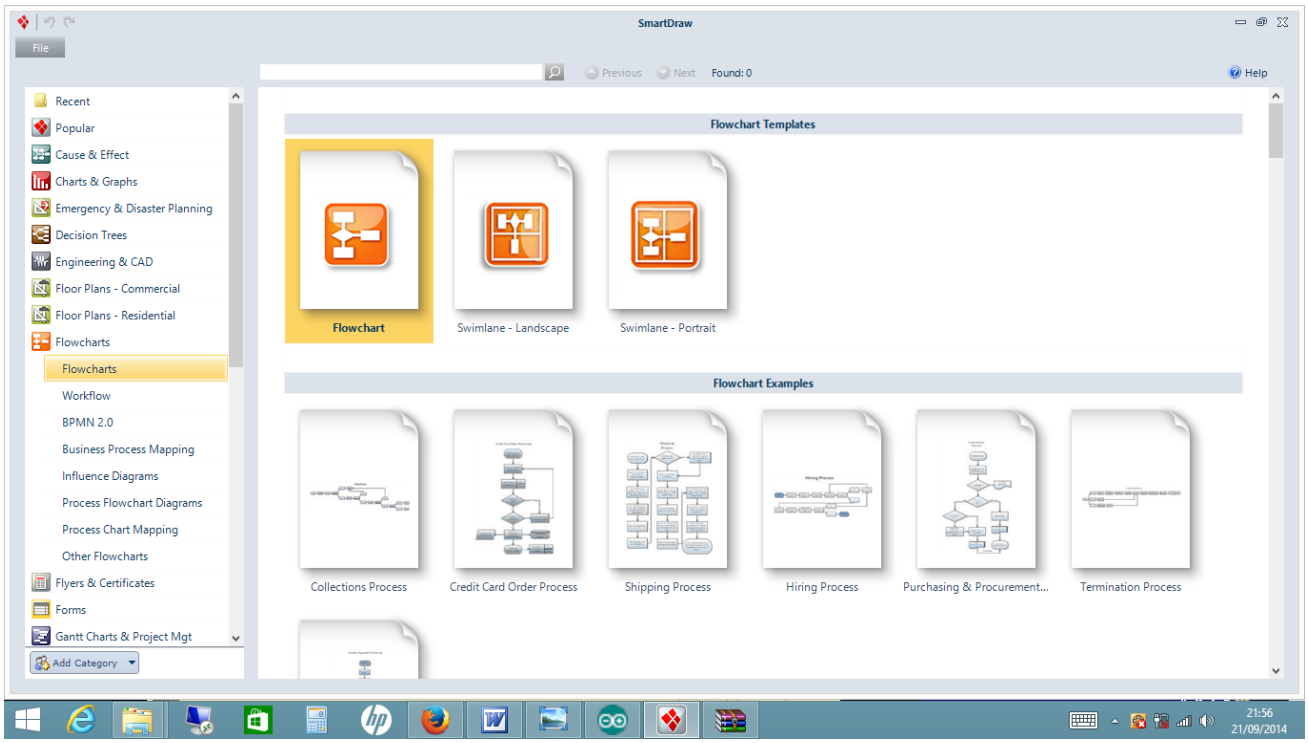


Imagen 8. Menú de inicio de SmartDraw CI 21.2.4.6.

Debido a la gran cantidad de opciones que tiene el programa, vamos a centrarnos en la opción de "Diagrama de Flujo", que es la que hemos utilizado para realizar el diagrama de flujo para el programa del robot.

Para crear diagramas de flujo, SmartDraw tiene todas las opciones necesarias y es muy visual. En el menú de la izquierda aparecen todos los tipos de elementos posibles que se pueden añadir a un diagrama de flujo. También tiene varios botones para añadir automáticamente nuevos objetos en el diagrama en la posición que deseemos. Además de esto, tiene diferentes funciones que permiten crear diferentes ramas en el diagrama de flujo, muy útil ya que simplifica mucho la tarea de crear estas ramas.

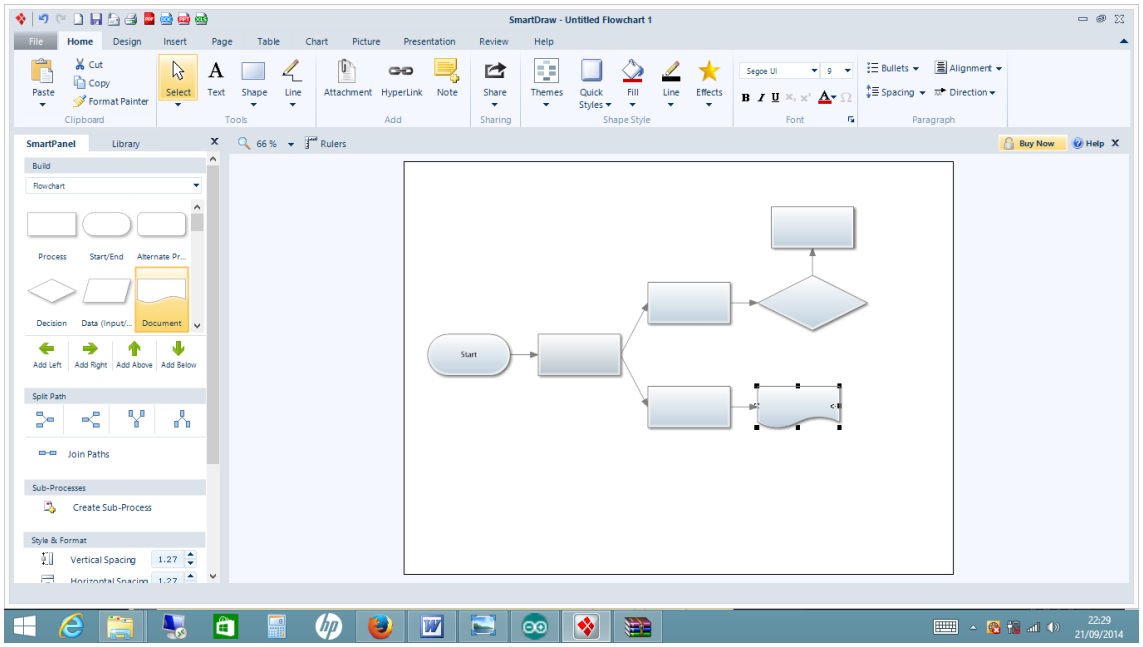


Imagen 9. Diagramas de flujo en SmartDraw CI 21.2.4.6.

4.2.2 Entorno de programación Arduino 1.0.5-r2

Tomando como referencia la página web de Arduino en general (ref.1 y ref.3) y en concreto el apartado "Environment"(ref.4), sabemos que Arduino es una plataforma de prototipado electrónico de código abierto basado en la flexibilidad y facilidad de uso de su hardware y software.

El entorno de código abierto Arduino hace fácil escribir código y subirlo a la placa Arduino o utilizarlo en Windows, Mac OS y Linux. El entorno está escrito en Java y basado en Processing, avr - gcc, y otros software de código abierto.

El entorno de desarrollo Arduino contiene un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para las funciones comunes, y una serie de menús donde hay funciones adicionales. Además, se conecta con el hardware Arduino para cargar los programas y comunicarse con ellos.

4.2.2.1 Writing Sketches

Los programas escritos utilizando Arduino se llaman "sketches". Estos "sketches" se escriben en el editor de texto y se guardan con la extensión de archivo ".ino". Tiene características para cortar/pegar y para buscar/reemplazar texto. El área de "feedback" proporciona información mientras se guarda el "sketch" o se exporta al hardware, y muestra los errores que se hayan producido. La consola muestra la salida de texto por el entorno Arduino incluyendo mensajes de error completos y demás información. La esquina derecha inferior de la ventana muestra placa Arduino conectada y el puerto serie. Los botones de la barra de herramientas le permiten compilar y cargar programas, crear, abrir y guardar "sketches", y abrir el "monitor serial" que se encarga de comprobar el estado del puerto serie actual.

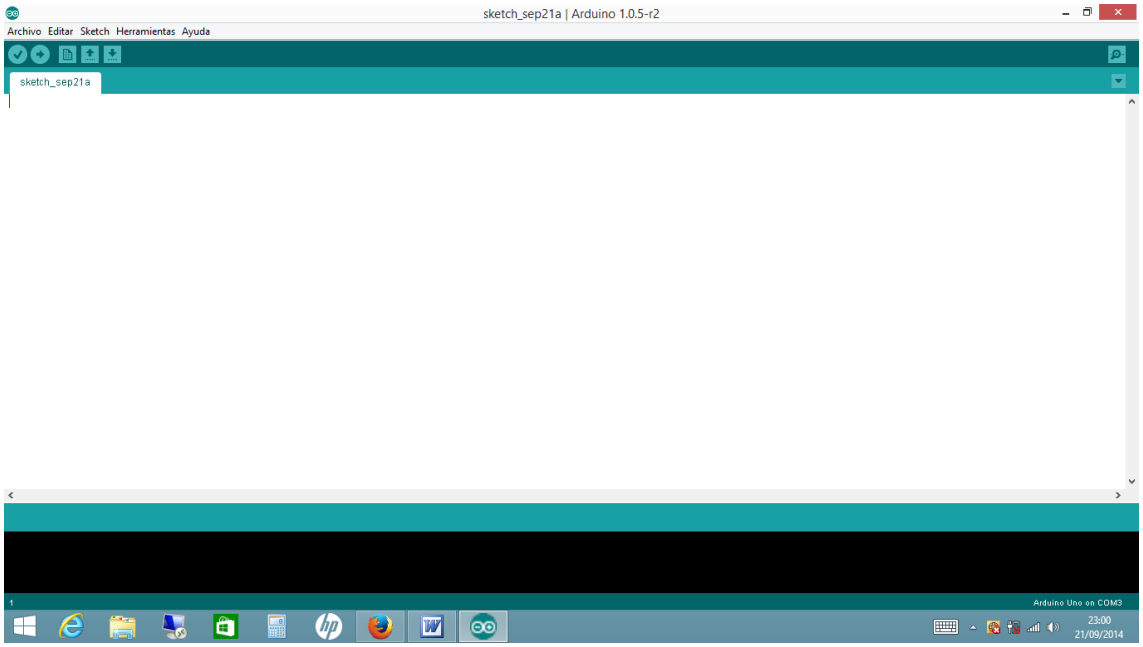


Imagen 10. Entorno de programación Arduino.

4.2.2.2 Sketchbook

El entorno de Arduino utiliza el concepto de "Sketchbook: una ubicación estándar para almacenar sus programas (o "sketches") . Los "sketches" en su "sketchbook" se pueden abrir desde el menú "Archivo> Sketchbook" o desde el botón "Abrir" de la barra de herramientas. La primera vez que se ejecuta el software de Arduino, se creará automáticamente un directorio para su "sketchbook". Puede ver o cambiar la ubicación del "sketchbook" en el cuadro de diálogo Preferencias.

A partir de la versión 1.0, los archivos se guardan con la extensión de archivo ".ino". Las versiones anteriores utilizan la extensión ".pde". Es posible abrir archivos ".pde" en las versiones 1.0 y posteriores, aunque el entorno cambiará automáticamente la extensión del archivo a ".ino".

El "sketchbook" es utilizado principalmente para tener organizado el trabajo realizado en el entorno de programación.

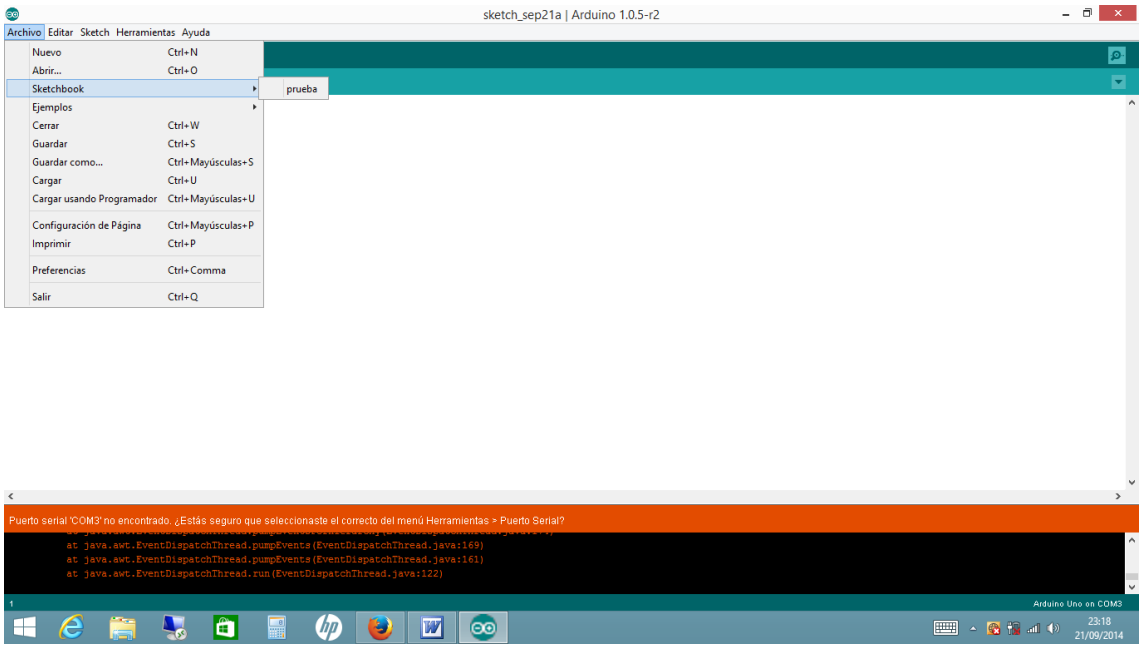


Imagen 11. Menú "Sketchbook" en Arduino.

4.2.2.3 Tabs, múltiples archivos y compilación.

Permite gestionar "sketches" con más de un archivo (cada uno de los cuales aparece en su propia pestaña). Pueden ser archivos normales Arduino (sin extensión) , archivos de C (extensión .c) , archivos de C ++ (.cpp) , o archivos de cabecera (.h).

4.2.2.4 Carga de archivos

Antes de subir un "sketch", es necesario seleccionar los elementos correctos del menú "Herramientas > Placa y Herramientas> Puerto Serie. En Windows el puerto serie es probablemente COM1 o COM2 (para una placa serie) o COM3, COM4, COM5 , COM7 , o superior (para una placa USB) - para saberlo, busca los dispositivos serie USB en la sección de puertos del Administrador de dispositivos de Windows.

Una vez que ha seleccionado la placa y el puerto serie correctos, hay que pulsar el botón de carga en la barra de herramientas o seleccionar la opción "Cargar" en el menú "Archivo". Las placas Arduino actuales se restablecerán automáticamente y comenzarán la carga. Con placas más antiguas (pre - Diecimila) que carecen de auto-reset , hay que pulsar el botón de reinicio de la placa justo antes de comenzar la carga. En la mayoría de las placas, se verá como se encienden los LED's RX y TX mientras el "sketch" se carga. El entorno de Arduino mostrará un mensaje de carga completa o error cuando finalice.

Cuando se carga un "sketch", Arduino utiliza un pequeño programa llamado "bootloader", que está cargado en el microcontrolador de la placa. Arduino permite cargar código sin necesidad de utilizar ningún hardware adicional. El "bootloader" se activa durante unos segundos mientras la placa se resetea. Una vez reseteado, el "sketch" que se acaba de cargar en el microcontrolador se inicia y comienza a funcionar.

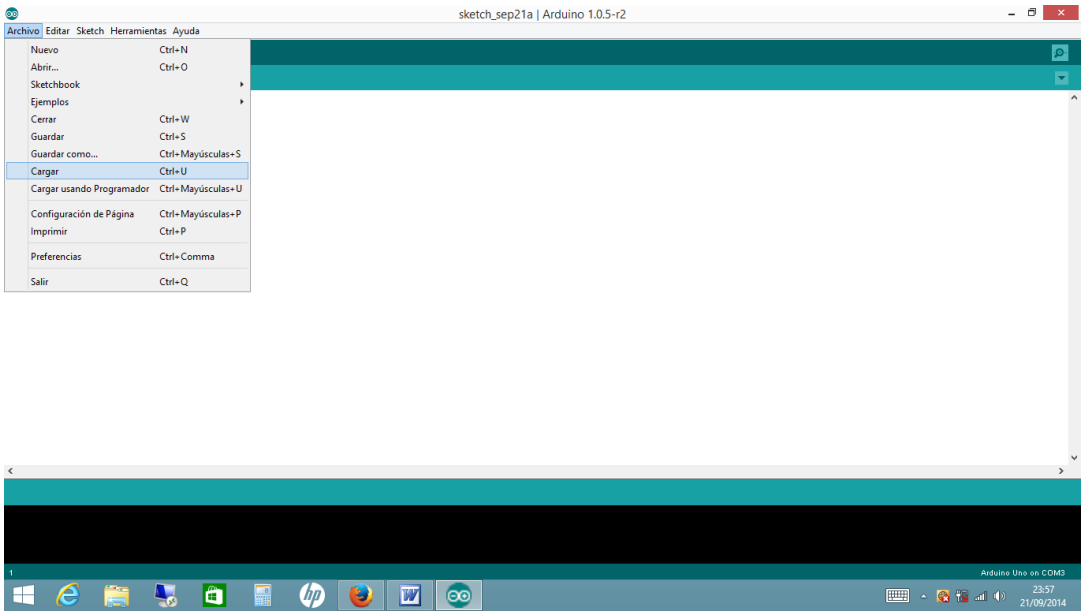


Imagen 12. Menú "Carga" en Arduino.

4.2.2.5 Librerías

Las librerías proporcionan funcionalidades adicionales para su uso en "sketches", por ejemplo, de trabajo con el hardware o la manipulación de datos . Para utilizar una biblioteca en un "sketch", hay que seleccionarlo en el menú "Sketch > Import Library". Esto insertará una o más sentencias "#include" en la parte superior del "sketch" y compilará la librería con el "sketch". Debido a que las bibliotecas se cargan en la placa con el "sketch" ,se aumenta la cantidad de espacio que el programa ocupa . Si un "sketch" ya no necesita una biblioteca, sólo hay que borrar las instrucciones "#include" en la parte superior del código.

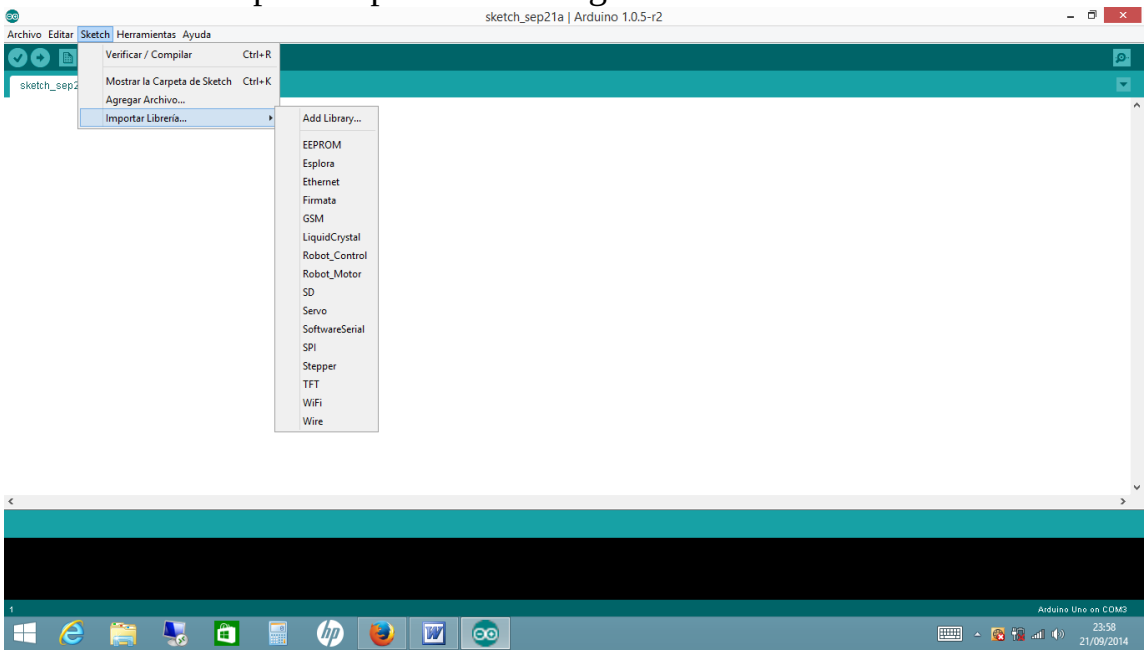


Imagen 13. Menú "Importar Librerías" en Arduino.

4.2.2.6 Lenguaje de programación Arduino

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el lenguaje de programación de alto nivel "Processing". Sin embargo, es posible utilizar otros lenguajes de programación y aplicaciones populares en Arduino, debido a que Arduino usa la transmisión serial de datos soportada por un gran número de lenguajes de programación. Para los que no soportan el formato serie de forma nativa, es posible utilizar software intermediario que traduzca los mensajes enviados por ambas partes para permitir una comunicación fluida.

El lenguaje de programación "Processing", que es en el que se basa el lenguaje de programación "Arduino", es un lenguaje de programación y

entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Para obtener una información detallada sobre la composición del lenguaje, consultar la página web de Arduino (ref.2).

5. Desarrollo

A continuación pasaremos a explicar todos los pasos realizados para la construcción del robot evita obstáculos. Para construirlo se han realizado trabajos físicos de bricolaje y ensamblaje, y se ha desarrollado un programa en Arduino para controlar el comportamiento del autómata.

5.1 Construcción y montaje del robot

En primer lugar, vamos a tratar la construcción del chasis y el montaje de los componentes hardware del robot.

La plancha de metacrilato utilizada para realizar la estructura es una plancha de 0.5X1m y 5mm de grosor.

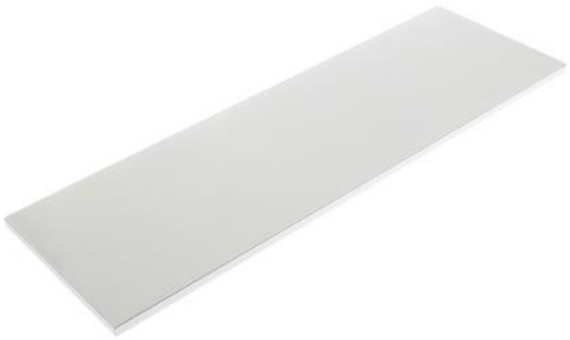


Imagen 14. Plancha de metacrilato.

En la superficie de la plancha dibujamos las piezas del chasis siguiendo el diseño mostrado en la "Imagen 1", recordando dibujar dos piezas laterales.

Una vez dibujadas las piezas en la plancha, comenzamos a realizar los cortes en ella. Fijando la plancha en el torno de banco, el primer corte que hacemos, utilizando siempre la sierra de marquetería mostrada en el punto "4.1.2", es para trabajar con una pieza más pequeña, ya que sobra mucha plancha para las piezas que vamos a cortar.

Una vez cortada la plancha separando la parte útil para le construcción de la estructura del robot de la restante, nos disponemos a cortar cada una de las 6 piezas de las que se compone el chasis del robot. Éste no es un trabajo sencillo, ya que hay que trabajar con mucha precisión para no desviarnos y realizar mal los cortes, lo que supondría tener que empezar de nuevo con el dibujo en el resto de plancha y el corte de la pieza.

Una vez cortadas todas las piezas, hay que taladrarlas para poder montar los componentes en ellas. En el punto "3.1-Diseño físico" se indicó la distribución de los componentes en las piezas del chasis, por lo tanto, siguiendo estas indicaciones, realizamos 6 agujeros en la base de la estructura (2 para la base para baterías y 4 para la placa Arduino), 4 en el frontal (2 para cada uno de los sensores de infrarrojos), y 2 en cada una de las dos piezas laterales (para fijar los servomotores en los huecos creados para ellos).

Ya tenemos todas las piezas cortadas y taladradas, ahora hemos de unirlas y ya tendremos la estructura. Debido a los problemas que hemos tenido para encontrar una escuadra de tres lados del tamaño adecuado, para unir las piezas hemos utilizado pegamento industrial. La idea inicial era unir las piezas creando la estructura mostrada en la "Imagen 2", pero al final ha habido una modificación. Inicialmente la pieza trasera iba a colocarse entre las piezas laterales y pegado a la parte trasera de la base. Con la modificación realizada, la parte trasera queda pegada a las dos piezas laterales, dejando así un hueco entre la base de la estructura y la pieza trasera. Este hueco está pensado para pasar los cables de los servomotores por él, y hacer así más estético el modelo y que los cables no salgan por fuera de la estructura.

Con la estructura creada, ya sólo falta montar los componentes en la estructura. Lo primero que vamos a montar son los componentes ubicados en la base de la estructura: la placa Arduino y la base para baterías. Proseguimos montando los dos sensores de infrarrojos en la parte exterior de la pieza frontal y la rueda omnidireccional en la parte interior. Por último, montamos un servomotor en cada pieza lateral y unimos las ruedas apretándolas en el eje de cada servomotor.

Ahora ya tenemos la estructura con todos sus componentes. Para finalizar la construcción física del robot, sólo tenemos que conectar los diferentes componentes entre ellos.

El componente central sobre el que el resto del hardware gira es la placa Arduino UNO rev3 (ref.5).

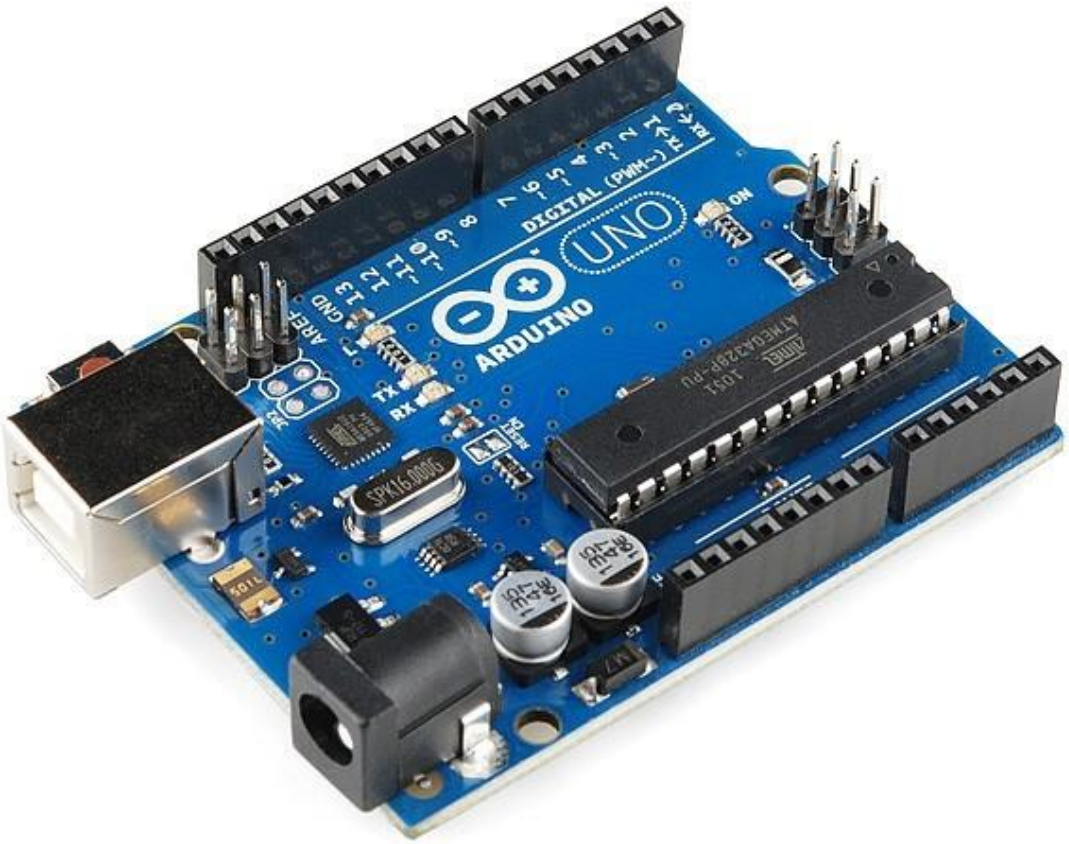


Imagen 15. Placa Arduino UNO rev3.

Esta placa contiene el cerebro del robot, el microcontrolador ATMega328 de 32 Kb donde se carga el programa, y también los pines de conexión necesarios para conectar el resto de componentes a él.

La placa Arduino Uno R3 es una placa electrónica basada en el microcontrolador Atmega328 y que tiene su módulo USB mejorado. Dispone de 14 entradas/salidas digitales y 6 de estas pueden utilizarse para salidas PWM. Además dispone de 6 entradas analógicas, un oscilador de

16MHz, una conexión USB, un conector de alimentación, un cabezal ICSP y un pulsador para el reset. Para empezar a utilizar la placa sólo es necesario conectarla al PC a través de un cable USB, o alimentarla con un adaptador de corriente AC/DC. También, para empezar, puede alimentarse sencillamente con una batería. Una de las características principales de la UNO es que no utiliza el convertidor USB-serial FTDI. Por el contrario, ofrece el microcontrolador Atmega16U2 programado como convertidor USB-serial.

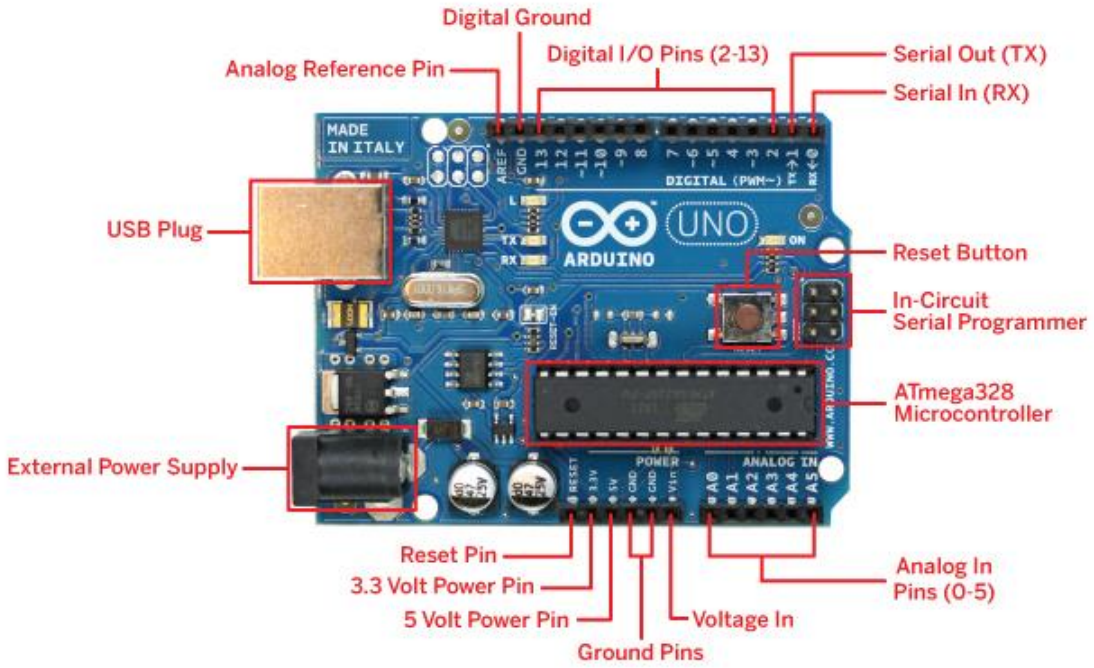


Imagen 16. Esquema Placa Arduino UNO rev3.

A la placa Arduino conectaremos la base para baterías, los dos sensores y los dos servomotores. Debido a que algunos cables de los diferentes componentes se tienen que conectar al mismo pin de la placa, hemos utilizado dos regletas de empalme de cables de 2x1.

Imagen 17. Regleta de empalme de cables.

En la primera regleta conectamos el cable de alimentación de la base para baterías y los cables de alimentación de los servomotores, para que los servomotores reciban la alimentación directamente de las baterías. En la segunda, conectaremos los cables de Gnd de los servomotores y los sensores infrarrojos. A partir de ahora omitiremos las conexiones a través de las regletas para hacer más fácil la comprensión de las conexiones directamente desde los diferentes componentes a la placa con microcontrolador Arduino.

La base para baterías tiene dos cables, uno rojo y uno negro. El cable rojo se conecta al pin de entrada Vin y el cable negro al pin Gnd.

Imagen 18. Base para baterías 4xAA.

Los servomotores utilizados son los servomotores "Pololu SM-S4303R" que tienen tres cables cada uno. Los cables rojos, de alimentación, se conectan directamente al cable de alimentación de la base para baterías, consiguiendo así un mayor amperaje y, por lo tanto, una mayor velocidad en el giro de los ejes de los servos. Los cables blancos de los servos se conectan a los pines digitales de la placa. El cable blanco del servomotor derecho se conecta al pin digital 11 y el de la izquierda al 10. Estos cables reciben el pulso que hace que los servomotores giren su eje en sentido horario o en sentido contrario a la agujas del reloj, según las ordenes recibidas por el microcontrolador.

Imagen 19. Servomotor Pololu SM-S4303R.

Products specification								Technical parameters						
Size (mm)					Weight		Wire	4.8V			6V			Rotation angle
								Speed	Torque		Speed	Torque		
A	B	C	D	E	g	oz	cm	rpm	kg·cm	oz·in	rpm	kg·cm	oz·in	
41.3	20.7	40.2	50.3	10.0	41	1.45	30.0	43	3.3	45.91	54	5.1	70.95	360°

(Specifications are subjected to change without notice.)

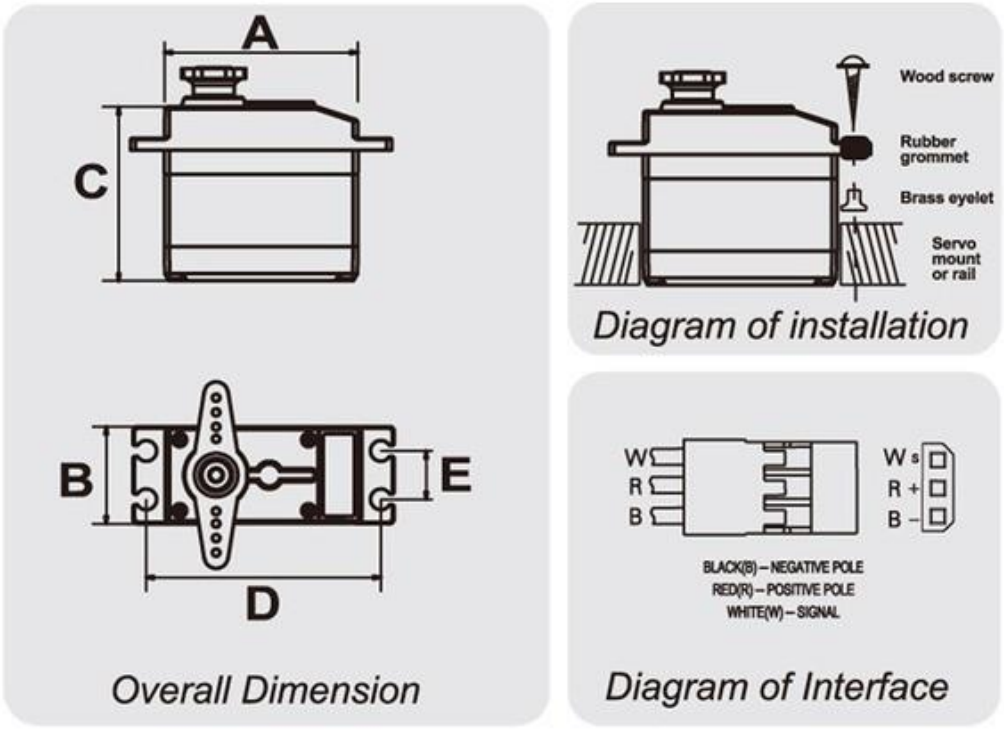


Imagen 20. Especificaciones técnicas y dimensiones Servomotor Pololu SM-S4303R(ref.7).

Una vez conectados los servos, unimos las ruedas bidireccionales a los ejes de los servos e instalamos la rueda omnidireccional en el interior de la pieza frontal utilizando una escuadra metálica para que quede en la posición deseada.

Imagen 21. Rueda.

Imagen 22. Rueda omnidireccional.

Por último, conectamos los cables de los sensores infrarrojos. Estos cables son externos a los sensores y se conectan a través de una conexión "JST".

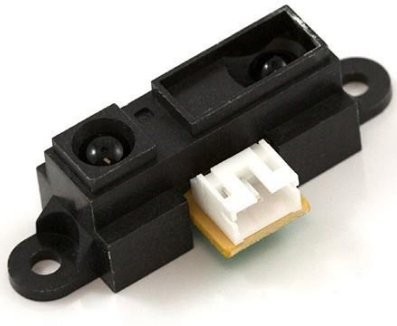
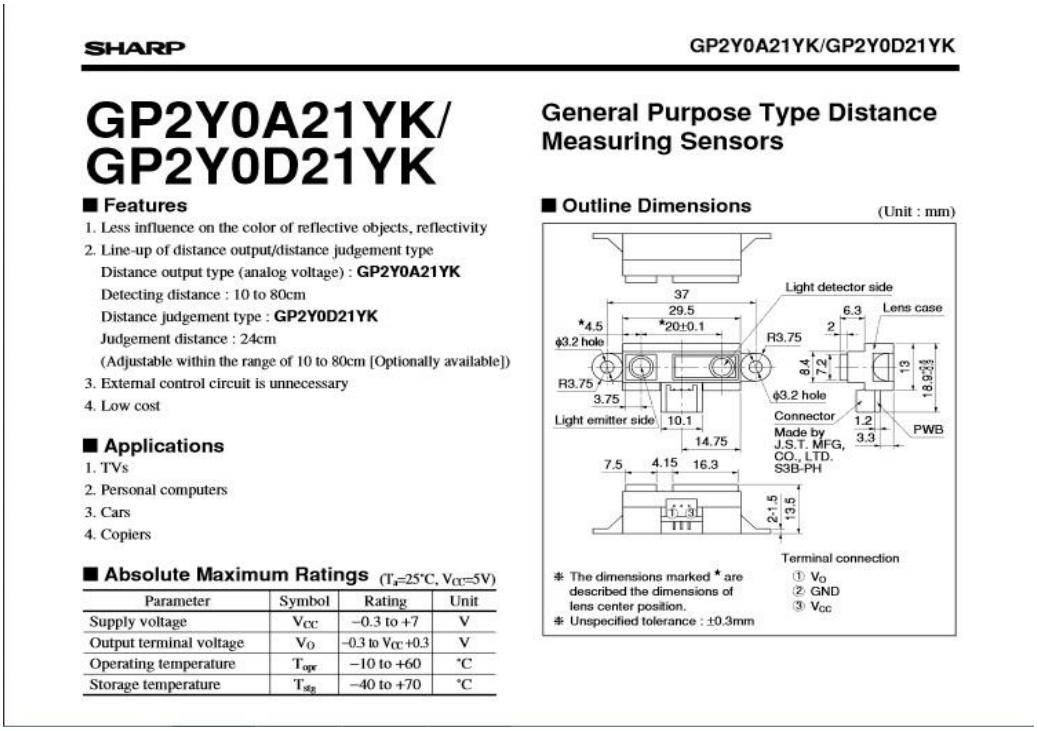


Imagen 23. Sensor IR SHARP.

Imagen 24. Cable JST

Conectamos los cables JST a las conexiones JST de los sensores. Los cables negros se conectan a "Gnd", los rojos al pin "5V" y los cables amarillos a los pines de entrada analógica. El cable amarillo del sensor derecho al pin analógico A5 y el del sensor izquierdo al A4.



SHARP GP2Y0A21YK/GP2Y0D21YK

GP2Y0A21YK/ GP2Y0D21YK

General Purpose Type Distance Measuring Sensors

■ Features

1. Less influence on the color of reflective objects, reflectivity
2. Line-up of distance output/distance judgement type
   Distance output type (analog voltage) : **GP2Y0A21YK**
   Detecting distance : 10 to 80cm
   Distance judgement type : **GP2Y0D21YK**
   Judgement distance : 24cm
   (Adjustable within the range of 10 to 80cm [Optionally available])
3. External control circuit is unnecessary
4. Low cost

■ Applications

1. TVs
2. Personal computers
3. Cars
4. Copiers

■ Absolute Maximum Ratings (T_a=25°C, V_{CC}=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	−0.3 to +7	V
Output terminal voltage	V_O	−0.3 to V_{CC} +0.3	V
Operating temperature	T_{opr}	−10 to +60	°C
Storage temperature	T_{stg}	−40 to +70	°C

■ Outline Dimensions (Unit : mm)

Imagen 25. Especificaciones técnicas físicas Sensor SHARP(ref.8).

Una vez acabado el montaje del robot, pasamos a pasar a detallar la programación del microcontrolador.

Imagen 26. Vista del robot en perspectiva.

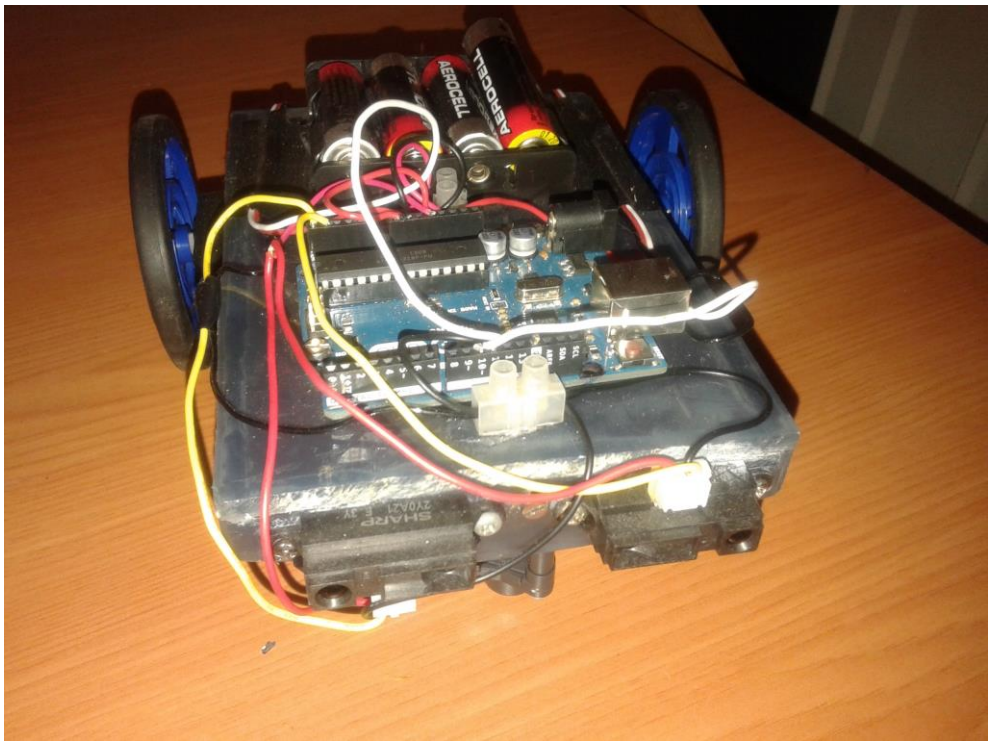


Imagen 27. Vista frontal del robot.

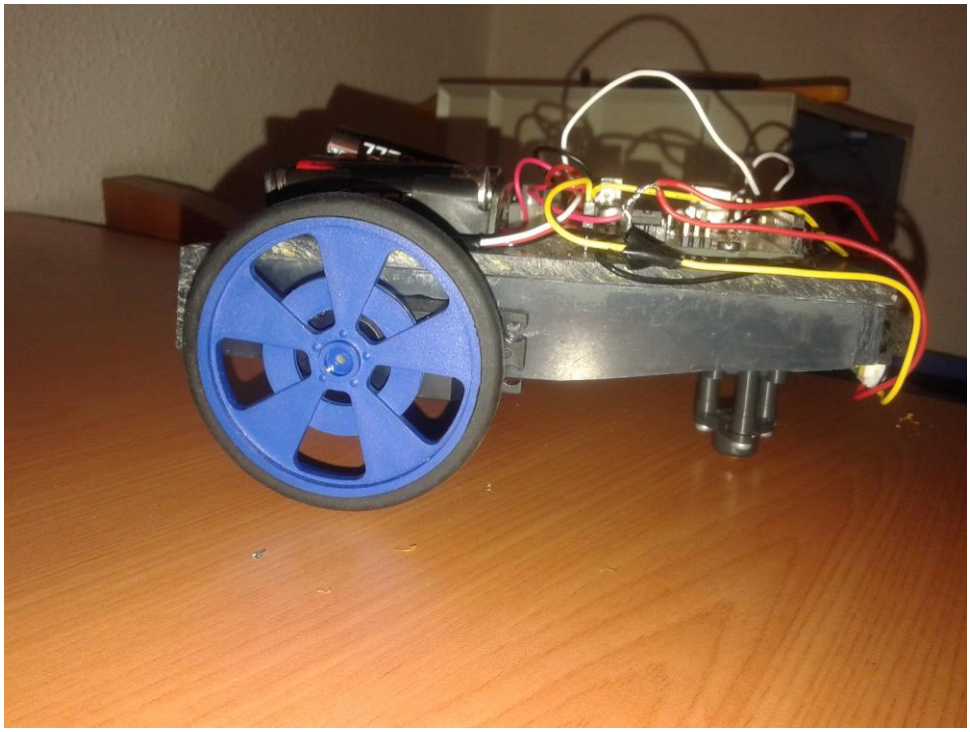


Imagen 28. Vista lateral del robot.

5.2 Programación

Una vez construida la estructura con sus componentes y conectados los cables pasamos a programar el microcontrolador. Tal y como dijimos en los puntos "2.2-Análisis de requerimientos de Software" y "3.2-Diseño del Software", el programa tiene que realizar la detección de obstáculos y dependiendo de si los encuentra o no, deberá girar en sentido contrario a donde ha encontrado el obstáculo o seguir hacia adelante.

Primero mostraremos el programa completo y después lo iremos explicando método a método.

```
int pinMotor1 = 11;
int pinMotor2 =10;
unsigned long tiempo1;
unsigned long tiempo2;
int sensorderecha=5;
int dato;
int dato2;
int sensorizquierda=4;
int contador;
```

```
int vector[5];
int minimo;
int maximo;
int media;
int suma;
int mediasensor;
boolean obstaculoIzquierda;
boolean obstaculoDerecha;


void setup()
{
     Serial.begin(115200);
     pinMode(pinMotor1, OUTPUT);
     pinMode(pinMotor2, OUTPUT);
     pinMode(sensorizquierda,INPUT);
     pinMode(sensorderecha,INPUT);
     tiempo1 =0;
     tiempo2 =0;
}


void loop()
{
     detectar();
     adelante();
}


void detectar()
{
     dato2=leersensor(sensorizquierda);
     dato=leersensor(sensorderecha);
     obstaculoIzquierda = false;
          obstaculoDerecha = false;

     if(dato2>260)
     {
          obstaculoIzquierda = true;
     }
     else if (dato>260)
     {
          obstaculoDerecha = true;
     }
```

```
      else
      {
            obstaculoIzquierda = false;
            obstaculoDerecha = false;
      }
}


int leersensor(int pin)
{
      for ( contador=0; contador<5;contador=contador+1)
      {
            vector[contador]=analogRead(pin);
      }

      mediaSensor=calcularmedia();
      return mediaSensor;
}


int calcularmedia()
{
      minimo=1023;
      maximo=0;
      suma=0;

      for(contador=0;contador<5;contador=contador+1)
      {
            minimo=min(minimo,vector[contador]);
            maximo=max(maximo,vector[contador]);
            suma = suma+vector[contador];
      }

      suma= suma -minimo-maximo;
      media= suma/3;
      return media;
}


void adelante()
{
      if(obstaculoDerecha==true)
      {
            izquierda();
```

```
        }
        else if (obstaculoIzquierda==true)
        {
                derecha();
        }
        else
        {
                dosmotor(1257,1700);
        }
}


void izquierda()
{
        while ((obstaculoDerecha==true) ||
        (obstaculoIzquierda==true))
        {
                        dosmotor(1300,1300);
                        detectar();
        }
}


void derecha()
{
        while ((obstaculoIzquierda==true) ||
        (obstaculoDerecha==true))
        {
                dosmotor(1700,1700);
                detectar();
        }
}


void dosmotor(int anchodepulsom1, int
anchodepulsom2)
{
        pulso(anchodepulsom1,pinMotor1);
        pulso(anchodepulsom2,pinMotor2);
}


boolean pulso (int anchoPulso, int servo)
{
```

```
        unsigned long tiempoantiguo;

        if(servo==pinMotor1)
        {
             tiempoantiguo=tiempo1;
        }
        else  if(servo==pinMotor2)
        {
             tiempoantiguo=tiempo2;
        }

        tiempoactual=millis();

        if( tiempoactual>tiempoantiguo+22)
        {
             if(servo==pinMotor1)
             {
                  tiempo1=millis();
             }
             else  if(servo==pinMotor2)
             {
                  tiempo2=millis();
             }

             digitalWrite(servo, HIGH);
             delayMicroseconds(anchoPulso);
             digitalWrite(servo, LOW);
             return true;
        }
        else
        {
             return false;
        }
   }
```

Código 1. Programa del robot móvil.

5.2.1 Inicialización del programa

La inicialización del programa está compuesta por la declaración de variables, el método "setup( )" y el método "loop( )". En el lenguaje de programación Arduino los métodos "setup( )" y "loop( )" son obligatorios.

```
int pinMotor1 = 11;
int pinMotor2 =10;
unsigned long tiempo1;
unsigned long tiempoactual;
unsigned long tiempo2;
int sensorderecha=5;
int dato;
int dato2;
int sensorizquierda=4;
int contador;
int vector[5];
int minimo;
int maximo;
int media;
int suma;
int mediaSensor
boolean obstaculoIzquierda;
boolean obstaculoDerecha;
```

Código 2. Declaración de variables.

En esta parte del programa declaramos las variables que vamos a utilizar en el programa.

Inicializamos las variables "pinmotor1=11;", "pinmotor2=10;", "sensorderecha=5" y "sensorizquierda=4", dándoles los valores de los pines a los que conectamos los cables de señal de los motores (pinmotor1 y pinmotor2) y de los sensores(sensorderecha y sensorizquierda) en la placa microcontroladora.

```
void setup()
{
     Serial.begin(115200);
     pinMode(pinMotor1, OUTPUT);
     pinMode(pinMotor2, OUTPUT);
```

```
        pinMode(sensorizquierda, INPUT);
        pinMode(sensorderecha, INPUT);
        tiempo1 =0;
        tiempo2 =0;
}
```

Código 3. Método "setup()".

El método setup sirve para inicializar el comportamiento de los diferentes pines de señales de entrada y salida de la placa utilizando la función "pinMode( )" en la que pasamos como parámetro el pin que queremos ajustar y si va a ser de entrada o de salida. Además, inicializamos las variables "tiempo1" y "tiempo2" a 0. Estas variables las utilizaremos en el método "pulso( )".

```
void loop()
{
        detectar();
        adelante();
}
```

Código 4. Método "loop( )".

El método loop( ) es el bucle que siempre se ejecuta en los programas arduino. Se usa para controlar de forma activa la tarjeta arduino. En nuestro caso, realiza los métodos "detectar( )" y "adelante( )" continuamente.

5.2.2 Detección de obstáculos

Una de las dos funciones principales del robot es la detección. El robot lee de los sensores infrarrojos, calcula una media aritmética de las cinco lecturas que hace cada uno de los sensores y trata ese valor para decidir si hay un obstáculo a evitar o no.

```
void detectar()
{
        dato2=leersensor(sensorizquierda);
        dato=leersensor(sensorderecha);
```

```
        obstaculoIzquierda = false;
        obstaculoDerecha = false;

        if(dato2>260)
        {
             obstaculoIzquierda = true;
        }
        else if (dato>260)
        {
             obstaculoDerecha = true;
        }
        else
        {
             obstaculoIzquierda = false;
             obstaculoDerecha = false;
        }
}
```

Código 5. Método "detectar( )".

El método "detectar( )" lee de los sensores con el método "leersensor( )", que explicaremos a continuación, y asigna el valor devuelto en las variables "dato" y "dato2". También inicializa a "false" las variables "obstaculoIzquierda" y "obstaculoDerecha". Si el valor leído desde el sensor derecho es mayor que 260, "obstaculoIzquierda" se pone a "true", y si el valor del sensor izquierda es mayor que 260, "obstaculoDerecha" también se pone a "true". Si ninguno de los valores es mayor que 260 las dos variables se vuelven a poner a "false".

```
int leersensor(int pin)
{
     for ( contador=0; contador<5;contador=contador+1)
     {
          vector[contador]=analogRead(pin);
     }

     mediaSensor=calcularmedia();
     return mediaSensor;
}
```

Código 6. Método "leersensor(int pin)".

PARAMETRO	SIMBOLO	EVALUACION	UNID
Voltaje de alimentacion	V_{CC}	-0.3 to +7	V
Terminal de salida de tension	V_O	-0.3 to (V_{CC} +0.3)	V
Temperatura de operacion	Topr	-10 to +60	°C
Temperatura de almacenamiento	Tstg	-40 to +70	°C

PARÁMETRO	SÍMBOLO	CONDICIONES	MIN.	TYP.	MAX.	UNIDAD
Range de la medicion de distancia	ΔL		10	-	80	cm
Terminal de salida de tensión	V_O	L = 80 cm	0.25	0.4	0.55	V
Diferencia de voltaje de salida	ΔV_O	Salida de cambio en ΔL (80 cm - 10 cm)	1.65	1.9	2.15	V
Promedio actual de la fuente	I_{CC}	L = 80 cm	-	30	40	mA

GRAFICO DE VOLTAJE DE SALIDA (VS) DISTANCIA REFLECTADA

Imagen 29. Voltaje de salida vs. distancia reflectada(ref.8).

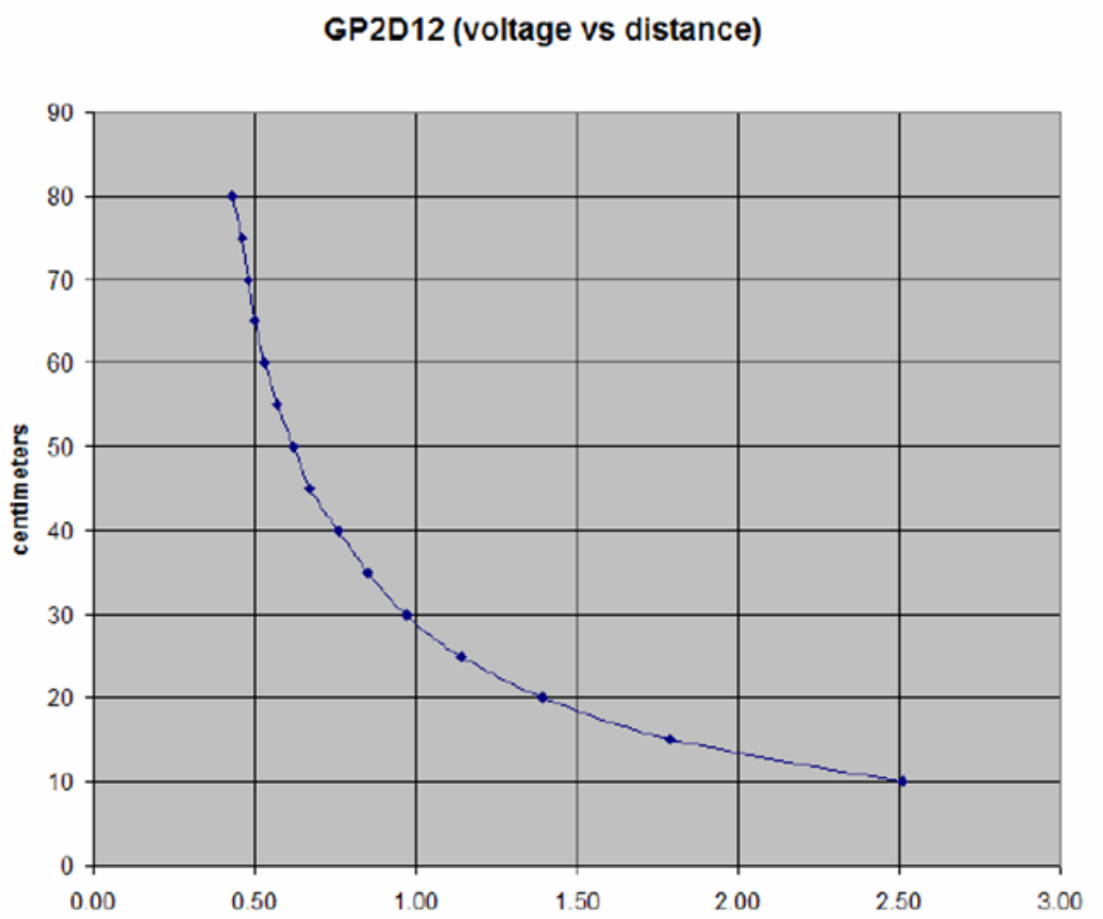


Imagen 30. Gráfico voltaje vs. distancia(ref.8).

El método "leersensor(int pin)" realiza cinco lecturas analógicas de la señal de uno de los sensores cuyo cable de señal está conectado al pin que se pasa como parámetro. Estas cinco lecturas se realizan para evitar una lectura errónea del sensor. Una vez tenemos las cinco lecturas del sensor se calcula la media de las señales y se devuelve la variable "mediasensor", que, como vimos, se utiliza en el método "detectar( )".

```
int calcularmedia()
{
      minimo=1023;
      maximo=0;
      suma=0;

      for(contador=0;contador<5; contador=contador+1)
      {
            minimo=min(minimo,vector[contador]);
            maximo=max(maximo,vector[contador]);
            suma = suma+vector[contador];
      }

      suma= suma -minimo-maximo;
      media= suma/3;
      return media;
}
```

Código 7. Método "calcularmedia( )".

El método "calcularmedia( )" realiza una media aritmética de las cinco lecturas analógicas realizadas por el método "leersensor( )". Inicializa las variables "minimo=1023", "maximo=0" y "suma=0". Una vez inicializadas las variables, realiza un "for" calculando en cada iteración si el valor actual del vector "contador" es menor que el valor de la variable "minimo" utilizando la función "min", si el valor actual del vector "contador" es mayor que el valor de la variable "maximo" utilizado la función "max", y asignando en la variable "suma" la suma de los cinco valores del vector "contador. Habiendo determinado los valores de las variables "minimo", "maximo" y "suma" en el "for", pasamos a eliminar los dos primeros de estos valores en la variable "suma" y después sacamos la media aritmética de los tres valores restantes. Este resultado se asigna a la variable "media" que es devuelta por el método.

5.2.3 Movimiento hacia adelante y evasión de obstáculos

La otra función principal del robot es la del movimiento. Teniendo las variables "obstaculoIzquierda" y "obstaculoDerecha", el robot gestiona si el robot tiene que girar a izquierda o derecha o debe moverse hacia adelante.

```
void adelante()
{
     if(obstaculoDerecha==true)
     {
          izquierda();
     }


     else if (obstaculoIzquierda==true)
     {
          derecha();
     }
     else
     {
          dosmotor(1300,1700);
     }
}
```

Código 8. Método "adelante( )".

El método "adelante( )" comprueba si las variables "obstaculoDerecha" u "obstaculoIzquierda" están a "true. Si la primera es "true", llama al método "izquierda( )", para gestionar el giro a la izquierda del robot, si lo está la segunda, llama a "derecha( )", que gestiona el giro a la derecha. Si las dos variables están a "false", el robot sigue hacia adelante, llamando al método " dosmotor(int anchodepulsom1, int anchodepulsom2)", cuyos valores son los necesarios para que los ejes de los motores giren, uno, en sentido horario y el otro en sentido contrario a las agujas del reloj (veremos las especificaciones técnicas del funcionamiento del motor cuando hablemos del método "pulso"). Para corregir la desviación del robot hacia la derecha, cuando tenía que moverse

únicamente hacia adelante, modificamos el valor del parámetro del pulso utilizado para mover el motor derecho(esto se detallará en el apartado de pruebas).

```
void izquierda()
{
     while ((obstaculoDerecha==true) ||
     (obstaculoIzquierda==true))
     {
               dosmotor(1300,1300);
               detectar();
     }
}


void derecha()
{
     while ((obstaculoIzquierda==true) ||
     (obstaculoDerecha==true))
     {
          dosmotor(1700,1700);
          detectar();
     }
}
```

Código 9. Métodos "izquierda( )" y "derecha( )".

Los métodos izquierda y derecha, una vez llamados por el método adelante, entran en un bucle que se ejecuta mientras cualquiera de los dos sensores está detectando un obstáculo (si una de las dos variables "obstaculoIzquierda" u "obstaculoDerecha" están a "true"). Este bucle ejecuta el método "dosmotor" con los parámetros necesarios para girar el motor para girarlo a izquierda o derecha, según el método que lo esté llamando, y después llama al método "detectar( )" para saber si las dos variables siguen siendo "true" o cambian a "false", terminando la ejecución del bucle y del método.

```
void dosmotor(int anchodepulsom1,int anchodepulsom2)
{
     pulso(anchodepulsom1,pinMotor1);
     pulso(anchodepulsom2,pinMotor2);
}
```

Código 10. Método dosmotor(int anchodepulsom1,int anchodepulsom2)

El método "dosmotor(int anchodepulsom1, int anchodepulsom2)" llama al método "pulso(anchodepulsom*,pinMotor*)" dos veces, una por cada servomotor, pasando los parámetros que recibe a los métodos "pulso".

```
boolean pulso (int anchoPulso, int servo)
{
     unsigned long tiempoantiguo;

     if(servo==pinMotor1)
     {
          tiempoantiguo=tiempo1;
     }
     else  if(servo==pinMotor2)
     {
          tiempoantiguo=tiempo2;
     }

     tiempoactual=millis();

     if( tiempoactual>tiempoantiguo+22)
     {
          if(servo==pinMotor1)
          {
               tiempo1=millis();
          }
          else  if(servo==pinMotor2)
          {
               tiempo2=millis();
          }

          digitalWrite(servo, HIGH);
          delayMicroseconds(anchoPulso);
          digitalWrite(servo, LOW);
          return true;
```

```
        }
        else
        {
              return false;
        }
}
```

Código 11. Método "pulso (int anchoPulso, int servo)".

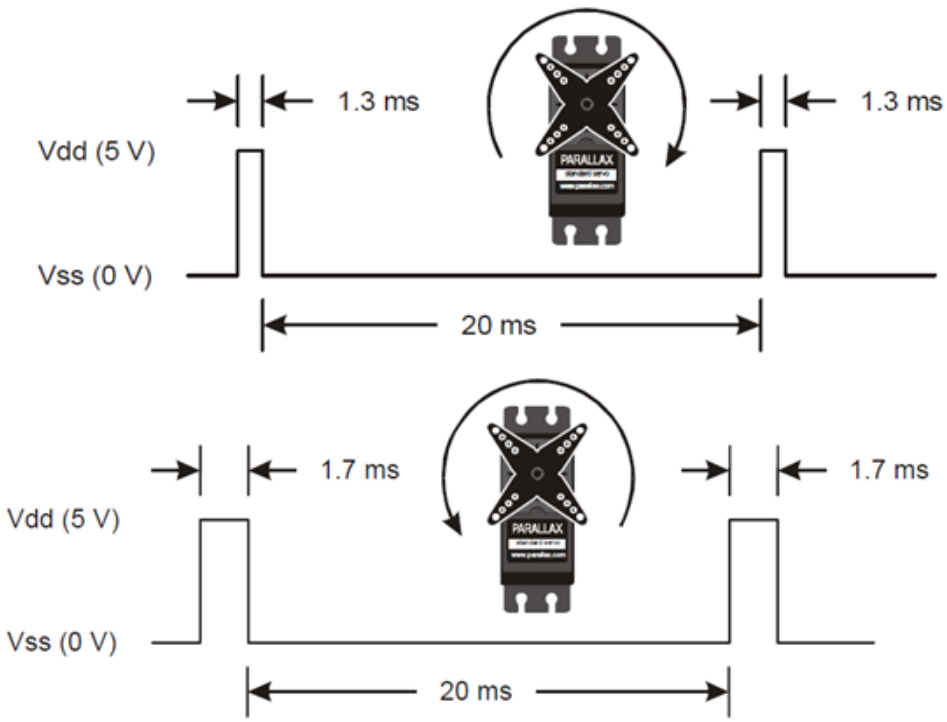


Imagen 31. Sentido de rotación de los servos con respecto al ancho de pulso.

El método "pulso" recibe el valor del ancho de pulso que vas a introducir en el servomotor y el pin de salida al que está conectado el cable de señal de del servomotor como parámetros. Utilizando el parámetro "servo", comprobamos cuál de los servos es sobre el que vamos a actuar. Una vez comprobado, asignamos el valor de la variable "tiempo1" o "tiempo2" (dependiendo del servo sobre el que estamos trabajando) a la variable "tiempoantiguo". Después de esto, a la variable "tiempoactual" se le asigna el valor, en milisegundos, del tiempo de ejecución del programa con la función "millis( )". Teniendo las variables "tiempoantiguo" y "tiempoactual" con valores asignados, comprobamos si "tiempoactual" es 22 milisegundos mayor que "tiempoantiguo". Si lo es, volvemos a

comprobar con qué motor estamos trabajando y asignamos el valor del tiempo de ejecución (función "millis( )") a la variable "tiempo1" o "tiempo2", dependiendo del servo con el que trabajamos en ese momento. Para poder mandar la señal al servomotor utilizamos las funciones "digitalWrite(servo, HIGH)" y "digitalWrite(servo, LOW)". Entre estas dos sentencias hay que escribir el ancho de pulso que mandamos al servomotor sobre el que estamos actuando con la función "delayMicroseconds(anchoPulso)".

6. Pruebas y correcciones

Ahora ya tenemos finalizado el robot y nos queda comprobar su funcionamiento. Introducimos las pilas AA en la base para baterías y el robot comienza a funcionar. En este apartado explicaremos las pruebas y modificaciones que hemos realizado para arreglar los pequeños fallos que hemos localizado en el comportamiento del robot.

6.1 Comprobación del funcionamiento de los sensores IR

Para comprobar el correcto funcionamiento de los sensores de infrarrojos SHARP, realizamos un pequeño programa para sacar por pantalla el valor que lee cada sensor.

Lo que realiza este programa es la lectura analógica de los datos recibidos por los sensores, mostrando por pantalla el dato analógico leído y su equivalente en centímetros de cada uno de los sensores SHARP.

```
int ir_sensor0 = 4;
int ir_sensor1 = 5;

void setup()
{
     Serial.begin(9600);
}

void loop()
{
     int lectura, cm;
     lectura = analogRead(ir_sensor0);
     Serial.print("Izquierda:");
     Serial.println(lectura);
     cm = pow(3027.4 / lectura, 1.2134);
     Serial.print("Sensor Izquierda: ");
     Serial.println(cm);
     delay(2000);
```

```
        lectura = analogRead(ir_sensor1);
        Serial.print("Derecha:");
        Serial.println(lectura);
        cm = pow(3027.4 / lectura, 1.2134);
        Serial.print("Sensor Derecha: ");
        Serial.println(cm); // lectura del sensor 1
        delay(4000);
}
```

Código 12. Código del programa "prueba".

Primero inicializamos las variables que identifican a cada uno de los sensores con el valor del pin en los que están conectados los cables de señal de cada sensor. Como todos los programas en Arduino, éste tiene un método "setup( )", donde inicializamos la comunicación serie, y un método "loop( )", que en este caso contiene todo el código del programa.

En el método "loop( )" procedemos a realizar dos veces las mismas operaciones, una para el sensor izquierdo y otra para el sensor derecho. Declaramos las variables que vamos a utilizar, realizamos una lectura analógica del sensor, mostramos por pantalla el valor del dato analógico, convertimos este valor a centímetros y lo mostramos por pantalla.

Entre el código de prueba de cada uno de los sensores y al final del bucle, utilizamos las funciones "delay(2000)" y "delay(4000)" para que no saque los datos demasiado rápido por pantalla y se espere 2 segundos y 4 segundos respectivamente desde el inicio del bucle.

Al sacar por pantalla los datos de los dos sensores, podemos comprobar si las mediciones realizadas son correctas o incorrectas.

6.2 Desviación a la derecha

Al dejar el robot en una zona libre de obstáculos, éste comienza su movimiento hacia delante. Al observar este movimiento, nos damos cuenta de que el robot se va desviando poco a poco hacia su derecha. Enviando el mismo ancho de pulso a los dos servomotores, el robot debería moverse en línea recta sin desviarse hacia ningún lado, por lo que deducimos que el problema se encuentra en la estructura del robot, que al haberse realizado a mano, no tiene una simetría perfecta y por lo tanto, las ruedas bidireccionales no están en posición paralela. Para corregir este error, modificamos el ancho de pulso del servomotor derecho para que gire más rápido. A través del método "prueba y error" conseguimos hallar el valor del ancho de pulso correcto que hemos de introducir en el servo derecho para que el robot se mueva en línea recta hacia delante.

```
void adelante()
{
     if(obstaculoDerecha==true)
     {
          izquierda();
     }
     else if (obstaculoIzquierda==true)
     {
          derecha();
     }
     else
     {
          dosmotor(1257,1700);
     }
}
```

Código 8. Método "adelante( )".

Como podemos observar en la llamada a "dosmotor(1275,1700)", hemos modificado el valor del primer parámetro a 1275 en vez de 1300 que es el que nos indican las especificaciones técnicas del servomotor. Reducir

este valor hace que el servomotor gire más rápido y por lo tanto ajusta la desviación del robot hacia la derecha.

6.3 Choque de las ruedas

Al poner en marcha el robot y dejarlo funcionar, nos hemos dado cuenta de que en ocasiones, cuando gira y deja de detectar obstáculos, comienza a moverse hacia adelante y las ruedas se enganchan con el obstáculo que acaba de esquivar. Esto sucede por que las ruedas sobresalen de la estructura, por lo tanto, según el sensor, ya no hay obstáculo, pero las ruedas siguen chocando con el obstáculo.

Para solucionar este problema, hemos intentando aumentar el tiempo que el robot gira para evitar un obstáculo y realizar dos veces seguidas el envío de señal a los servomotores, entre otras ideas, sin conseguir resultados satisfactorios, ya que, con la primera solución, puede hacer que el robot choque con otro obstáculo, y con la segunda, toma el ancho de pulso introducido por las dos llamadas como si se introdujera en una llamada solo, se pongan las llamadas una detrás de la otra, o escribiendo primero una poniendo la escritura en el sensor a "HIGH" y cerrándola con "LOW" y repitiendo el proceso para separar las escrituras en el sensor.

Después de realizar diferentes pruebas de programación, hemos llegado a la conclusión que para que el robot no sufra estos contratiempos sería necesario instalar unos sensores infrarrojos, como los frontales, en los laterales del robot, consiguiendo así una lectura de la distancia a la que se encuentran los obstáculos a los lados del robot que podríamos manejar para que el robot se alejara de esos obstáculos cuando estuvieran muy próximos.

7. Conclusiones

Al finalizar este trabajo se han obtenido importantes conclusiones, una de ellas y tal vez la más relevante es que para la realización de un robot no se necesita grandes inversiones económicas ni tecnología punta, pues contando con recursos limitados se ha logrado desarrollar un robot autónomo con microcontrolador.

Al estudiarlos y trabajar con microcontroladores, se ha interiorizado la gran importancia y enorme versatilidad de la que disponen, ya que las aplicaciones relativamente complejas, como las que envuelve al presente proyecto, son llevadas a cabo sin mayor problema pues con un sólo microcontrolador se controla la plataforma. Eso demuestra que se puede hacer el control automático de casi cualquier proceso o conjunto de procesos.

En este proyecto en particular hemos conectado al microcontrolador dos servomotores, dos sensores de infrarrojos y una base para baterías, permitiéndonos controlarlos y hacer que el robot funcione tal y como estaba previsto. La programación de la interacción de unos con otros ha resultado bastante sencilla.

La evasión de objetos ha resultado satisfactoria, permitiéndonos empezar a pensar en usos más específicos para este tipo de robot, tal y como explicaremos en el punto "7.1 Mejoras y usos futuros".

Se ha logrado el objetivo general de este proyecto, ya que la creación manual de la estructura y la programación para controlar el robot móvil han sido satisfactorias. La estructura es equilibrada y estable y la programación cumple con todas las funciones que se requiere realice el robot.

7.1 Mejoras y usos futuros

Debido a que la función de este robot es la evasión de obstáculos, se puede pensar en casi cualquier mejora y/o uso. Hay muchísimos ámbitos en los que puede ser útil incorporar la evasión de obstáculos en robots móviles. En este apartado vamos a mostrar algunos ejemplos de ello.

Se puede mejorar el robot poniendo sensores infrarrojos laterales de alta precisión para tener un mayor control del entorno a su alrededor.

Además, con un sensor inferior apuntando hacia el suelo por delante de la rueda frontal, podríamos evitar que el robot se cayera por algún desnivel.

Incorporando un GPS, y programando una ruta preestablecida, se podría conseguir que el robot actuase antes cualquier obstáculo que encontrara en el camino y volviera a la ruta que estaba siguiendo. Una vez conseguido esto se podrían añadir más componentes y programarlos para conseguir un robot de seguridad, de limpieza, de reconocimiento, etcétera.

Añadiendo componentes podemos construir casi cualquier tipo de robot móvil

Las características de nuestro robot le permiten acoplarse a otros sistemas automatizados tales como robots de exploración, en los cuales si no pueden evitar obstáculos podrían quedarse bloqueados y no seguir sin sus funciones, en los coches, para conseguir evitar accidentes y salvar vidas, y muchos más.

8.Referencias

1.http://www.arduino.cc/en/pmwiki.php?n=

2.http://arduino.cc/en/Reference/HomePage

3.http:/arduino.cc/en/Guide/Introduction

4.http:/arduino.cc/en/Guide/Environment

5.http:/arduino.cc/en/Main/ArduinoBoardUno

6.http:/www.smartdraw.com/product/

7.http:/www.pololu.com/product/1248

8.https://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf9.

9.http://es.wikipedia.org/wiki/Tornillo_de_banco

10.http://es.wikipedia.org/wiki/Segueta

8.1 Índice de imágenes

8.2 Índice de extractos de código