UNLaM

# Sistemas Operativos Avanzados
# Proyecto CarryBot

## Informe Final


### Docentes:

- Graciela de Luca
- Waldo A. Valiente
- Sebastián Barillaro
- Mariano Volker
- Carnuccio, Esteban Andrés
- Gerardo Garcia

### Integrantes:

- Kaucic Florencia
- Bistolfi Facundo Raul
- Palopoli Juan José
- Martinez Sebastian

# Índice


Motivación del proyecto ..... 2
Estructura y funcionamiento general del proyecto ..... 2
- Estructura ..... 2
- Funcionamiento ..... 3

Proyecto: Sistema Embebido ..... 4
- Concepto ..... 4
- Sensores ..... 4
- Actuadores ..... 5

Componentes utilizados ..... 6
Circuito ..... 9
Descripción de componentes ..... 10
- Modulo Bluetooth HC-05 ..... 10
- Placa de presión ..... 12
- LED ..... 13
- Parlante ..... 15
- Sensor Ultrasónico ..... 16

Modelos 3D y su rol ..... 19
Problemas SE y sus soluciones ..... 23
- Bluetooth ..... 23
- Alimentación ..... 26

Android ..... 27
- Aplicación ..... 27
- Diagrama de comunicación general ..... 28

Mejoras para el producto ..... 29
- Tecnología Bluetooth para rastrear el destino ..... 29
- Cantidad de productos por pedido ..... 29
- Forma y soporte de la bandeja. ..... 29

Anexo I: Arduino Mega 2560 ..... 30
Anexo II: Protocolo de comunicación ..... 33

# Motivación del proyecto

¿Se ha encontrado con recurrentes quejas de sus clientes? ¿Pedidos que no han sido tomados, elevadas demoras en la entrega de los mismos, o aun peor, entregar algo que no coincide con lo que el cliente ha pedido? No sufra más, ¡CarryBot llega al rescate!
**Un Robot que se encarga de entregarle al cliente sus respectivos pedidos.**

CarryBot destaca por 5 características clave:

- **Pedidos desde aplicación:** mediante una aplicación mobile el cliente puede realizar sus pedidos ¡sin tiempo de espera!
- **Entrega autónoma de pedidos:** CarryBot se encarga de llevar su pedido sin necesidad de intervención de terceros.
- **Siempre al tanto:** Se envían notificaciones al Cliente sobre el estado de su pedido para librarlo de la horrible sensación de no saber qué pasa.
- **Cancelación instantánea:** Si el Cliente desea cancelar su pedido, puede hacerlo directamente desde la aplicación y esto tendrá un efecto instantáneo.
- **Identificación de Restaurant:** Gracias a la tecnología GPS, se reconoce la sucursal en la que el Cliente se encuentra.

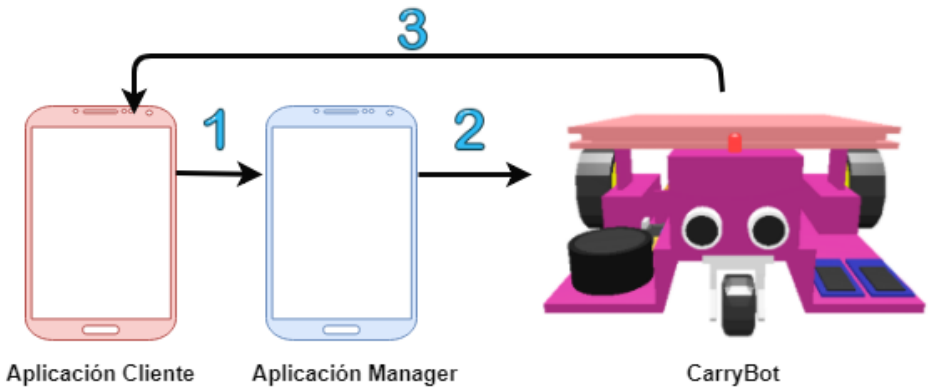
# Estructura y funcionamiento general del proyecto

## Estructura

Para cumplir con su objetivo, Proyecto CarryBot presenta 3 elementos principales:

- **Aplicación Cliente:** La cual debe ser instalada en un dispositivo móvil que disponga de Bluetooth para que el **Dispositivo CarryBot** pueda localizarlo, y acceso a conexión WiFi para poder enviar su pedido.
- **Aplicación Manager:** La cual debe ser instalada en un dispositivo móvil que disponga de Bluetooth, para poder transmitir los datos necesarios al **Dispositivo CarryBot** para llevar los pedidos al Cliente, y acceso a conexión WiFi para poder recibir los pedidos.
- **Dispositivo CarryBot**: El cual viene equipado con un módulo Bluetooth el cual utiliza para comunicarse con la **Aplicación Manager** y otro módulo Bluetooth para poder localizar al Cliente a quien le entregará el pedido.

# Funcionamiento

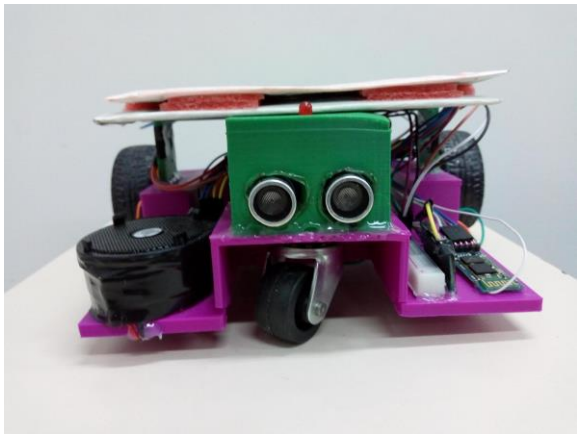
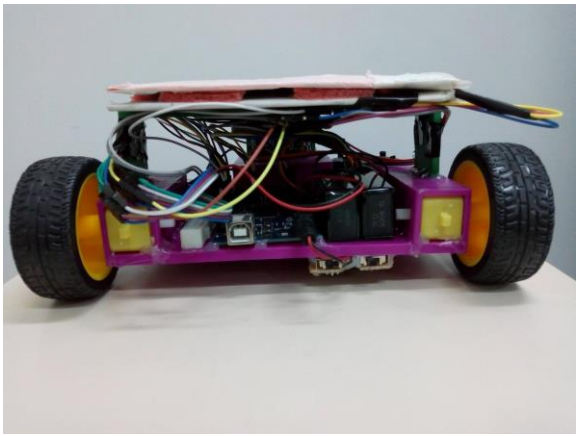
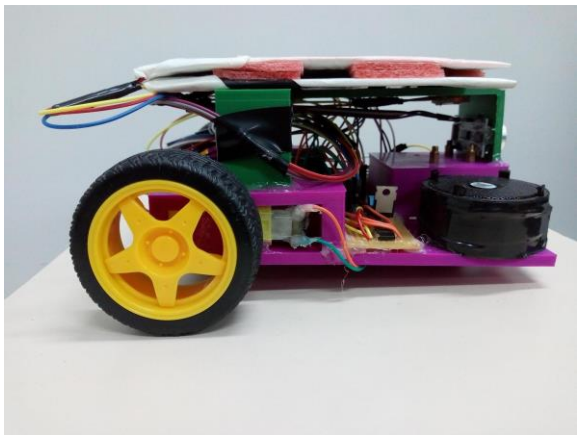
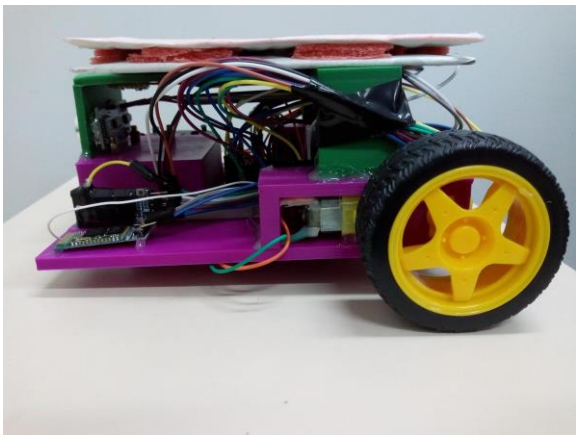



1. El Cliente, desde la Aplicación Cliente, una vez haya sido identificado el Restaurant en el que está ubicado realiza su pedido utilizando la aplicación. Dichos pedidos son enviados a la Aplicación Manager.
2. Los pedidos de los clientes que lleguen a la Aplicación Manager serán enviados a CarryBot acorde al orden en que fueron llegando.
3. El Dispositivo CarryBot, una vez recibe un pedido, espera a que se coloque el plato en su bandeja. Una vez colocado el mismo, procede a localizar y llevar el pedido al Cliente.

# Proyecto: Sistema Embebido

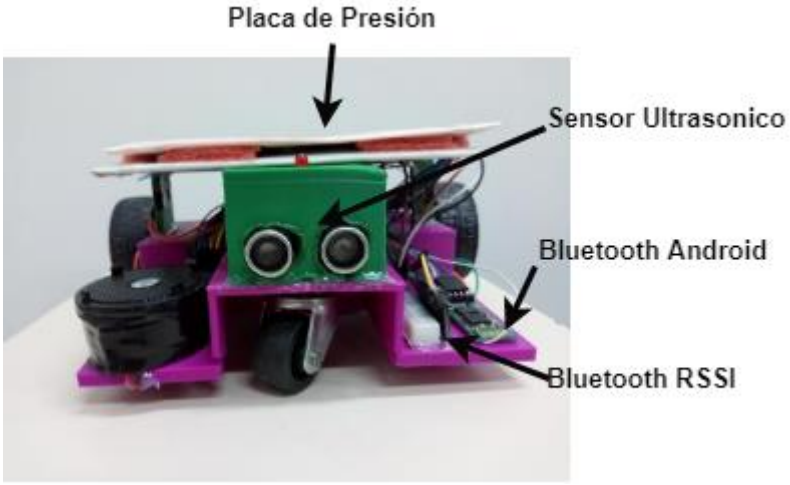
## Concepto

CarryBot surge de la necesidad de transportar objetos pequeños en espacios reducidos, como puede ser una habitación, un living, una zona de trabajo, etc.
CarryBot detecta cuando se coloca un objeto a transportar sobre él y lo transporta hacia el destino, siendo su destino un dispositivo capaz de emitir una señal Bluetooth, la cual será medida por Carrybot para lograr acercarse a su objetivo.

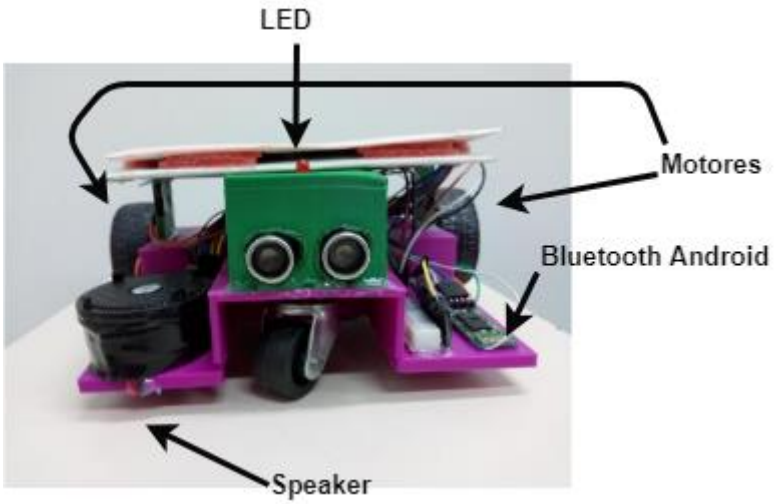
## Sensores

Los sensores de Carrybot detectan cuando tiene un objeto sobre su bandeja para empezar el recorrido. Tiene a la vez un sensor bluetooth a través del cual detecta la señal de los dispositivos cercanos. Esta señal será utilizada para seguir a la persona correspondiente. Y por último, tiene un sensor de ultrasonido, con el cual detectará los obstáculos que se encuentre en su camino para evitar chocarse con ellos. Adicionalmente, posee otro módulo Bluetooth el cual utiliza para comunicarse con el dispositivo móvil del Manager (quien estuviera encargado de gestionar los pedidos).




## Actuadores

Carrybot cuenta con dos motores que actúan en base a lo medido por los sensores y que permitirán que alcance su objetivo. También tiene actuadores (un parlante y un LED) que permitirán informar cuando llegó a su destino, o si tiene algún inconveniente que impida cumplir su objetivo.

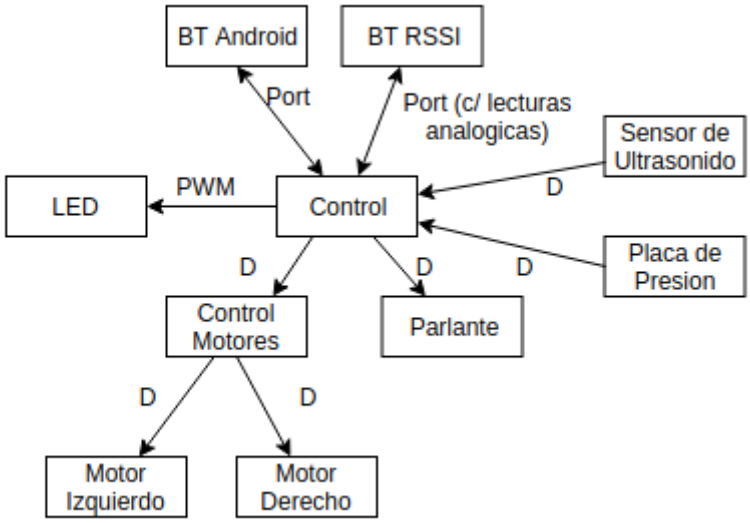


# Componentes utilizados

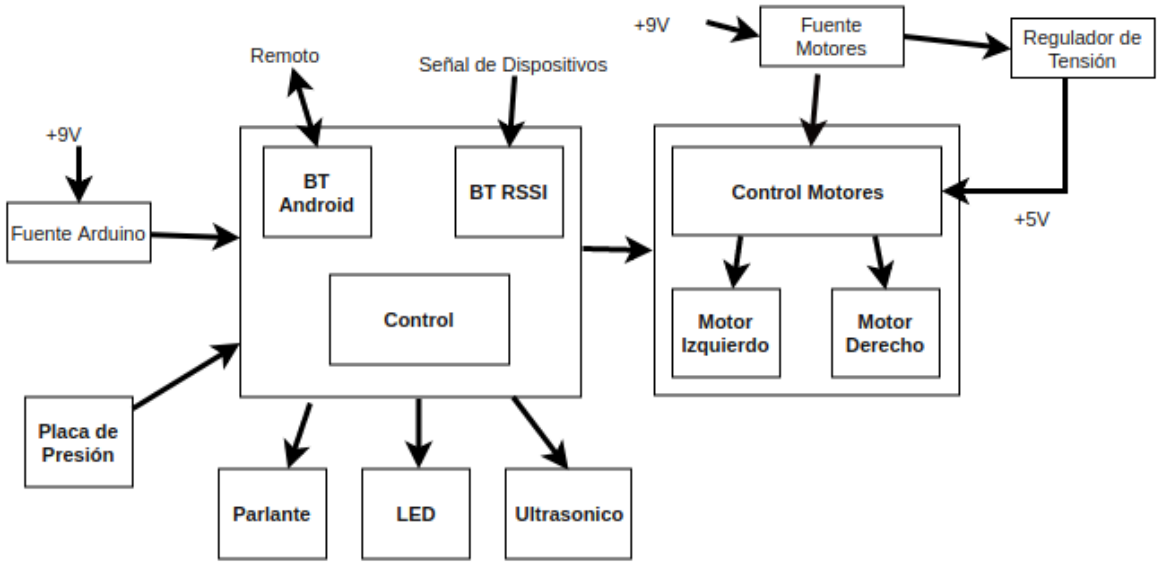
- 1 x Arduino Mega 2560
- 2 x Módulo Bluetooth HC-05
- 1 x Módulo de Sensor Ultrasónico HC-SR04
- 1 x CI Puente-H L293D
- 1 x Transistor LM7805 (regulador de voltaje con salida de 5v)
- 2 x Motor DC 6v
- 2 x Rueda para motor
- 1 x Rueda loca
- 1 x Placa de presión por peso (hecha por nosotros)
    - Papel aluminio para las superficies
    - 1 x Resistencia 10k ohms
    - Cables
    - Plancha de polietileno para la estructura o similar
- 1 x LED rojo
- 1 x Resistencia 220 ohms
- 2 x Pila 9v
- 1 x Conector para pila 9v
- 1 x Parlante
- Chasis (impresión 3D de plástico)
- Cables

# Diagramas en Bloque

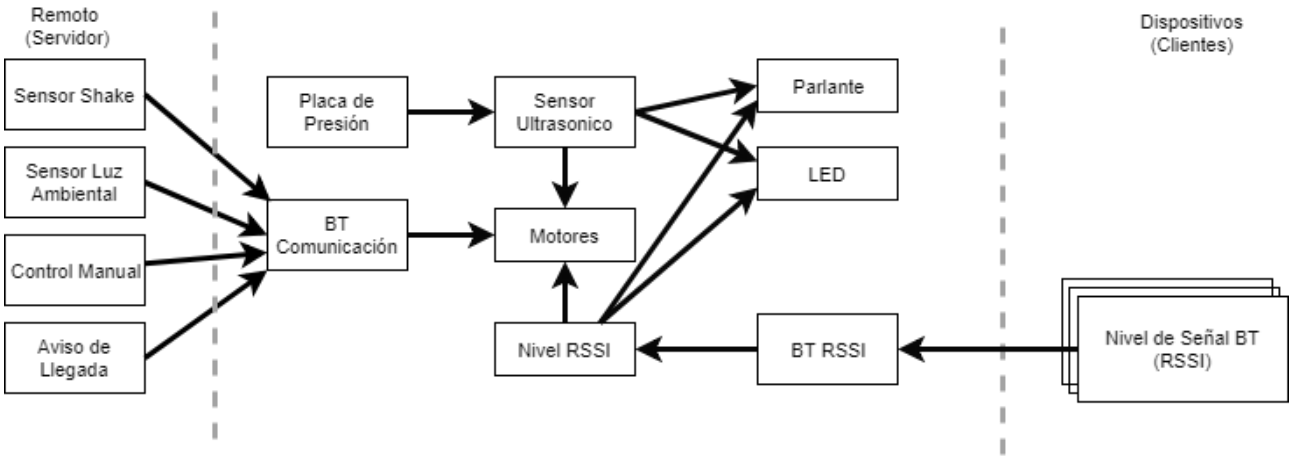
## Funcional



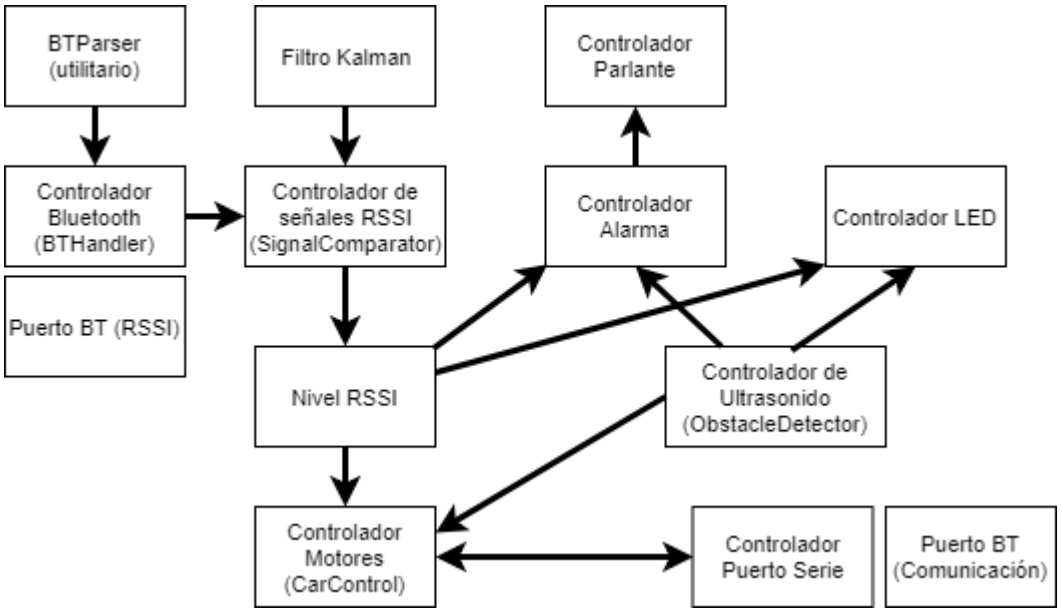

## Físico



## Lógico

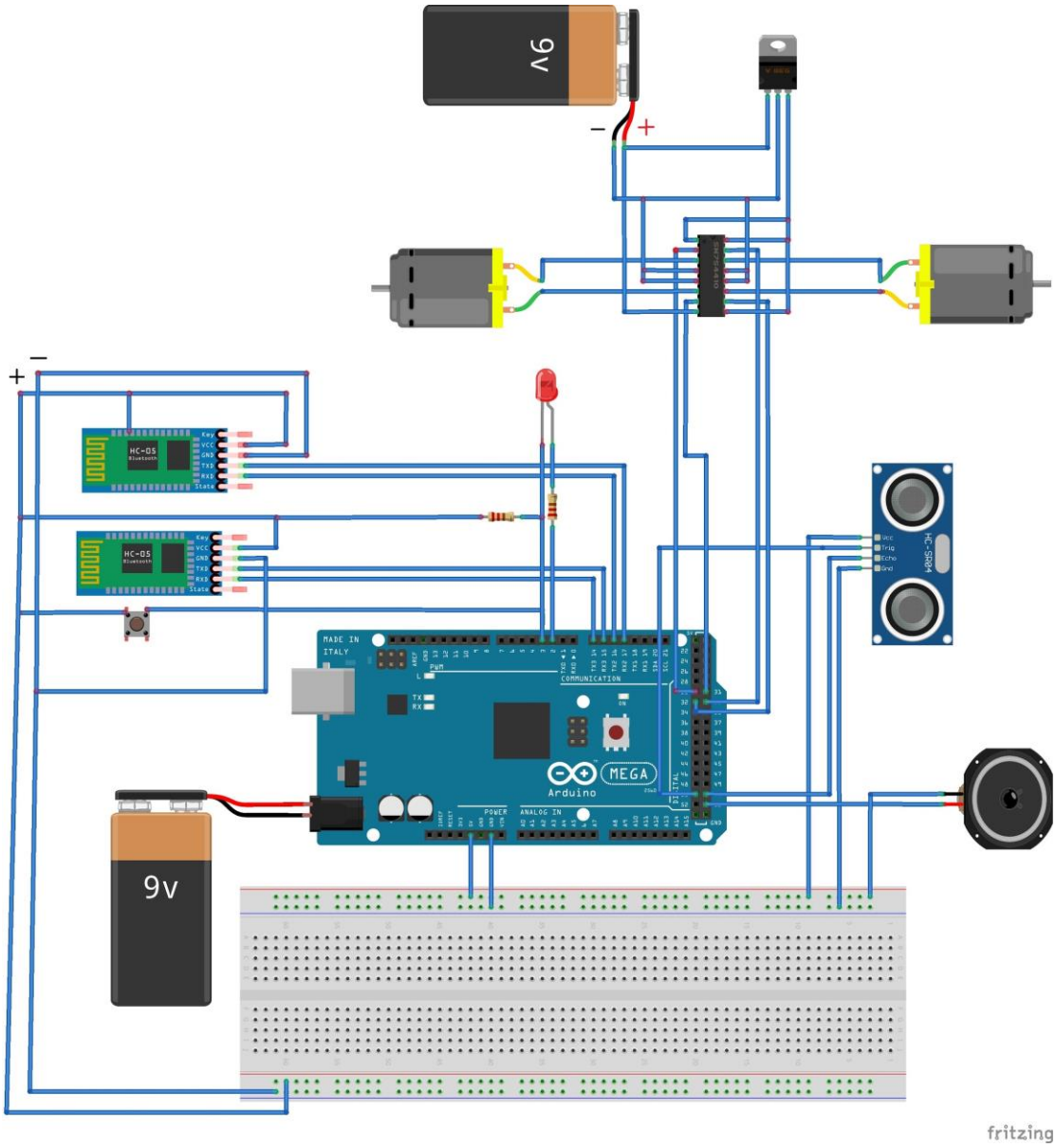



## Software



# Circuito

El siguiente circuito fue elaborado utilizando Fritzing (http://fritzing.org/home/)



# Descripción de componentes

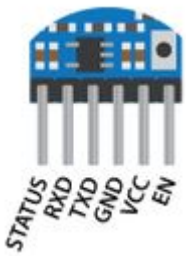
## Modulo Bluetooth HC-05

Tanto el módulo Bluetooth utilizado para captar las señales bluetooth del ambiente como el utilizado para la comunicación con Android son módulos HC-05.
El módulo HC-05 puede configurarse tanto como Master que como Slave. En nuestro caso elegimos el HC-05 porque posee los comandos necesarios para realizar la lectura de la intensidad de las señales.

### Especificaciones técnicas

- Módulo Bluetooth para Arduino y otros microcontroladores.
- Tensión de operación: de +4v a +6v (Comúnmente +5v)
- Corriente nominal: 30mA
  Rango: <100m
- Trabaja con comunicación en serie (USART) y es compatible con TTL
- Sigue el protocolo IEEE 802.15.1 (Estándar Bluetooth)
- Utiliza FHSS (Espectro expandido por salto de frecuencia)
- Puede operar en modo maestro (Master), esclavo (Slave) y en un modo híbrido (Master/Slave)
- Permite una interfaz sencilla con dispositivos Bluetooth
- Soporta las siguientes velocidades de transmisión (en baudios): 9600,19200,38400,57600,115200,230400 y 460800.

### Conexiones




El módulo HC-05 posee dos pares importantes de pines: los de alimentación y los de transmisión.

Los pines de transmisión (**RXD** y **TXD**) se conectan a los pares de pines de un puerto serial del arduino (O en el caso de no tener un puerto serie libre, al par de pines seleccionados utilizando la librería “SoftwareSerial”). Se debe conectar el pin de transmisión del arduino con el de recepción del HC-05 y viceversa. En la teoría no se debe conectar el pin de recepción del HC-05 directo al de transmisión de Arduino ya que los pines de Arduino transmiten con un máximo de 5V y los pines de transmisión del módulo HC-05 solo permiten hasta 3.3V, pero en la práctica no hemos tenido problema haciendo una conexión directa.


Los pines de alimentación (**VCC** y **GND**) se conectan directamente a la alimentación del arduino. El pin VCC del módulo HC-05 requiere 5V para funcionar correctamente.

El último par de pines (**STATUS** y **EN**) no los hemos utilizado ya que no logramos entender el correcto funcionamiento de los mismos o simplemente hemos decidido prescindir de los mismos por no considerarlos necesarios para nuestro proyecto. En la teoría el pin **STATUS** es utilizado para indicar si el módulo está funcionando correctamente, no lo utilizamos ya que las lecturas que obteniamos de este pin no coincidían con las que esperábamos y nos pareció inútil invertir más tiempo del proyecto en el entendimiento del pin **STATUS**, por lo que prescindimos de el. El pin **EN** (Enable/Key) fue otro gran problema: En la teoría este pin es utilizado para poner el módulo HC-05 en modo AT simplemente enviando un pulso alto en el momento que se enciende el módulo, pero nuevamente nos topamos con algo que no funcionaba como indicaba la teoría. La solución fue omitir el uso de este pin, y para utilizar el modo AT nos decidimos por la alternativa de puentear el pin 34 del módulo HC-05 y conectarlo a VCC. Conectando el pin 34 al VCC el módulo inicia siempre en modo AT.

### Modo AT

El modo AT es el estado del módulo HC-05 en el cual se pueden enviar comandos AT. Estos comandos se utilizan para configurar el módulo y para enviarle órdenes. Las diferencias visibles entre el modo normal y el AT son las siguientes: En modo AT el led del módulo parpadea mucho más lento que en modo normal, y la velocidad de transmisión en modo AT es de 38400 baudios mientras que la del modo normal es de 9600 baudios (por defecto, se puede modificar con los comandos AT)
La lista de comandos AT utilizados es la siguiente. Todos los comandos deben terminar con los caracteres “\r\n” para que el módulo HC-05 los tome correctamente.

| **Comando AT** | **Respuesta(s) esperada(s)** | **Descripción** |
|---|---|---|
| AT | OK | Este comando se utiliza para saber si el HC-05 está en modo AT (Devuelve OK) o no (no devuelve nada) |
| AT+RESET | OK | Se utiliza para resetear el módulo HC-05 |
| AT+ADDR | +ADDR:<dirección MAC><br>OK | Se utiliza para saber la dirección MAC del módulo |
| AT+NAME? | +NAME:<nombre><br>OK | Se utiliza para saber el nombre del módulo |
| AT+NAME=<nombre> | OK | Se utiliza para cambiar el nombre del módulo |
| AT+PSWD? | +PSWD:<password><br>OK | Se utiliza para saber la contraseña del módulo (Por defecto es “1234”) |
| AT+PSWD=<password> | OK | Se utiliza para cambiar la contraseña del módulo |
| AT+ ROLE? | +ROLE:<rol><br>OK | Se utiliza para saber en qué modo está el módulo. Si devuelve 0 está en modo esclavo, 1 en maestro y 2 en híbrido. Por defecto devuelve 0. |
| AT+ROLE=<rol> | OK | Se utiliza para configurar el modo del módulo |
| AT+ INQM? | +INQM=<p1>,<p2>,<p3><br>OK | Se utiliza para conocer los 3 parámetros del modo INQ (Que es el modo en el cual se buscan los dispositivos cercanos)<br>p1: indica si se debe realizar el descubrimiento de |

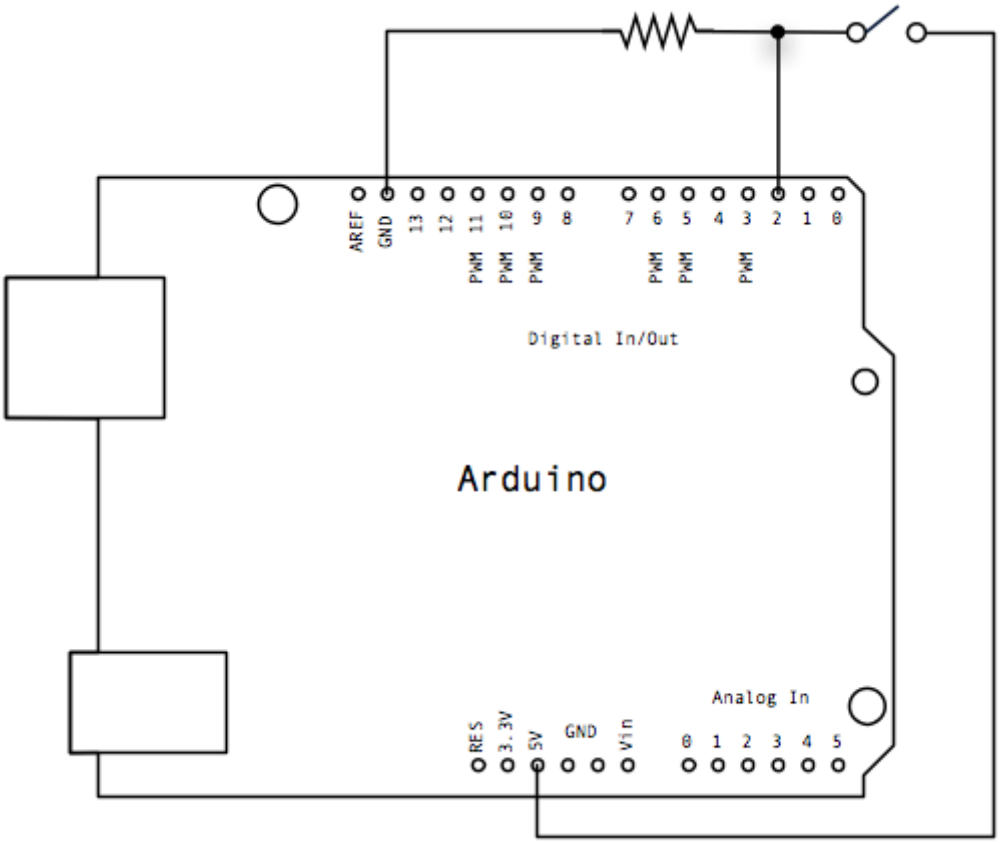
| | | |
|---|---|---|
| | | dispositivos con modo rssi (1) o no (0). Siempre utilizamos este parámetro en 1 porque nos interesa conocer el RSSI (indicador de fuerza de la señal recibida)<br>p2: Indica el máximo de dispositivos que se van a buscar durante el descubrimiento.<br>p3: Indica el timeout del modo INQ. Su valor real será p3*1.28s. En caso de no encontrar lecturas acordes a los parámetros de Clases, el modo INQ se verá interrumpido por el valor real |
| AT+INQM=<p1>,<p2>,<p3> | OK | Se utiliza para configurar los tres parámetros del proceso de descubrimiento de dispositivos (Modo INQ) |
| AT+STATE? | +STATE:<estado><br>OK | Se utiliza para conocer el estado actual del módulo. Los estados pueden ser los siguientes:<br>● "INITIALIZED" > Módulo inicializado<br>● "READY" > Módulo listo<br>● "PAIRABLE" > Esperando emparejamiento<br>● "PAIRED" > Módulo emparejado<br>● "INQUIRING" > Descubriendo dispositivos<br>● "CONNECTING" > Conectando a un dispositivo<br>● "CONNECTED" > Conectado a un dispositivo<br>● "DISCONNECTED" > Desconectado<br>En nuestro caso de aplicación, los estados que nos interesan son el de "módulo listo" para saber que podemos empezar a configurar el módulo y buscar dispositivos; y el de "descubriendo dispositivos" ante una espera larga para saber si es que el módulo sigue buscando dispositivos o está tildado. |
| AT+INIT | OK | Se utiliza para iniciar el antes mencionado "Modo INQ". No se puede realizar la búsqueda de dispositivos sin primero enviar este comando. El comando puede también devolver "ERROR(17)", que nos indica que ya habíamos enviado este comando anteriormente y ya estamos en modo INQ |
| AT+INQ | +INQ=<p1>,<p2>,<p3><br>...<br>OK | Se utiliza este comando para realizar la búsqueda de dispositivos. Devuelve una línea "+INQ..." por cada dispositivo encontrado y un OK al final para indicar que terminó la búsqueda. Los parámetros de cada dispositivos encontrado son los siguientes:<br>p1: La dirección MAC del dispositivo<br>p2: El tipo de dispositivo<br>p3: La intensidad de la señal recibida<br>Este comando puede también devolver "ERROR(16)", que nos indica que no hemos utilizado el comando AT+INIT aun. |
| AT+INQC | OK | Se utiliza este comando para cancelar la búsqueda de dispositivos |

# Placa de presión

La placa de presión que utilizamos fue armada por nosotros mismos, debido a que necesitábamos que fuera fácilmente accionable, de manera que al apoyar un objeto encima lo detecte.

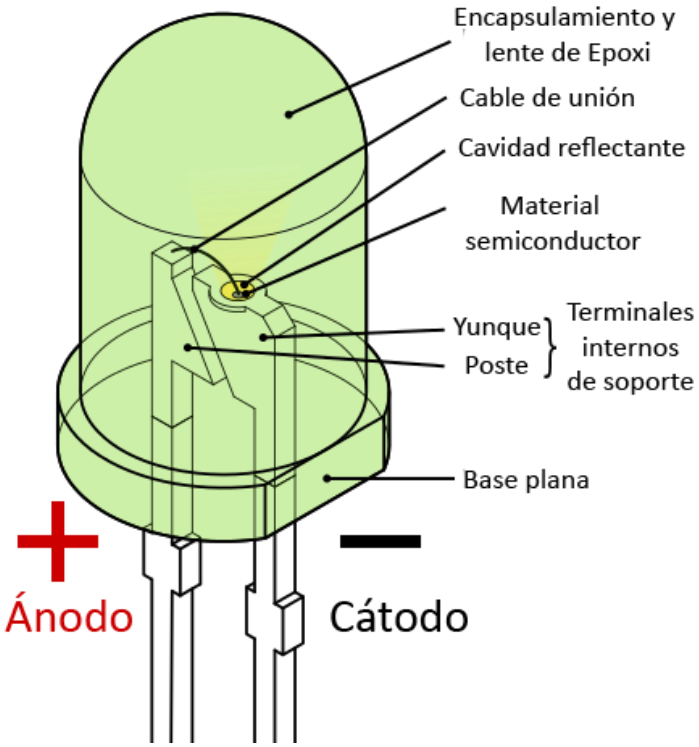
La misma está compuesta por una base sólida y una placa blanda, levemente separadas,

cubiertas de aluminio en su interior. La placa blanda, en nuestro caso hecha con una plancha de polietileno, permite que objetos de masa pequeña la desplacen lo suficiente para que haga contacto con la base sólida. Para separar una placa de la otra, utilizamos la misma plancha de polietileno en los lados, dejando el centro libre.
También tiene una resistencia de 10k Ω para que haga el pull-down necesario para su correcta lectura. Esta lectura es digital, y el circuito queda conectado de la siguiente forma:




## LED

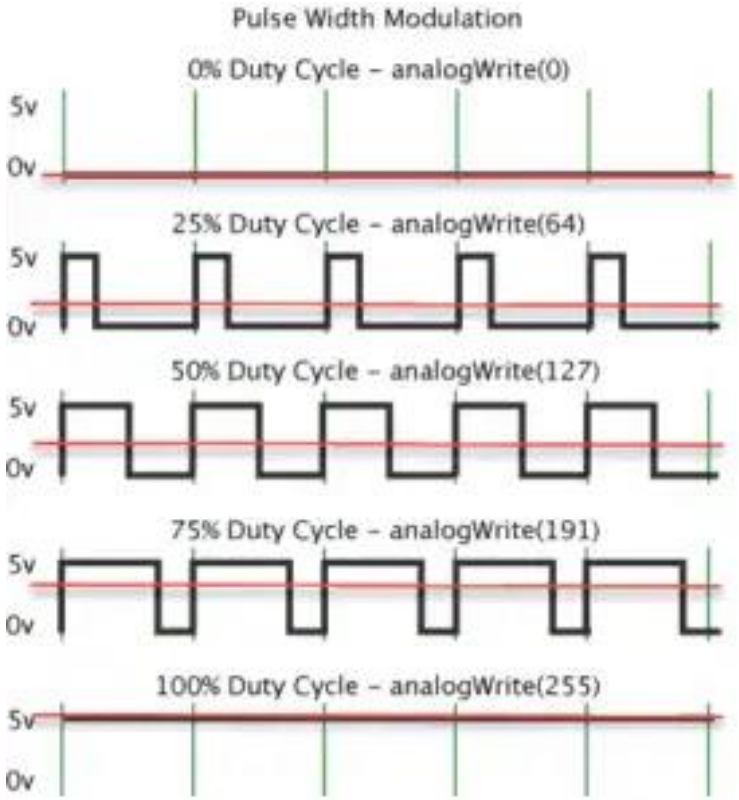
Entre los actuadores, utilizamos un LED para notificar, junto con el parlante, los cambios de estado del sistema embebido.
Un LED es un diodo de unión PN, es decir, tiene dos electrodos, uno de ellos contiene electrones libres, y el otro huecos. Cuando se le aplica corriente, los electrones libres se re-acomodan en los huecos, liberando a su paso fotones que provocan el efecto conocido como electroluminiscencia. El colorel color de la luz generada (que depende de la energía de los fotones emitidos) viene determinado por la anchura de la banda prohibida del semiconductor.




El LED que utilizamos es un LED rojo difuso, hecho de fosfuro de galio y arsénico (GaAsP), que tiene las siguiente propiedades:

Voltaje Min: 1.8v
Voltaje Max 2.2v
Consumo (mW): 40

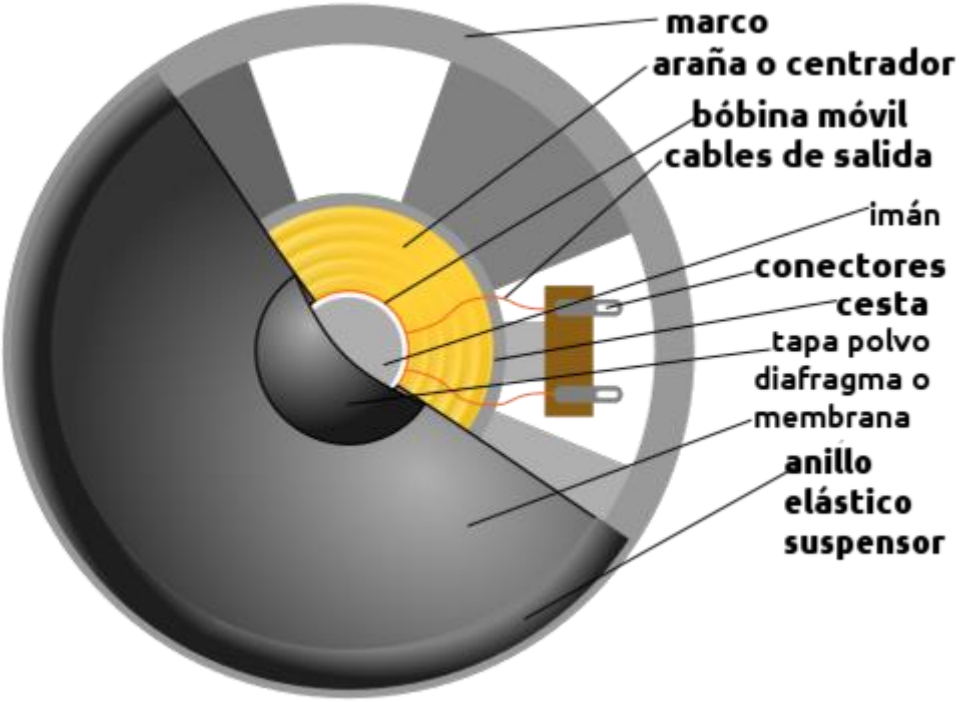
Para que el LED se encienda y apague progresivamente, tuvimos que controlar la intensidad de la luz emitida mediante PWM. Esto significa que para obtener el 50% de la intensidad total, tuvimos que enviar pulsos digitales, con un duty cycle del 50% del ancho del ciclo. Para una menor intensidad, el duty cycle debe ser un porcentaje menor.




En Arduino, se puede enviar dicha señal usando la función analogWrite(ledPin, value); donde value es un valor entre **0** y **255**, correspondiente al porcentaje de duty cycle que se quiere obtener. En nuestro caso, se aumentó progresivamente el valor que tomaba value entre los valores extremos para que se encienda gradualmente, y se disminuyó progresivamente entre esos valores para apagarlo gradualmente.

## Parlante

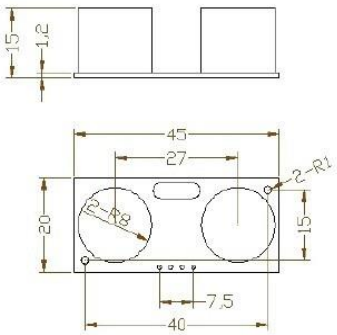
Un parlante es un transductor electro-acústico (eléctrico - mecánico - acústico, en realidad) compuesto por un imán, una bobina móvil y un cono principalmente, que convierte las ondas eléctricas en energía mecánica, y luego convierte la energía mecánica en ondas de frecuencia acústica




Para generar las notas correspondientes en el parlante, utilizamos la función tone() de Arduino. Esta función toma como parámetros el pin donde está conectado el parlante, la frecuencia del sonido a generar, y la duración del mismo (que es opcional, ya que si no se indica ninguna, sonará la misma nota hasta que se llame a la función notone() ), y genera una onda cuadrada de la frecuencia especificada con un duty cycle del 50%. Esto es lo que diferencia este método de PWM, ya que lo que cambia es la frecuencia mientras que el duty cycle es siempre el mismo, contrario a lo que sucede en PWM.
No es posible generar tonos inferiores a 31 Hz.

## Sensor Ultrasónico

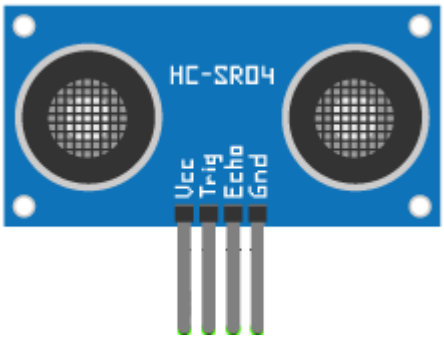
Se optó por emplear un sensor ultrasónico para poder detectar los objetos frente al auto, que pudieran ocasionar colisiones.
En este caso, el modelo utilizado es el HC-SR04, siendo este uno de los más conocidos por su performance estable, y una precisión de alto rango

## Especificaciones técnicas

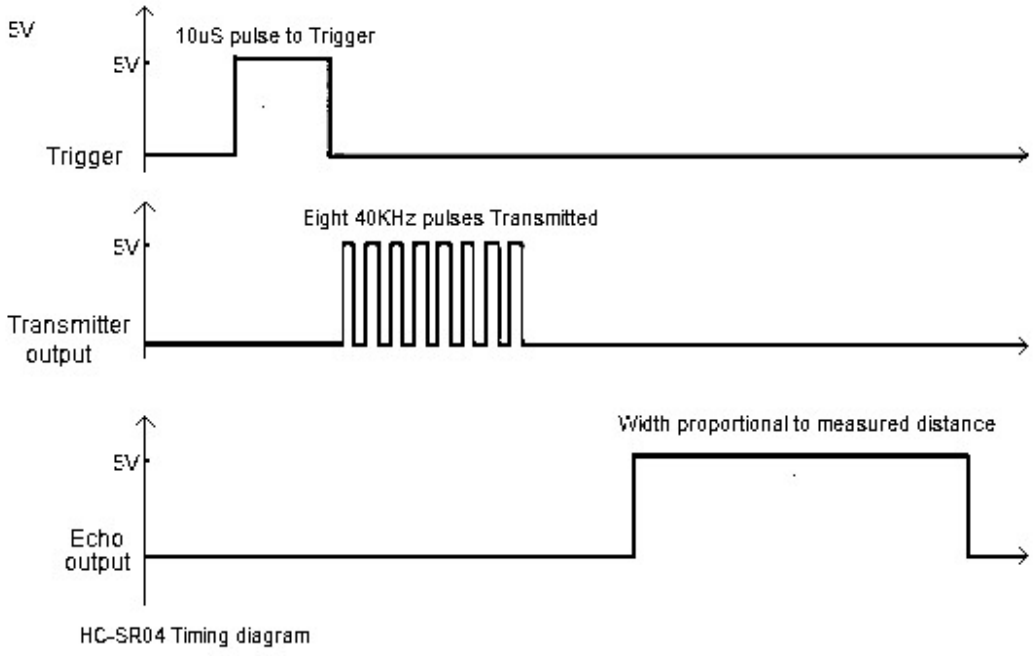
- Sensor ultrasonico para arduino y otros microcontroladores
- Tensión de operación: +5v
- Corriente nominal: 15mA
- Rango: De 2cm a 400cm (Con un error de 3mm)
- Ángulo de medición: 15 grados
- Compatible con TTL
- Dimensiones: 45*20*15mm

## Conexiones




El módulo HC-SR04 dispone de dos pares de pines de conexión. Un par de pines corresponde a la alimentación del módulo (**VCC** y **GND**). El otro par de pines (**TRIG** y **ECHO**) corresponde a la medición de la distancia. Estos dos pines se conectan directamente a dos pines digitales del Arduino, el que se conecte a **TRIG** será de salida y el que se conecte a **ECHO** será de entrada.

## Modo de uso




HC-SR04 Timing diagram

En lo que refiere a su funcionamiento, partiendo de un tiempo cero, se transmite un pulso ultrasónico corto, el cual en caso de impactar contra un objeto, hará que se refleje. El sensor recibe una señal y la convierte en una señal eléctrica.
Al periodo de tiempo se lo conoce como “Periodo de Ciclo”. El Periodo de Ciclo recomendado no debe ser menor a 50 milisegundos. Para operar se envía un pulso disparador (Trigger Pulse) de 10µs de ancho al pin de señal (Signal Pin). Esto hace que el modulo ultrasonico emita 8 pulsos de 40 KHz, y luego se detecta el eco que se produce en

respuesta (si no se detecta un obstáculo, el pin de salida enviará una señal alta de 38 milisegundos)
Luego se mide este ancho de pulso y se calcula la distancia siguiendo la fórmula:

$$d\ [cm]\ =\ t\ [\mu s]\ /\ v_s[\frac{\mu s}{cm}]\ /\ 2$$

Donde las variables son:

- **d** = distancia al objeto, en centímetros
- **t** = ancho de pulso del eco, en microsegundos
- **$v_s$** = velocidad del sonido, en microsegundos por centímetros

Esta fórmula sale de la definición de distancia, que es velocidad por tiempo. Al querer calcular la distancia a la que estamos del objeto, tenemos que recordar que el tiempo medido es el tiempo de ida y vuelta del pulso, por lo que se divide a la mitad el valor final. También invertimos las unidades de la velocidad para simplificar la fórmula dentro del código y no realizar cambio de unidades. El valor de la velocidad del sonido es 343 m/s

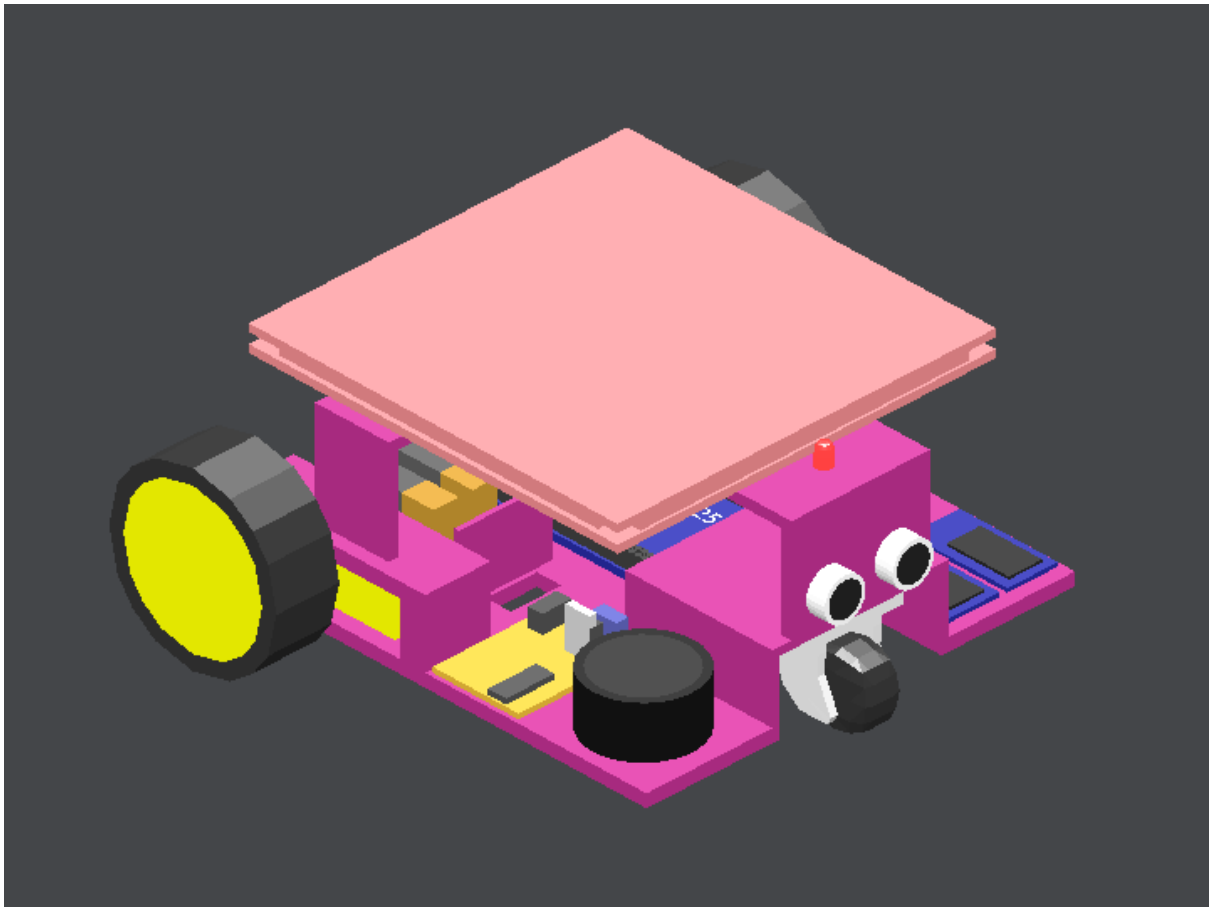
$$v_s = \ 343\ \frac{m}{s} = 34300\ \frac{cm}{s} = 0.0343 \frac{cm}{\mu s}\ \simeq 29 \frac{\mu s}{cm}$$

Por lo que obtenemos que la velocidad del sonido equivale a 29 microsegundos por centímetro aproximadamente, que es el valor que utilizamos en nuestra fórmula.

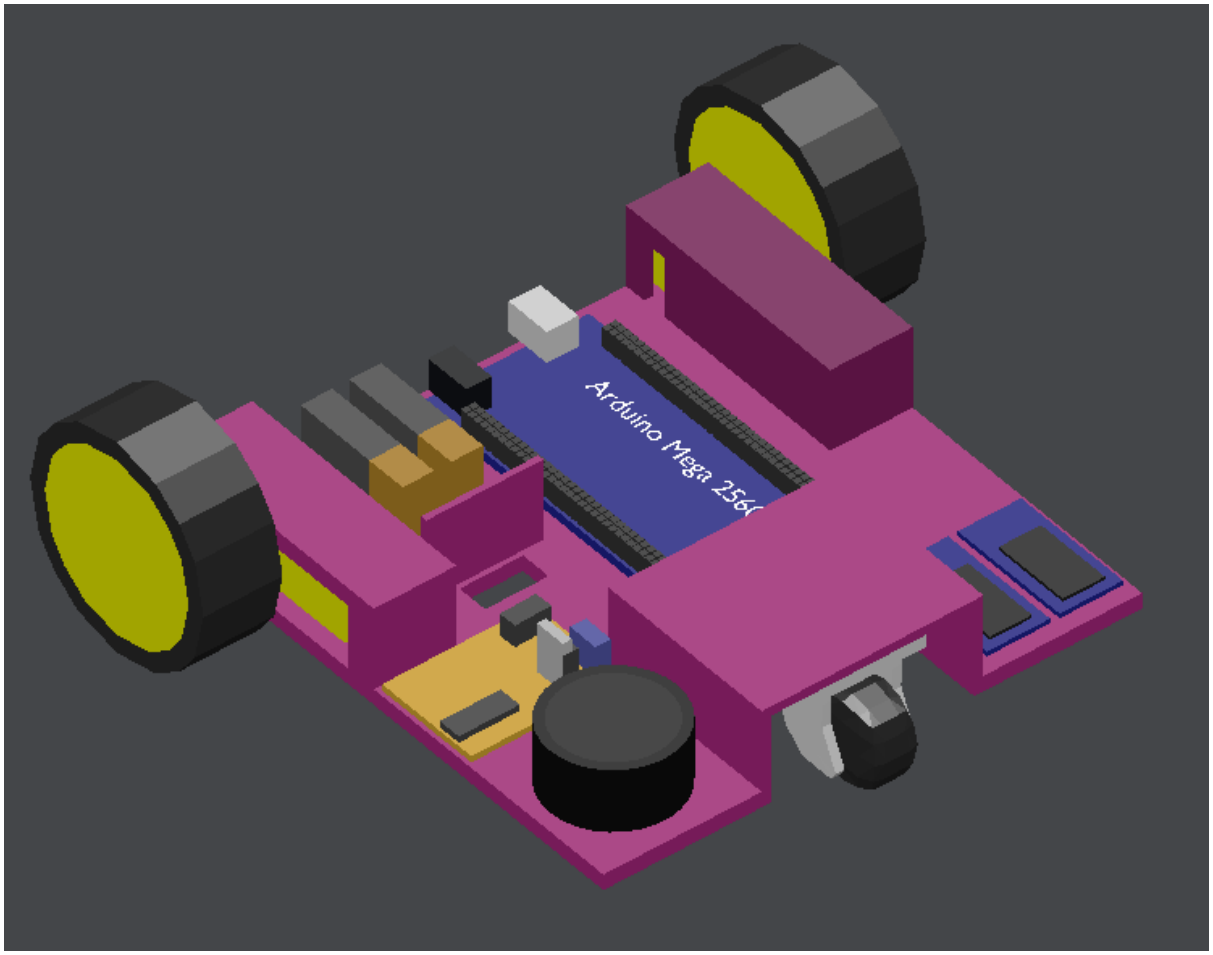
Elegimos estas unidades ya que para medir el ancho de pulso utilizamos la función de arduino `pulseIn`, que nos devuelve el ancho de un pulso de entrada de un pin de lectura en microsegundos. Al tener el ancho de pulso en microsegundos, decidimos modificar las unidades de la fórmula para que los valores de las constantes no sean ni muy grandes ni muy pequeños, y podemos guardar estas constantes en un entero.

# Modelos 3D y su rol

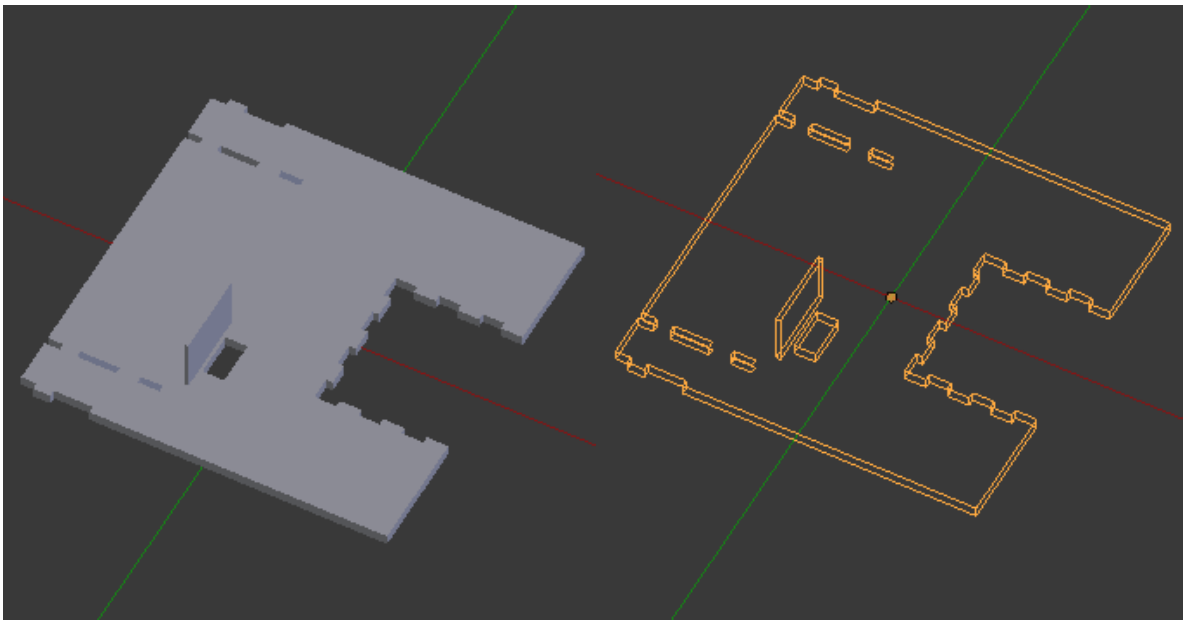
La estructura base donde montamos el robot fue diseñada con un software open-source para creación de objetos 3D (Blender), e impreso posteriormente con una impresora 3D.



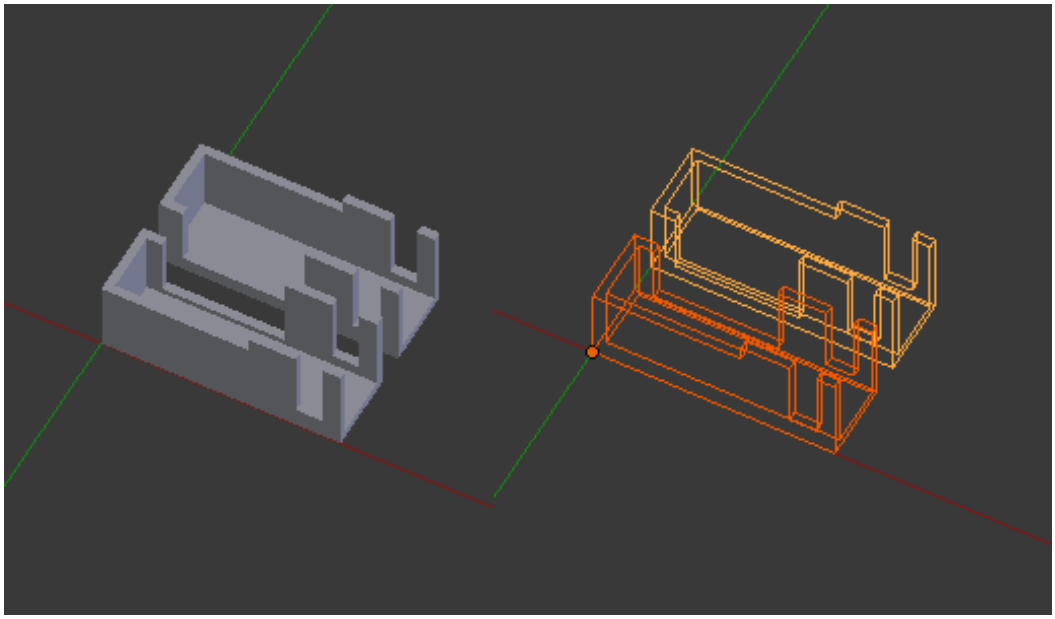
**Modelo completo**



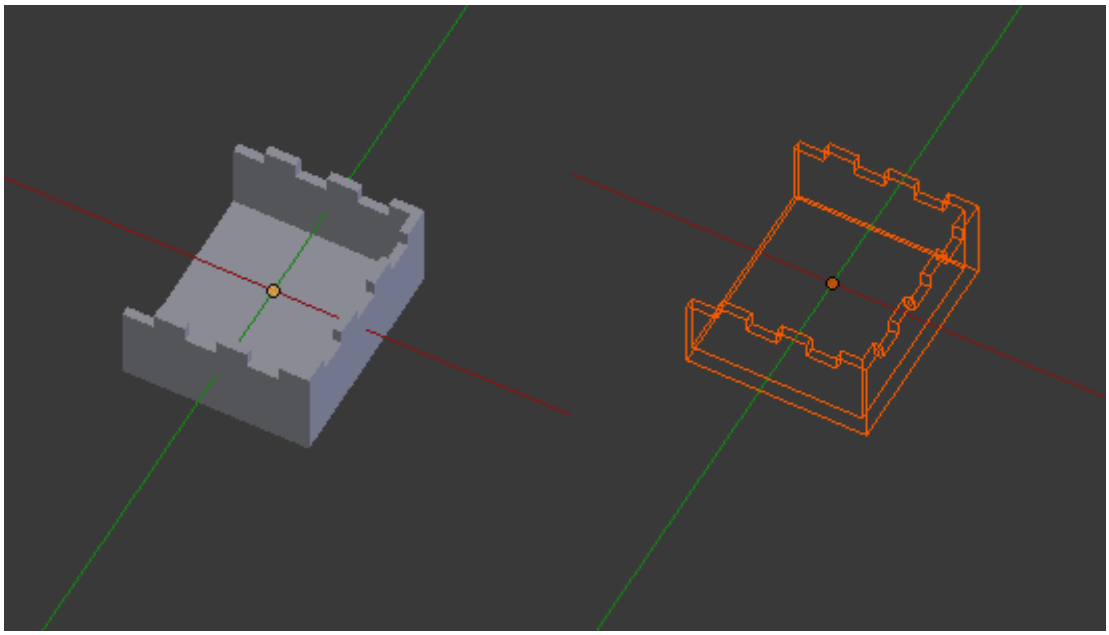

**Modelo sin los componentes superiores**



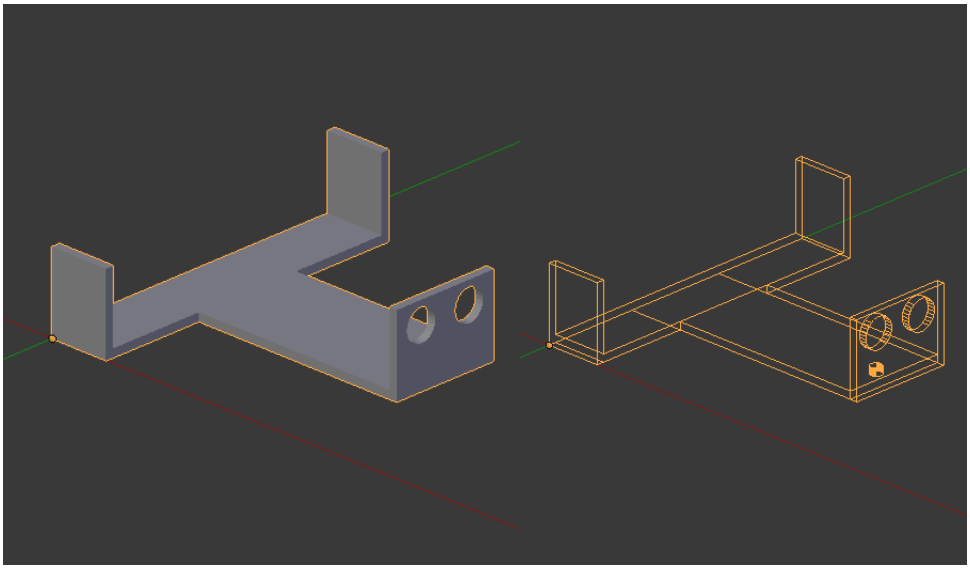
**Base**



**Soporte para motores**



**Soporte para rueda loca**



**Tripode (soporte para sensor de ultrasonido, placa de presión y Led)**

# Problemas SE y sus soluciones

## Bluetooth

El problema más grande que tenemos es determinar la distancia hacia el destino basándonos en las lecturas de RSSI (nivel de fuerza de la señal recibida) del módulo Bluetooth HC-05.
El primer inconveniente encontrado fue que el módulo no viene preparado para entrar en modo AT (modo en el cual se pueden utilizar sólamente comandos para ordenarle al módulo que realice acciones), por lo cual tuvimos que soldar un cable a un pin del submódulo, el cual recibe una señal en HIGH, antes de poner en HIGH el pin de VCC del módulo, para entrar en dicho modo.
El segundo inconveniente, que no pudo ser solucionado completamente, es que la señal de RSSI medida no es confiable. La precisión es muy mala, lo cual puede ser observado en las pruebas que tomamos, como lo muestra la figura 1, donde varias muestras tomadas a la misma distancia obtienen valores muy diferentes entre sí.

Para realizar las pruebas, tomamos mediciones cada 0.25m hasta llegar a 3m de distancia con respecto al objeto que emitía la señal Bluetooth. En cada distancia, tomamos 10 muestras, y al finalizar completamos una tabla, que se puede ver en la figura 2, indicando valor el valor mínimo, el máximo, el promedio, la mediana y la moda para las muestras tomadas.

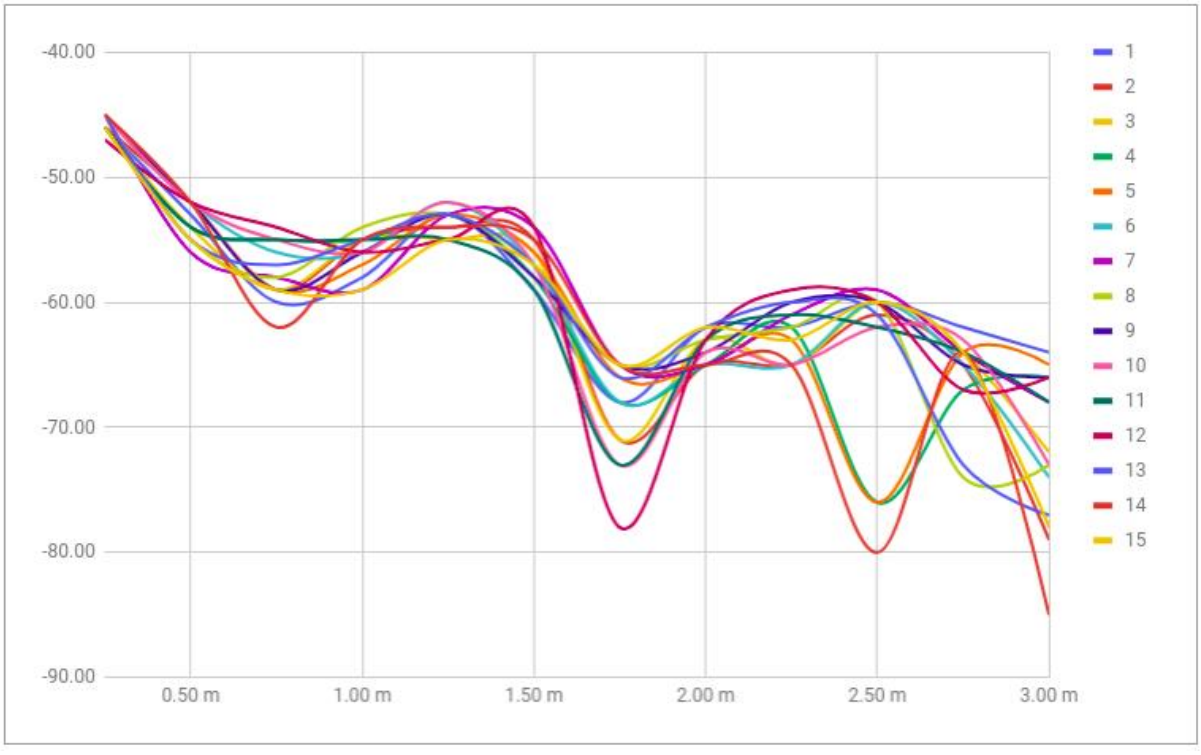



**Figura 1. Muestras tomadas para cada distancia**

| Distancia | 0.25 m | 0.50 m | 0.75 m | 1.00 m | 1.25 m | 1.50 m | 1.75 m | 2.00 m | 2.25 m | 2.50 m | 2.75 m | 3.00 m |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N° muestra | DBm | DBm | DBm | DBm | DBm | DBm | DBm | DBm | DBm | DBm | DBm | DBm |
| 1 | -46.00 | -53.00 | -60.00 | -58.00 | -53.00 | -59.00 | -68.00 | -62.00 | -62.00 | -60.00 | -62.00 | -64.00 |
| 2 | -46.00 | -52.00 | -59.00 | -55.00 | -54.00 | -55.00 | -71.00 | -65.00 | -65.00 | -61.00 | -65.00 | -79.00 |
| 3 | -46.00 | -54.00 | -59.00 | -55.00 | -55.00 | -56.00 | -71.00 | -63.00 | -65.00 | -60.00 | -64.00 | -72.00 |
| 4 | -46.00 | -54.00 | -55.00 | -55.00 | -53.00 | -58.00 | -68.00 | -65.00 | -62.00 | -76.00 | -67.00 | -66.00 |
| 5 | -45.00 | -52.00 | -59.00 | -57.00 | -53.00 | -56.00 | -66.00 | -65.00 | -63.00 | -76.00 | -64.00 | -65.00 |
| 6 | -45.00 | -52.00 | -56.00 | -56.00 | -52.00 | -57.00 | -68.00 | -65.00 | -65.00 | -60.00 | -65.00 | -74.00 |
| 7 | -45.00 | -56.00 | -58.00 | -59.00 | -53.00 | -54.00 | -65.00 | -65.00 | -61.00 | -59.00 | -64.00 | -68.00 |
| 8 | -45.00 | -55.00 | -58.00 | -54.00 | -53.00 | -57.00 | -65.00 | -63.00 | -62.00 | -60.00 | -74.00 | -73.00 |
| 9 | -45.00 | -52.00 | -59.00 | -56.00 | -53.00 | -58.00 | -65.00 | -64.00 | -60.00 | -60.00 | -65.00 | -66.00 |
| 10 | -45.00 | -52.00 | -55.00 | -56.00 | -52.00 | -57.00 | -73.00 | -64.00 | -65.00 | -62.00 | -63.00 | -73.00 |
| 11 | -46.00 | -54.00 | -55.00 | -55.00 | -55.00 | -59.00 | -73.00 | -63.00 | -61.00 | -62.00 | -64.00 | -68.00 |
| 12 | -47.00 | -52.00 | -54.00 | -56.00 | -55.00 | -54.00 | -78.00 | -63.00 | -59.00 | -60.00 | -67.00 | -66.00 |
| 13 | -45.00 | -55.00 | -57.00 | -55.00 | -53.00 | -57.00 | -66.00 | -62.00 | -60.00 | -61.00 | -73.00 | -77.00 |
| 14 | -45.00 | -52.00 | -62.00 | -55.00 | -54.00 | -55.00 | -65.00 | -65.00 | -65.00 | -80.00 | -64.00 | -85.00 |
| 15 | -46.00 | -55.00 | -59.00 | -59.00 | -55.00 | -57.00 | -65.00 | -62.00 | -63.00 | -60.00 | -64.00 | -78.00 |
| Minimo | -47.00 | -56.00 | -62.00 | -59.00 | -55.00 | -59.00 | -78.00 | -65.00 | -65.00 | -80.00 | -74.00 | -85.00 |
| Maximo | -45.00 | -52.00 | -54.00 | -54.00 | -52.00 | -54.00 | -65.00 | -62.00 | -59.00 | -59.00 | -62.00 | -64.00 |
| Promedio | -45.53 | -53.33 | -57.67 | -56.07 | -53.53 | -56.60 | -68.47 | -63.73 | -62.53 | -63.80 | -65.67 | -71.60 |
| Mediana | -45.00 | -53.00 | -58.00 | -56.00 | -53.00 | -57.00 | -68.00 | -64.00 | -62.00 | -60.00 | -64.00 | -72.00 |
| Moda | -45.00 | -52.00 | -59.00 | -55.00 | -53.00 | -57.00 | -65.00 | -65.00 | -65.00 | -60.00 | -64.00 | -66.00 |

**Figura 2. Tabla con todos los valores**

Decidimos usar la moda (el valor más reincidente en las mediciones tomadas) porque el promedio se iba a ver influenciado por las mediciones que estuvieran más fuera de rango, como también pasaría con la mediana.

Incluso usando la moda de los valores obtenidos en una distancia, las mediciones seguían siendo poco confiables, por lo cual, incorporamos un filtro de Kalman para 1 dimensión simplificado.
Kalman es un algoritmo que tiene en cuenta parámetros de entrada con respecto al ruido del sensor y del proceso, establecidos con anticipación, y realiza ajustes dinámicamente a la covarianza de la estimación del error, acorde a las mediciones anteriores.

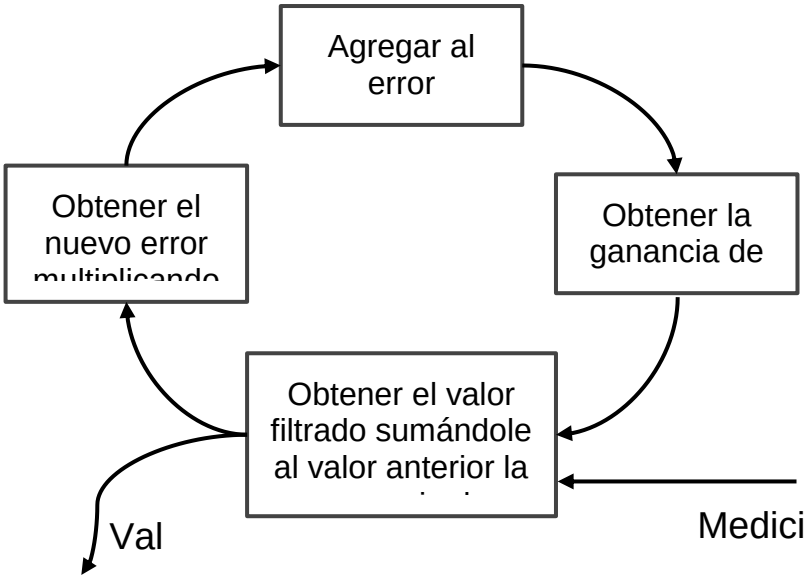
Siendo:

$$q = covarianza\ del\ ruido\ del\ proceso$$
$$r = covarianza\ del\ ruido\ de\ la\ medición\ (del\ sensor)$$
$$x = valor\ de\ interés\ (valor\ después\ de\ filtrar)$$
$$p = covarianza\ de\ la\ estimación\ del\ error$$
$$k = ganancia\ de\ Kalman$$
$$medicion = valor\ sin\ filtrar$$

$$p = p + q$$
$$k = \frac{p}{p + r}$$
$$x = x + k * (medicion - x)$$
$$p = (1 - k) * p$$

Donde los valores de $k$ y $p$ se modifican a medida se tomen más mediciones para obtener mayor precisión en el valor que devuelve el filtro de Kalman.

Por lo tanto, Kalman funciona de la siguiente manera:




Entonces, luego de filtrar los valores de prueba, obtuvimos los valores que se pueden ver en la figura 3. Comparándolos con los valores de la moda calculada basada en las muestras medidas, podemos ver que es mucho más exacta la medición ahora, pero no alcanza para determinar correctamente la distancia relativa, por lo cual consideramos que no es un buen método para encontrar un destino, sobretodo porque se ve muy influenciado por el ambiente.
Como solución alternativa, implementamos un sistema de manejo manual, donde se le otorga el control del manejo de Carrybot a la aplicación servidor de Android.

| Distancia | Moda | Kalman |
|---|---|---|
| **0.25 m** | -45.00 | -43.64 |
| **0.50 m** | -52.00 | -47.76 |
| **0.75 m** | -59.00 | -51.49 |
| **1.00 m** | -55.00 | -52.38 |
| **1.25 m** | -53.00 | -52.50 |
| **1.50 m** | -57.00 | -53.28 |
| **1.75 m** | -65.00 | -55.03 |
| **2.00 m** | -65.00 | -56.35 |
| **2.25 m** | -65.00 | -57.39 |
| **2.50 m** | -60.00 | -57.68 |
| **2.75 m** | -64.00 | -58.33 |
| **3.00 m** | -66.00 | -59.07 |

**Figura 3. Tabla comparativa del valor de la moda antes y después de filtrar**

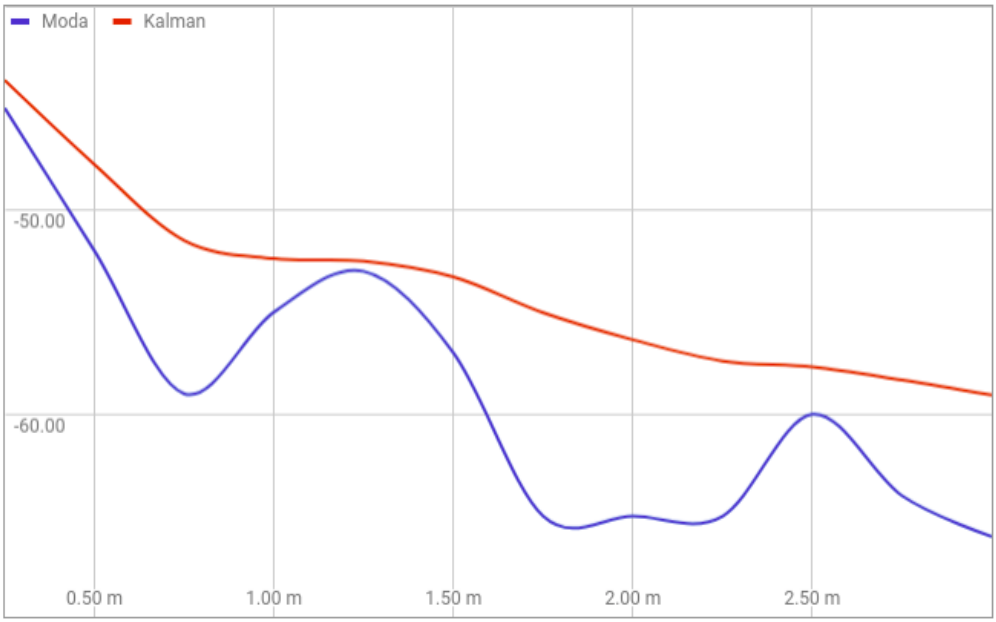



**Figura 4. Comparación entre las modas obtenidas antes y después de filtrarlas**

## Alimentación

Uno de los problemas más grandes que nos encontramos fue la alimentación de los circuitos. Originalmente probamos alimentar todo con una pila de 9v. Con el consumo de los motores y la placa Arduino en sí, no llegaban a accionar los motores siquiera.
Una solución, que está lejos de ser la óptima, fue separar la alimentación en dos pilas de 9v. Una de ellas, dedicada al Arduino, y la otra para alimentar los motores y el integrado que utilizamos como puente H (L293D). De esta manera, si bien el consumo es muy alto y poco eficiente, logramos su funcionamiento correcto.

Por otro lado, estuvimos investigando otras soluciones a este problema, dado que las pilas de 9v tienen muy poca capacidad en cuanto a mAh (alrededor de 600mAh, dando una autonomía muy baja al robot). Estas soluciones parten de este concepto, sugiriendo usar, mínimamente pilas AA en serie hasta alcanzar el voltaje necesario, pero como las pilas AA suelen tener hasta 4 veces la capacidad de las pilas de 9v, se obtendrían mejores resultados.

# Android

## Aplicación

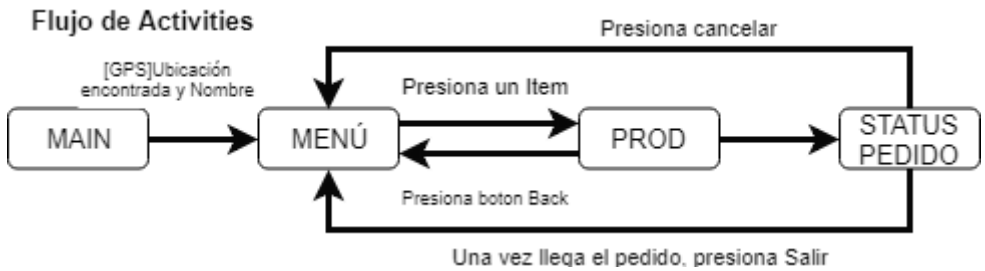
Disclaimer: Las aplicaciones de Carrybot están desarrolladas para la plataforma Android, orientadas a dispositivos con Android 5.0 en adelante, cubriendo así al 71.3% de dispositivos aproximadamente.

La plataforma ofrecida por Carrybot está compuesta por dos aplicaciones: una aplicación para los clientes y una aplicación para el servidor. Están diseñadas para manejar los pedidos de un restaurant, y comunicarse con Carrybot para cumplir dicho objetivo.

A continuación, se detalla la funcionalidad de cada una y cómo interactúan entre sí.

### Cliente

Orientada al cliente del restaurant Carrybot, permite que dicho cliente acceda al menú del restaurant y haga pedidos, mediante el servidor de mensajería en la nube de Firebase, a la sede del restaurant en la que se encuentra (información que se determina mediante el uso del sensor GPS del dispositivo). También tendrá información disponible sobre el estado del pedido que realizó.




### Servidor

Orientada al personal del restaurant Carrybot, permite que el empleado se comunique con Carrybot via Bluetooth para conectarse y configurar los parámetros de inicio (que varían en cuanto al contexto), y conectarse al servidor de Firebase correspondiente a la sede (determinado automáticamente mediante el uso del sensor GPS del dispositivo) para recibir los pedidos emitidos por los clientes y el listado de los mismos.
Mediante el sensor de luz ambiental determina si el restaurant está abierto o cerrado para empezar o finalizar de operar.
Mantiene una lista de los pedidos recibidos, y se pueden cancelar los mismos agitando el dispositivo, comunicándole de manera automática los cambios a Carrybot.
También cuenta con un modo manual de controlar el camino que realiza Carrybot, en caso de que el método primario no sea posible.

**Flujo de Activities**

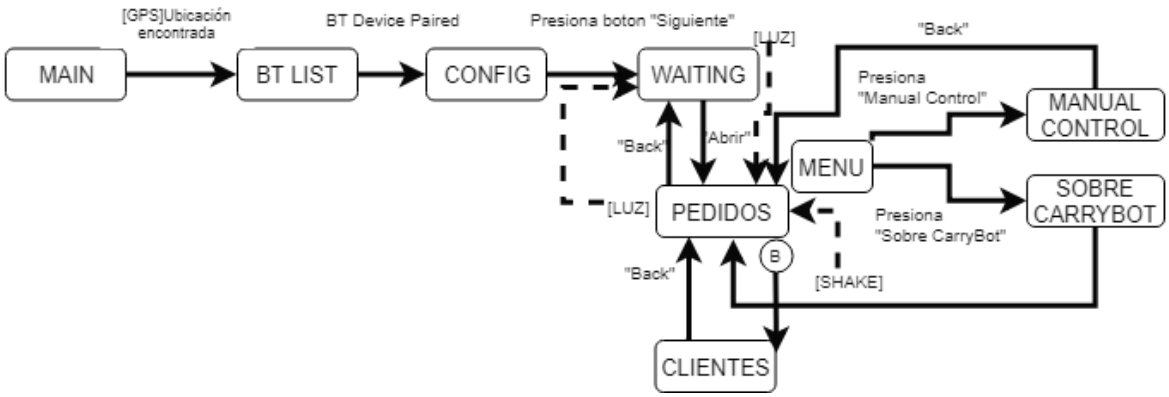
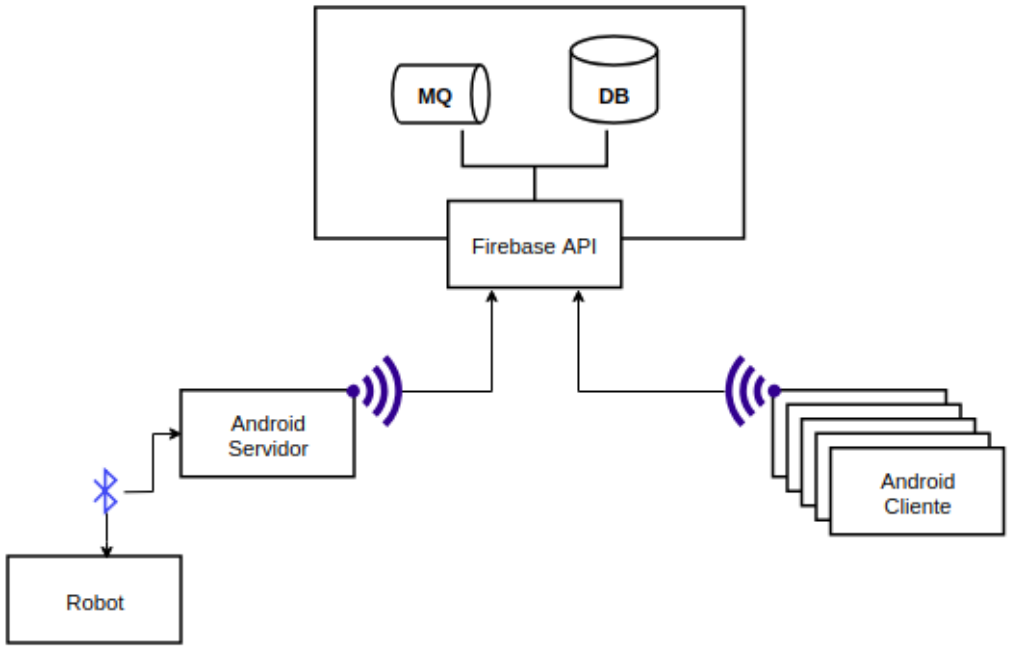



Diagrama de comunicación general



# Mejoras para el producto

## Tecnología Bluetooth para rastrear el destino

Si bien, actualmente CarryBot alcanza su destino obteniendo mediciones de los dispositivos Bluetooth cercanos, es decir, su RSSI (Nivel de ruido en la señal), posteriormente aplicando la moda, y finalmente, Kalman a la misma, este método no es del todo preciso:

1. El Nivel de Ruido en la Señal, como se expone más adelante, presenta un elevado margen de error. CarryBot podría estar cerca del destino y obtener un valor muy bajo de RSSI, o estar muy alejado y obtener un valor elevado.
2. Aun empleando métodos probabilísticos, y Kalman, se torna muy impreciso, puesto que a veces los valores se desvirtúan respecto de lo lógicamente esperado.
3. Al menos en lo que concierne al módulo HC-05, emplear el modo AT (el cual es necesario para utilizar los comandos que se mencionan en los anexos posteriores) puede ser un verdadero dolor de cabeza, puesto que si no se envia HIGH a los pines en el orden correcto, y durante un cierto periodo de tiempo, puede no activarse el modo AT, o entrar en un modo AT "Buggeado", el cual no tiene utilidad alguna.

Dicho esto, consideramos que la incorporación de un método más preciso para alcanzar el destino al que CarryBot quiere ir, es imperativo.

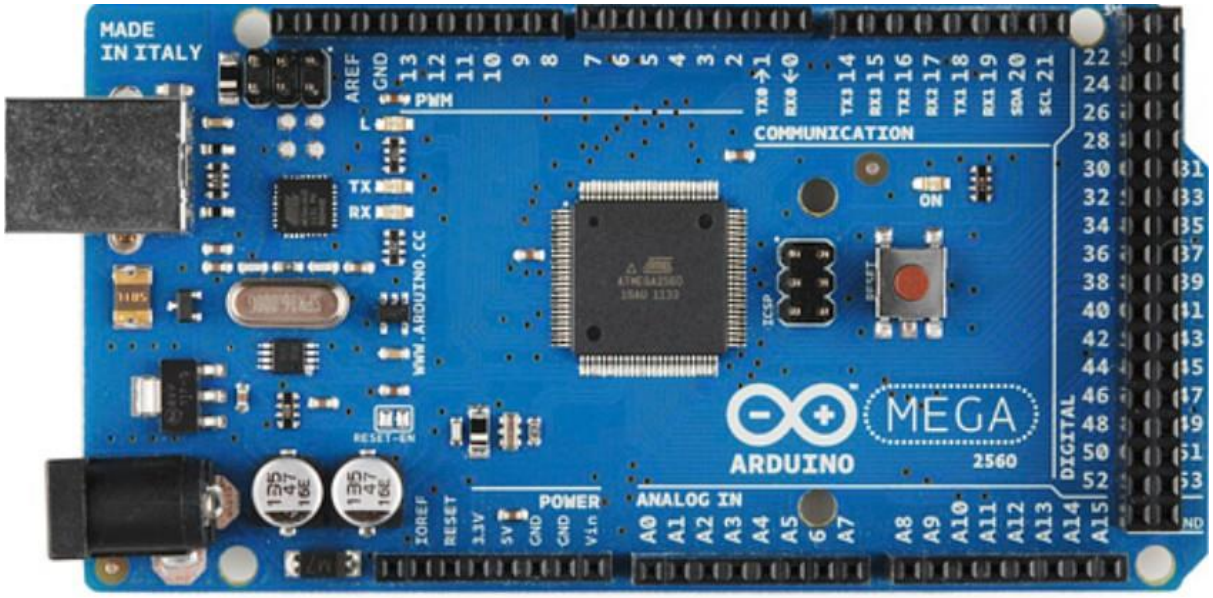
## Cantidad de productos por pedido

Actualmente, la aplicación de Cliente está preparada para que el mismo solo pueda pedir una cierta cantidad de un producto. Una mejora sería permitirle al mismo seleccionar diferentes cantidades, de diferentes productos, para el mismo viaje.
Dicho esto, si se quisiera aplicar este cambio, se debe si o si considerar el cambio *"Forma y soporte de la bandeja"*.

## Forma y soporte de la bandeja.

Si bien actualmente, la placa de presión que actúa como bandeja cumple su función, si se deseara aplicar la mejora *"Cantidad de productos por pedido"*, sería necesario alterar la forma de la bandeja, no solo en cuanto al tamaño de la base, sino también, dotándola de una forma semejante a la de una caja, permitiendo conservar la temperatura de los pedidos, y a su vez, reducir el riesgo de que se caigan.

# Anexo I: Arduino Mega 2560




Arduino Mega 2560 es una versión ampliada de la tarjeta original de Arduino y está basada en el microcontrolador Atmega2560.
Dispone de 54 entradas/salidas digitales, 14 de las cuales se pueden utilizar como salidas PWM (modulación de anchura de pulso). Además dispone de 16 entradas analogicas, 4 UARTs (puertos serie), un oscilador de 16MHz, una conexión USB, un conector de alimentación, un conector ICSP y un pulsador para el reset.

## Alimentación

El Arduino Mega puede ser alimentado vía la conexión USB o con una fuente de alimentación externa. El origen de la alimentación se selecciona automáticamente.
Las fuentes de alimentación externas (no-USB) pueden ser tanto un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1mm con centro positivo en el conector hembra de la placa. Los cables de la batería puede conectarse a los pines Gnd y Vin en los conectores de alimentación (POWER)
La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. Si el voltaje suministrado es inferior a 7V, el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable; si se usan mas de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.
Los pines de alimentación son los siguientes:

- **VIN:** La entrada de voltaje a la placa Arduino cuando se está usando una fuente externa de alimentación (en opuesto a los 5 voltios de la conexión USB). Se puede proporcionar voltaje a través de este pin, o, si se está alimentando a través de la conexión de 2.1mm , acceder a ella a través de este pin.

- **5V:** La fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de VIN a través de un regulador integrado en la placa, o proporcionada directamente por el USB u otra fuente estabilizada de 5V.
- **3.3V:** Una fuente de voltaje de 3.3 voltios generada por un regulador integrado en la placa. La corriente máxima soportada 50mA.
- **GND:** Pines de toma de tierra.

## Memoria

El ATmega2560 tiene 256KB de memoria flash para almacenar código, de los cuales 8KB son usados para el arranque del sistema (bootloader). El ATmega2560 tiene 8 KB de memoria SRAM y 4KB de EEPROM, a la cual se puede acceder para leer o escribir con la librería EEPROM.

## Entradas Y Salidas

### Entradas y salidas digitales

Cada uno de los 54 pines digitales en el Mega pueden utilizarse como entradas o como salidas usando las funciones pinMode(), digitalWrite(), y digitalRead(). Las E/S operan a 5 voltios. Cada pin puede proporcionar o recibir una intensidad máxima de 40 mA y tiene una resistencia interna de pull-up (desconectada por defecto) de 20-50k Ohms. Además, algunos pines tienen funciones especializadas:

- **Serie 0 (RX) y 1 (TX),Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX)**
  Usados para recibir (RX) transmitir (TX) datos a través de puerto serie TTL. Los pines Serie: 0 (RX) y 1 (TX) están conectados a los pines correspondientes del chip FTDI USB-to-TTL.
- **Interrupciones Externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2).**
  Estos pines se pueden configurar para lanzar una interrupción en un valor LOW(0V), en flancos de subida o bajada (cambio de LOW a HIGH(5V) o viceversa), o en cambios de valor. Ver la función attachInterrupt() para más detalles.
- **PWM: de 0 a 13.**
  Proporciona una salida PWM (Pulse Wave Modulation, modulación de onda por pulsos) de 8 bits de resolución (valores de 0 a 255) a traves de la función analogWrite().
- **SPI: 50 (SS), 51 (MOSI), 52 (MISO), 53 (SCK).**
  Estos pines proporcionan comunicación SPI, usando la librería SPI.
- **LED: 13.**
  Hay un LED integrado en la placa conectado al pin digital 13, cuando este pin tiene un valor HIGH(5V) el LED se enciende y cuando este tiene un valor LOW(0V) este se apaga.

### Entradas y salidas analogicas

El Mega tiene 16 entradas analógicas, y cada una de ellas proporciona una resolución de 10bits (1024 valores). Por defecto se mide desde 0V a 5V, aunque es posible cambiar la cota superior de este rango usando el pin AREF y la función analogReference().

- **I2C: 20 (SDA) y 21 (SCL).**
  Soporte para el protocolo de comunicaciones I2C (TWI) usando la librería Wire.
- **AREF.**
  Voltaje de referencia para la entradas analogicas. Usado por analogReference().
- **Reset.**
  Suministrar un valor LOW (0V) para reiniciar el microcontrolador. Típicamente usado para añadir un botón de reset a los shields que no dejan acceso a este botón en la placa.

## Comunicaciones

EL Arduino Mega facilita en varios aspectos la comunicación con la PC, otro Arduino u otros microcontroladores. El ATmega2560 proporciona cuatro puertos de comunicación vía serie UART TTL (5V). Un ATmega16U2 integrado en la placa canaliza esta comunicación serie a través del puerto USB y los drivers (incluidos en el software de Arduino) proporcionan un puerto serie virtual en el ordenador. El software incluye un monitor de puerto serie que permite enviar y recibir información textual de la placa Arduino. Los LEDS RX y TX de la placa parpadean cuando se detecta comunicación transmitida través de la conexión USB (no parpadean si se usa la comunicación serie a través de los pines 0 y 1).
La librería SoftwareSerial permite comunicación serie por cualquier par de pines digitales del Mega.
El ATmega2560 también soporta la comunicación I2C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso el bus I2C.

## Programación

El Arduino Mega se puede programar con el software gratuito de Arduino.
El ATmega2560 en el Arduino Mega viene precargado con un gestor de arranque (bootloader) que permite cargar nuevo código sin necesidad de un programador por hardware externo. Se comunica utilizando el protocolo STK500 original.
También puede evitarse el gestor de arranque y programar directamente el microcontrolador a través del puerto ICSP (In Circuit Serial Programming).

# Anexo II: Protocolo de comunicación

Para la comunicación entre el embebido y el servidor basado en una aplicación android disponemos de los siguientes comandos enviados al arduino y las respuestas del mismo

## Comandos

A continuación la lista de comandos que se envían desde la aplicación android al embebido. Todos los comandos deben terminar con un salto de linea (carácter “\n”).

| **Comando** | **Descripción** |
|---|---|
| PING | Se utiliza este comando para indicar el inicio de la comunicación entre el servidor Android y el embebido. |
| SET+MAC= | Se utiliza este comando durante la configuración para configurar la MAC del dispositivo Bluetooth, va seguido de la dirección MAC del servidor. |
| SET+LOOPS= | Se utiliza este comando durante la configuración para configurar la cantidad de veces que avanza CarryBot antes de volver a consultar la señal Bluetooth. Va seguido de un número entero. |
| SET+THRESH= | Se utiliza este comando durante la configuración para configurar la diferencia de valor mínima que debe variar la señal para considerar que CarryBot modificó su distancia del objetivo. Va seguido de un número entero que representa el valor absoluto de la diferencia. |
| SET+TGIRO= | Se utiliza este comando durante la configuración para configurar en milisegundos el tiempo de giro de CarryBot y así regular cuanto gira cada vez que lo hace. Va seguido de un número entero. |
| SET+LLEGADA= | Se utiliza este comando durante la configuración para setear la intensidad de señal mínima para considerar que CarryBot llegó a destino. Va seguido de un número entero positivo que representa el valor absoluto de la señal. |
| SET+DONE | Se utiliza este comando durante la configuración para indicar que ha terminado la etapa de configuración. |
| MANUAL | Se utiliza para cambiar al modo Manual estando en el modo Automático. |
| AUTOMATIC | Se utiliza para cambiar al modo Automático estando en el modo Manual. |
| CANCEL | Se utiliza para cancelar el viaje actual hacia el cliente. |
| MOVE+LEFT<br>MOVE+RIGHT | Se utilizan para conducir a CarryBot estando en el modo manual. |

| | |
|---|---|
| MOVE+FORWARD<br>MOVE+BACKWARD | |
| MOVE+ARRIVED | Se utiliza para avisar que se llegó a destino estando en modo manual. |

## Respuestas

A continuación la lista de respuestas e información que el embebido enviar por Bluetooth para informar de su estado a la aplicación.

| **Respuesta** | **Descripción** |
|---|---|
| PONG | Se utiliza para responder al comando PING |
| SET+OK | Se utiliza para informar que todo fue bien durante la configuración de algún parámetro |
| SET+ERROR | Se utiliza para informar que hubo algún error durante la configuración de algún parámetro |
| STATUS+ONTRAVEL | Se utiliza para informar que CarryBot comenzó a viajar hacia su destino |
| STATUS+RBLOCK | Se utiliza para informar que CarryBot no puede avanzar porque tiene un objeto enfrente |
| STATUS+ONDEST | Se utiliza para informar que CarryBot llegó a su destino |
| STATUS+ALOAD | Se utiliza para informar que CarryBot está esperando que se le cargue un objeto para transportar |
| STATUS+AUNLOAD | Se utiliza para informar que CarryBot está esperando que se le descargue un objeto transportado |
| STATUS+AMAC | Se utiliza para informar que CarryBot está esperando la dirección MAC del dispositivo cliente al que debe dirigirse |