



## PROYECTO FIN DE GRADO

**TÍTULO:** VladBot. Robot para posicionamiento de medida

**AUTOR:** Ángel Plata Rueda

**TUTOR (o Director en su caso):** Lino García Morales

**DEPARTAMENTO:** DIAC

**TITULACIÓN:** Grado en Ingeniería de Sonido e Imagen

VºBº

**Miembros del Tribunal Calificador:**

**PRESIDENTE:** Nicolás López Pérez

**TUTOR:** Lino García Morales

**SECRETARIO:** Danilo Simón Zorita

**Fecha de lectura:** 30 de septiembre de 2013

**Calificación:**

El Secretario,

## RESUMEN

*VladBot* es un robot autónomo diseñado para posicionar en interiores un micrófono de medida. Este prototipo puede valorar la idea de automatizar medidas acústicas en interiores mediante un robot autónomo. Posee dos ruedas motrices y una rueda loca. Ésta rueda loca aporta maniobrabilidad al robot. Un soporte extensible hecho de aluminio sostiene el micrófono de medida. *VladBot* ha sido diseñado con tecnologías de bajo coste y bajo una plataforma abierta, Arduino.

Arduino es una plataforma electrónica libre. Esto quiere decir que los usuarios tienen libre acceso a toda la información referente a los micro-controladores (*hardware*) y referente a el *software*. Ofrece un IDE (*Integrated Development Environment*, en español, Entorno de Desarrollo Integrado) de forma gratuita y con un sencillo lenguaje de programación, con el que se pueden realizar proyectos de cualquier tipo. Además, los usuarios disponen de un foro donde encontrar ayuda, “Arduino Forum”.

*VladBot* se comunica con el usuario a través de Bluetooth, creando un enlace fiable y con un alcance suficiente (aproximadamente 100 metros) para que controlar a *VladBot* desde una sala contigua. Hoy en día, Bluetooth es una tecnología implantada en casi todos los ordenadores, por lo que no necesario ningún sistema adicional para crear dicho enlace. Esta comunicación utiliza un protocolo de comunicaciones, JSON (*JavaScript Object Notation*). JSON hace que la comunicación sea más fiable, ya que sólo un tipo de mensajes preestablecidos son reconocidos. Gracias a este protocolo es posible la comunicación con otro *software*, permitiendo crear itinerarios en otro programa externo.

El diseño de *VladBot* favorece su evolución hasta un sistema más preciso ya que el usuario puede realizar modificaciones en el robot. El código que se proporciona puede ser modificado, aumentando las funcionalidades de *VladBot* o mejorándolas. Sus componentes pueden ser cambiados también (incluso añadir nuevos dispositivos) para aumentar sus capacidades. *Vladbot* es por tanto, un sistema de transporte (de bajo coste) para un micrófono de medida que se puede comunicar inalámbricamente con el usuario de manera fiable.

## PALABRAS CLAVE

Robot, posicionamiento, interiores, micrófono de medida, Bluetooth, Arduino...

## ABSTRACT

*VladBot* is an autonomous robot designed to indoor positioning of a measurement microphone. This prototype can value the idea of making automatic acoustic measurements indoor with an autonomous robot. It has two drive wheels and a caster ball. This caster ball provides manoeuvrability to the robot. An extendible stand made in aluminium holds the measurement microphone. *VladBot* has been designed with low cost technologies and under an open-source platform, Arduino.

Arduino is a free-source electronics platform. This means that users have free access to all the information about micro-controllers (hardware) and about the software. Arduino offers a free IDE (Integrated Development Environment) with an easy programming language, which any kind of project can be made with. Besides, users have a forum where find help, "Arduino Forum".

*VladBot* communicates with the user by Bluetooth, creating a reliable link with enough range (100 meters approximately) for controlling *VladBot* in the next room. Nowadays, Bluetooth is a technology embedded in almost laptops, so it is not necessary any additional system for create this link. This communication uses a communication protocol, JSON (JavaScript Object Notation). JSON makes the communication more reliable, since only a pre-established kind of messages are recognised. Thanks to this protocol is possible the communication with another software, allowing to create routes in an external program.

*VladBot's* design favours its evolution to an accurate system since the user can make modifications in the robot. The code given can be changed, increasing *VladBot's* uses or improving these uses. Their components can be changed too (even new devices can be added) for increasing its abilities. So, *VladBot* is a (low cost) transport system for a measurement microphone, which can communicate with the user in a reliable way.

## KEY WORDS

Robot, positioning, indoor, measurement microphone, Bluetooth, Arduino...

## ÍNDICE

<b>RESUMEN .....</b>	<b>2</b>
<b>PALABRAS CLAVE.....</b>	<b>2</b>
<b>ABSTRACT .....</b>	<b>3</b>
<b>KEY WORDS.....</b>	<b>3</b>
<b>1. INTRODUCCIÓN .....</b>	<b>6</b>
1.1. OBJETIVOS.....	6
1.2. REVISIÓN BIBLIOGRÁFICA .....	8
1.3. APLICACIÓN.....	9
1.4. FASES DE TRABAJO .....	12
<b>2. DISEÑO .....</b>	<b>14</b>
2.1. POSICIONAMIENTO .....	14
2.1.1. Movimiento .....	14
2.1.2. Localización.....	16
2.1.3. Posicionamiento.....	19
2.2. MODOS DE DISEÑO.....	22
2.3. ARQUITECTURA.....	23
2.3.1. Plataforma móvil.....	23
2.3.2. Soporte de micrófono .....	30
2.3.3. Motores .....	32
2.3.4. Arduino .....	41
2.3.5. Controlador de motores.....	47
2.3.6. Alimentación.....	48
2.3.7. Encoders .....	49

2.3.8. Comunicaciones.....	51
<b>3. IMPLEMENTACIÓN .....</b>	<b>56</b>
3.1. CONEXIONES .....	56
3.2. ODOMETRÍA .....	63
3.3. INTERRUPCIONES.....	72
3.4. PROTOCOLOS .....	76
3.5. PROGRAMACIÓN .....	81
<b>4. PRUEBAS REALIZADAS.....</b>	<b>95</b>
4.1. PRUEBAS DE CÓDIGO .....	95
4.2. PRUEBAS DE PRECISIÓN.....	97
<b>5. PRESUPUESTO.....</b>	<b>102</b>
<b>6. LÍNEAS FUTURAS DE INVESTIGACIÓN .....</b>	<b>103</b>
<b>7. CONCLUSIONES .....</b>	<b>105</b>
<b>8. REFERENCIAS .....</b>	<b>107</b>
<b>9. ANEXO I .....</b>	<b>108</b>
9.1. CÓDIGO .....	108
<b>10. ANEXO II .....</b>	<b>124</b>
10.1. MANUAL DE USUARIO PARA MAC OS X .....	124

# 1. INTRODUCCIÓN

## 1.1. OBJETIVOS

El objetivo es el diseño de un robot (*VladBot*) de bajo coste que posea la capacidad de posicionar un micrófono de medida en tres dimensiones. La aplicación de este robot puede ser útil para programar medidas acústicas dentro de una sala. Aportaría la utilidad de llevar a cabo mediciones acústicas en una habitación sin la necesidad de estar entrando y saliendo constantemente en la sala sobre la que se está trabajando, ahorrando tiempo al interesado.

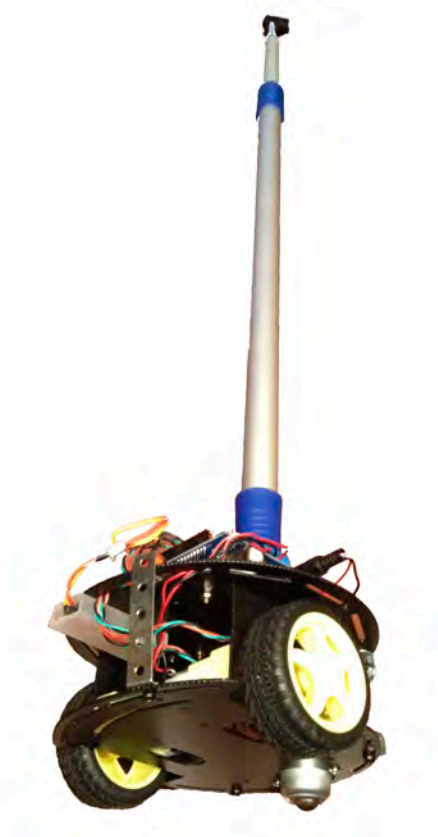


Figura 1: *VladBot*

Atendiendo a estas necesidades, el robot, debe ser preciso. Esta precisión depende en gran parte del sistema de posicionamiento y lo exacto que éste sea. Debido a que el área de trabajo de *VladBot* es la acústica, la precisión de éste debe satisfacer la exactitud que este tipo de medidas demande. Por otro lado, no debe interferir en las medidas acústicas. Esto es, *VladBot* no debe emitir ningún sonido mientras se está realizando la medida. Este requisito es de vital importancia ya que si no el propio sistema puede interferir en las medias acústicas. Algo completamente indeseable.

Un requisito importante, el cual se ha introducido en el primer párrafo, es el bajo coste. Esto se consigue creando un diseño acorde con una plataforma electrónica abierta. Estas plataformas ofrecen flexibilidad y facilidad de uso, tanto de hardware como de software, a un precio razonable. Además permiten un fácil acceso a contenidos y materiales.

*VladBot* es un prototipo el cual ha sido diseñado para evaluar la posibilidad de implementar un robot de las mismas características, pero empleando sistemas más precisos. El desarrollo de este robot pretende comprobar si la idea de automatizar medidas acústicas se puede implementar en una plataforma móvil robotizada. No es objetivo de *VladBot* realizar las medidas acústicas, si no de transportar un micrófono de medida. Siendo así el objetivo del proyecto implementar un sistema de transporte programable y preciso.

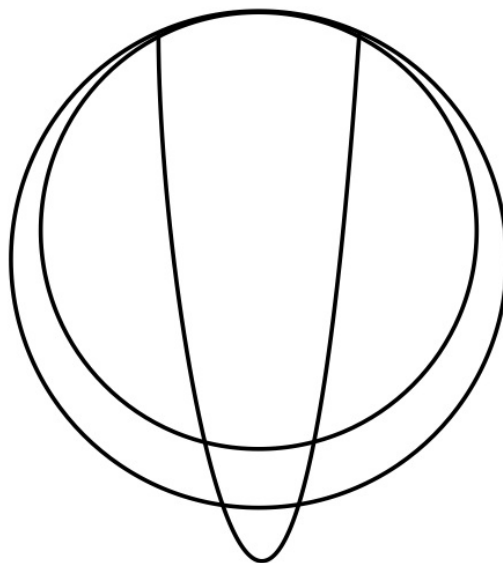


Figura 2: Logotipo *VladBot*

## 1.2. REVISIÓN BIBLIOGRÁFICA

Dado que *VladBot* es el prototipo de un sistema sin precedentes, no existe documentación precisa sobre el tema. Pero como se ha introducido en el capítulo anterior, *VladBot* es un sistema de transporte. Y en el área de la robótica existe una amplia documentación sobre plataformas móviles.

Como se cita en [1], en el mercado se pueden encontrar plataformas móviles de diversos tipos. Estructuras con varias ruedas móviles: tres, ERA-MOBI (Videre Desing); cuatro, RP6 (The Robot Shop); seis, MMP-8 (The Machine Lab).



Figura 3: Plataforma comercial MMP-8 [1]

Conociendo las estructuras sobre las que se ha podido construir a *VladBot*, inmediatamente después ha sido necesario el estudio de los métodos y sistemas posicionamiento. Este ha sido el principal problema y lo que ha supuesto la mayor inversión de tiempo. Ha sido necesario el estudio de las principales tecnologías, es decir, los dispositivos con los que poder implementar un sistema de posicionamiento: *encoders*, giroscopios, compases, balizas, ultrasonidos, etc [2].





Figura 4: Sensor de ultrasonidos

Una vez conocido los sistemas y las técnicas de posicionamiento, el siguiente paso es su implementación. Para ello ha sido necesario estudiar los fundamentos de programación con los que poder realizar esta integración [3]. Concretamente, en [4], se puede encontrar todo lo necesario para la programación del micro-controlador con el que ha sido implementado *VladBot*.

### 1.3. APLICACIÓN

En este apartado se pretende hacer una introducción a *VladBot*, el robot diseñado para satisfacer los objetivos de este proyecto. *VladBot* es una plataforma móvil robotizada controlada inalámbricamente desde un ordenador. Posee un soporte de altura regulable para sujetar un micrófono de medida.

La implementación de este robot surge de la necesidad de programar medidas acústicas dentro de una sala (*indoor*). En el campo de la acústica, existen numerosos ensayos en los que son necesarias repetitivas medidas dentro de una sala, como puede ser una cámara reverberante, una cámara anecoica o una salón de actos. Y en la mayoría de los casos, el sistema que procesa las medidas (PC, sistemas de filtros, procesadores, etc) se encuentran en la sala contigua (ver figura 5). *VladBot*, proporciona al usuario un ahorro considerable de tiempo, tiempo que

supone al usuario el cambio constante de una sala a otra para posicionar el micrófono de medida. Por esta razón la comunicación inalámbrica entre el robot y el usuario es imprescindible. Es este tipo de comunicación lo que hace posible la implementación del proyecto, ya que de otro forma (estableciendo una comunicación por un cable USB por ejemplo) se privaría a la plataforma de autonomía, restándole una preciada capacidad a *VladBot*.



Figura 5: Cámara anecoica

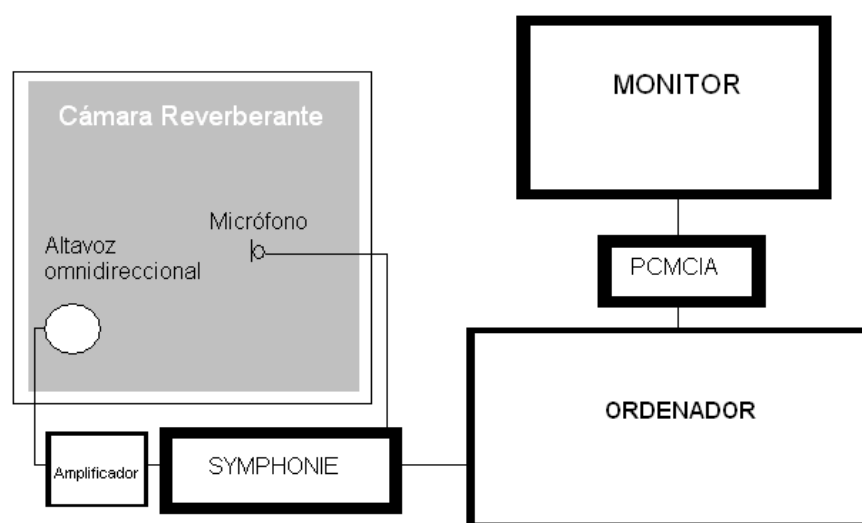


Figura 6: Diagrama de bloques de un ensayo acústico

Esta capacidad de programar sus movimientos, también es útil para seguir un itinerario preestablecido. Un micrófono de medida puede recoger información de la sala mientras el robot se encuentra en movimiento, ya que se trata de una plataforma silenciosa. Esta aplicación puede ser muy apropiada para ensayos en los que se requiera el estudio de la variación del campo sonoro de un recinto con la posición. Un ejemplo de esta práctica puede el estudio de los modos propios de una sala, en los que la posición del micrófono de medida tiene una importancia relevante.

La utilidad estas aplicaciones, y las quiera darle el usuario, siempre deben llevarse a cabo en una superficie plana, sin objetos que puedan obstaculizar sus movimientos, como puede ser grava, tierra, o juntas de baldosas. Hay que recordar que *VladBot* es un prototipo, y en esta primera versión no se ha contemplado su desplazamiento por superficies con relieves u obstáculos. Como se ha mencionado anteriormente, se trata de un proyecto de bajo coste, y dotar a *VladBot* de la capacidad de moverse sobre superficies abruptas requeriría el uso de dispositivos y sistemas de posicionamiento más sofisticados, y en consecuencia, se incrementaría el precio del sistema.

También es interesante citar que el usuario debe encontrarse “cerca” de *VladBot*. Esto es debido a las limitaciones que impone el posicionamiento *indoor*. Tecnologías como GPS (*Global Positioning System*) permitirían el control del robot a larga distancia, pero no es una herramienta que interese. Más adelante se concretará el término “cerca”, pero para una primera aproximación he introducir al lector, “cerca” puede tratarse de una sala contigua o cercana a la que está bajo estudio.

*VladBot* ha sido implementado con una plataforma electrónica abierta, Arduino. Esta plataforma ofrece una gran flexibilidad al usuario. El objetivo principal del uso de este tipo de tecnología es el bajo coste, pero advierte un abanico de posibilidades para el usuario: fácil manejo, posibilidad de recambio de piezas, mejoras adicionales, software gratuito, etc. Esto hace que la plataforma móvil no tenga limitaciones, sólo las que imponga el usuario.

Así, con el objeto de cerrar este apartado, se puede resumir que las aplicaciones para las que ha sido diseñado se encuentran dentro del campo de la acústica. Siendo los rasgos más interesantes de *VladBot*:

- Robot móvil para posicionamiento de un micrófono de medida en 3D.
- Inalámbrico.
- Plataforma electrónica abierta.
- Bajo coste.

#### 1.4. FASES DE TRABAJO

El trabajo realizado se ha llevado a cabo en 3 fases:

##### 1. Diseño

Se han tenido en cuenta 4 características indispensables:

- Movilidad: una plataforma móvil que otorga al robot su capacidad de desplazamiento por la sala.
- Soporte de micrófono: la plataforma posee una base donde se coloca un pie extensible para el micrófono de medida. Este soporte regulable es el responsable del posicionamiento del micrófono en la coordenada Z (atendiendo a un sistema de coordenadas cartesiano,  $[X, Y, Z]$ ).
- Comunicación: capacidad de comunicación inalámbrica con el usuario.
- Bajo coste: utilización de tecnologías abiertas que no requieran un gran desembolso de capital.

##### 2. Implementación

La integración de las características del diseño se ha llevado a cabo mediante la plataforma Arduino. Arduino ofrece un IDE de programación de fácil manejo, escrito en Java y de código abierto. Además se encuentra disponible en los sistemas operativos más comunes: Windows, Mac OS X y Linux.

Para la comunicación inalámbrica entre el robot y el usuario, se ha realizado un encapsulamiento de mensajes haciendo uso de un protocolo de comunicación.

### 3. Pruebas

Las pruebas realizadas se pueden agrupar en 2 grandes bloques:

- Posicionamiento: se ha estudiado las limitaciones que imponen las diferentes tecnologías disponibles (de bajo coste) para el posicionamiento de la plataforma. La precisión es uno de los principales objetivos, por tanto es necesario conocer el error que introduce el sistema de posicionamiento.
- Comunicación: otro punto bastante importante que hasta ahora no ha sido citado, es dotar al robot de una comunicación fiable con el usuario. Por esto también han sido necesarias diversas pruebas con el protocolo de comunicación que garanticen la correcta comunicación entre VladBot-persona.

## 2. DISEÑO

### 2.1. POSICIONAMIENTO

Uno de los problemas principales que ha planteado el robot móvil ha sido la navegación, que se puede definir como: dado un punto de partida A, alcanzar el punto de destino B (o varios puntos de destino). Este problema ha conllevado solucionar otros a más bajo nivel:

- Percepción: debe obtener correctamente la información que le aportan sus sensores.
- Localización: debe ser capaz de situarse en su entorno.
- Planificación: debe actuar consecuentemente para alcanzar su meta.
- Control de movimiento: debe ejecutar las acciones pertinentes en sus actuadores para conseguir el objetivo deseado.

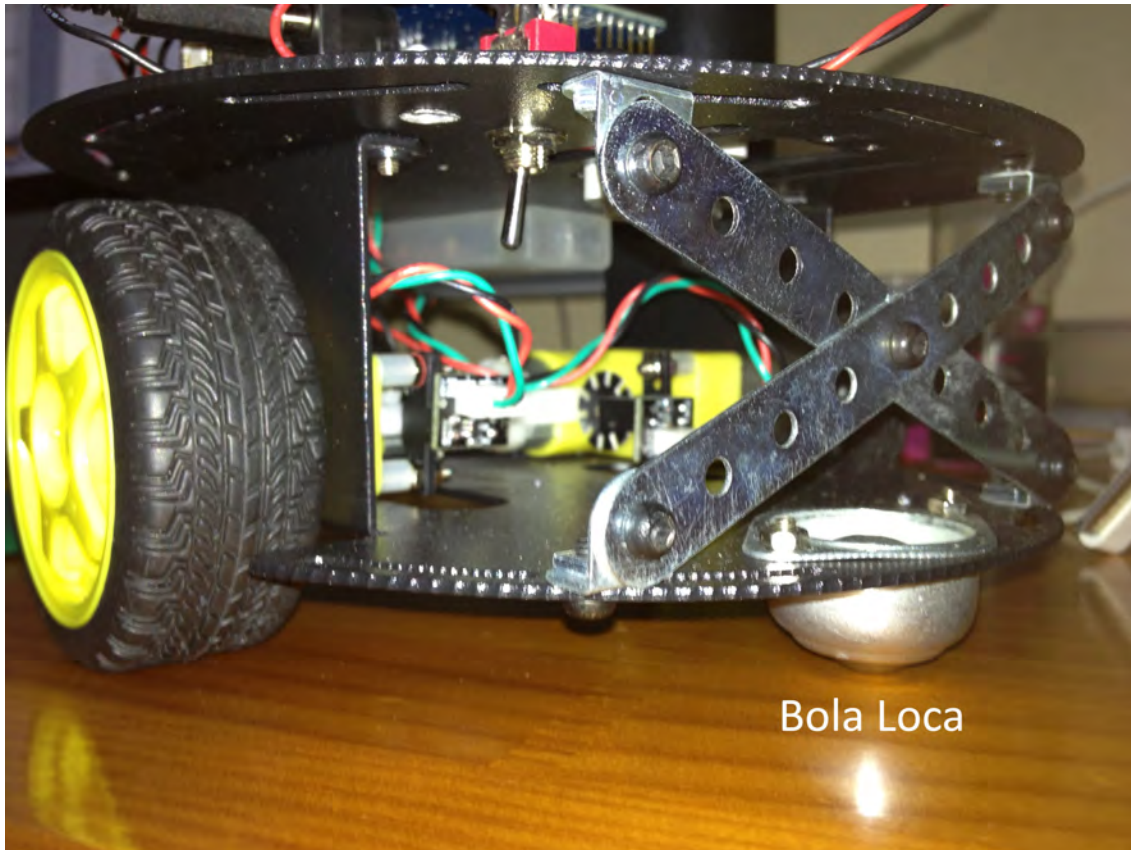
Por tanto, se ha englobado el diseño del posicionamiento en 3 niveles: movimiento o locomoción, localización y posicionamiento o trayectoria.

#### 2.1.1. MOVIMIENTO

El diseño del movimiento se ha llevado a cabo en 2 partes bien diferenciadas, el movimiento en el plano  $[X,Y]$  y el movimiento del micrófono  $[Z]$ . Esta discriminación se ha realizado porque resulta más sencillo abarcar los dos movimientos por separado, ya que la localización se ha enfocado en la plataforma móvil en una superficie plana, y no en localizar el micrófono en el espacio.

El movimiento sobre el plano lo realiza una estructura sobre 2 ruedas motrices y una bola loca (que facilita las maniobras de giro). Esta tercera rueda, o bola loca, es

la responsable de garantizar la maniobrabilidad del robot. Esta estructura es capaz de transportar un soporte de micrófono y un micrófono, además de transportar también la placa, y la fuente de alimentación. También otorga la posibilidad de alojar cualquier tipo de sensor que pudiese ser necesario. El movimiento del micrófono sobre el eje Z se ha conseguido con un soporte extensible.



**Figura 7: Bola loca**

Gracias a este diseño, sólo han sido necesarios 2 motores eléctricos con 2 ruedas de goma, disminuyendo el consumo, y facilitando maniobras de giro estáticas. Así, el *ValdBot* es capaz de moverse hacia delante y hacia atrás, y realizar giros sobre sí mismo para orientarse en una dirección deseada.

### 2.1.2. LOCALIZACIÓN

Los métodos de localización que existen son variados. A continuación se exponen todos los sistemas que se han barajado para el diseño de *VladBot*:

- GPS: este método ha sido descartado debido a que el sistema está pensado para ser utilizado en una sala (*indoor*), y GPS no funciona en interiores, si no que se trata de un sistema de posicionamiento para exteriores (triangulación de balizas, satélites). Además la precisión de este sistema es de unos pocos metros, una precisión que no es suficiente para el robot.



Figura 8: GPS

- Mapas: aparte del gran coste computacional, se ha decidido que el robot va a establecer su posición respecto a una posición de inicio preestablecida y no a un mapa. Este sistema de posicionamiento además requiere el uso de *software* más sofisticado y aumentaría su coste considerablemente.





Figura 9: Mapa para posicionamiento de un robot

- Odometría: este es un método sencillo para estimar la posición y se basa en la medición del giro de ruedas mediante unos sensores. El problema que presenta es que a medida que la trayectoria es mayor, el error aumenta, es decir, el error de estimación de la posición no está acotado.

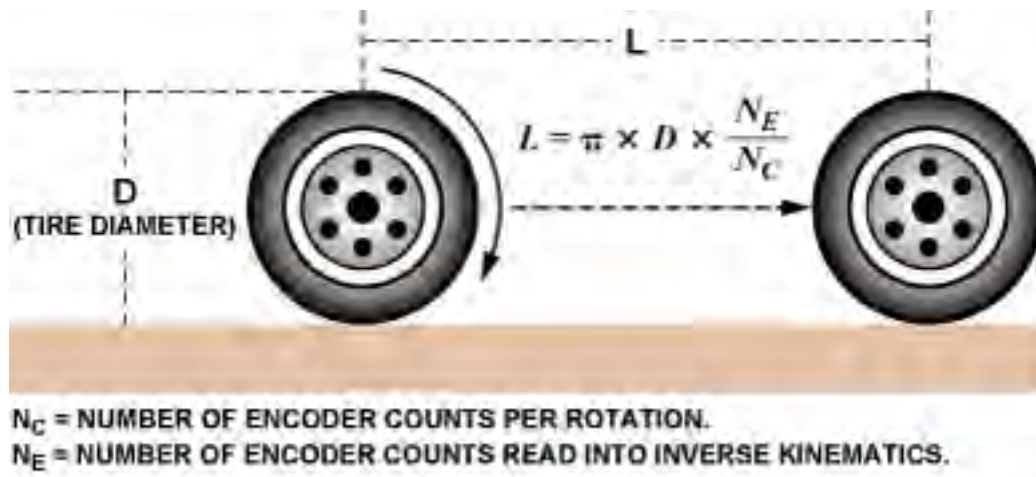


Figura 10: Odometría

- Rutas: raíles, marcas ópticas en el suelo, etc. No es menester que el micrófono recoja siempre la misma información, por lo que este método no es recomendable para las características que rigen su diseño.

- Balizas: visuales o de tiempo de vuelo: el robot emite una señal y espera la respuesta de las balizas. Una opción muy interesante, ya que no es muy cara y sencilla de implementar. Pero que requiere de una preinstalación de unas balizas fijas en la sala, restando flexibilidad a su funcionalidad.

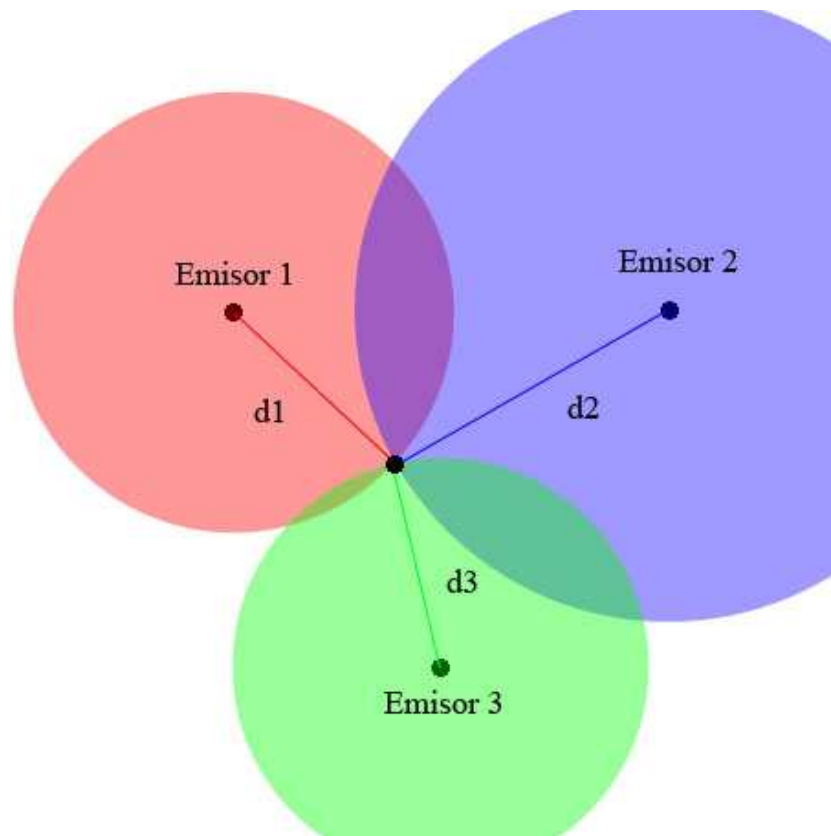


Figura 11: Localización mediante balizas [5]

- *Landmarks*: estima la posición por triangulación de una marca (artificial o natural), cuya posición es conocida. Al igual que un sistema de balizas, requiere una instalación, en este caso de las *landmarks*.

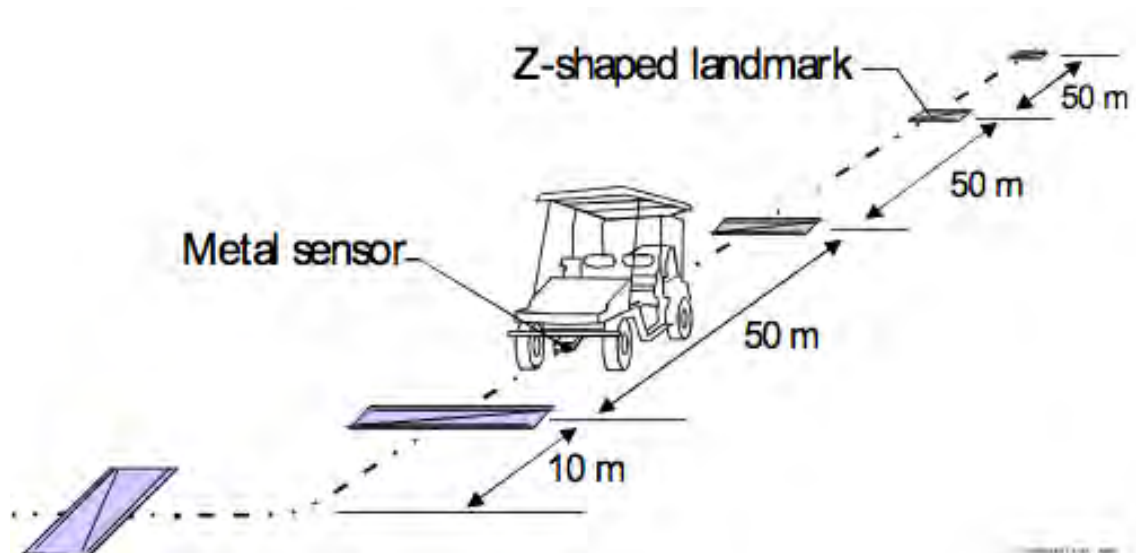


Figura 12: Navegación con *landmarks* [2]

Cotejadas las diversas opciones que por las que se han podido optar, y descartando algunas tecnologías por imposibilidad de implementación (como el GPS o las rutas) la que más ha favorecido al diseño ha sido la odometría. Se trata de la solución más sencilla de implementar con dispositivos de bajo coste. Razón suficiente para ser la tecnología escogida para posicionar el robot. Además aporta versatilidad a *VladBot*, ya que puede con este sistema de posicionamiento operar en cualquier sala. Con otro tipo de técnicas habría sido necesario la preinstalación de los dispositivos necesarios para la localización en la habitación de trabajo. Por tanto, asumiendo el determinado error que genera este sistema, se puede determinar con suficiente precisión la posición del robot.

### 2.1.3. POSICIONAMIENTO

El posicionamiento del robot, o trayectoria, la realiza a velocidad constante. Esta velocidad puede ser cambiada a gusto del usuario, no siendo un factor determinante en el movimiento de *VladBot*. No obstante, es recomendable que esta velocidad, tanto de avance como de giro, no sea muy elevada, ya que podría introducir un ruido no deseado en la medidas.

El movimiento sobre el plano, consiste en determinar la distancia a una posición de inicio conocida. Esta distancia es determinada gracias en unos *encoders* colocados en las 2 ruedas motrices de la plataforma. La misión de estos *encoders* es contar las

vueltas que dan las ruedas, para posteriormente y con las herramientas adecuadas, determinar la distancia a este punto.

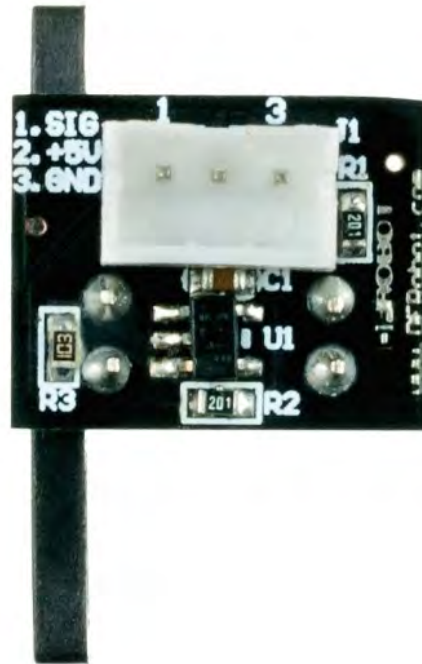


Figura 13: Encoder

El avance del robot, puede ser tanto hacia delante como hacia atrás. El usuario introduce la distancia en centímetros que desea que avance. Cuando *VladBot* ha terminado de posicionarse, manda una señal indicando que ha terminado su movimiento. Esto resulta una ventaja cuando se quieren programar distintos puntos de medida. El robot puede alcanzar cada uno de estos puntos y esperar a que se realice la medida oportuna para, a continuación, realizar su siguiente movimiento.

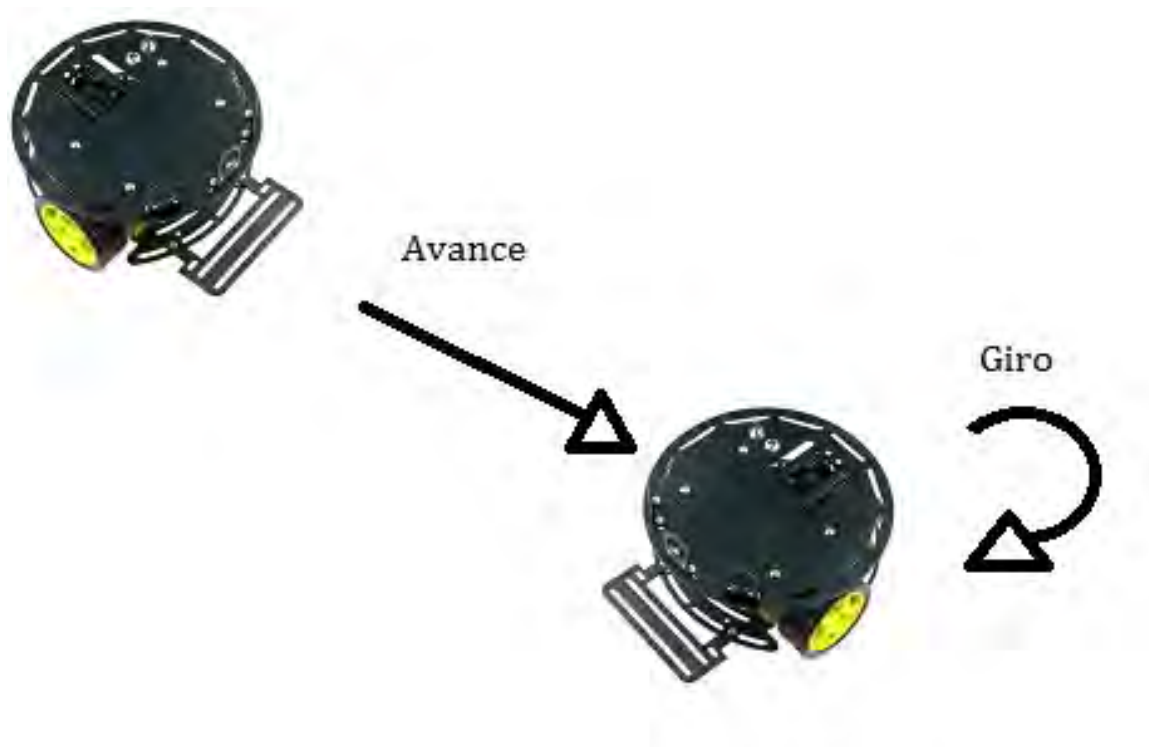


Figura 14: Movimientos en el plano

De igual modo se pueden programar sus giros. Estos giros orientan al robot en una determinada posición, definida por el usuario. Estos giros los realiza sobre sí mismo, gracias a la bola loca que favorece este tipo de movimiento. Este movimiento ha sido diseñado así para que la orientación del robot en una dirección sea más eficaz (odometría) y más intuitiva para el usuario.

Por otro lado, el posicionamiento del micrófono en el eje Z, se ha diseñado de forma que el usuario pueda elegir su posición (entre 90 y 140 cm). La posición del micrófono sobre el eje Z se establece de forma manual. Ha sido diseñado de esta manera para evitar más motores en la plataforma que la hiciesen más pesada. Esto aumentaría el consumo y demandaría otra batería. Además, debido a que en el campo de la acústica interesa sobre todo medir sobre el mismo plano, no ha sido considerado de vital importancia la motorización de este movimiento.

## 2.2. MODOS DE DISEÑO

El sistema de control del movimiento de *VladBot*, ha sido diseñado en lazo abierto. Existen dos modos de diseño:

- Lazo abierto: en estos sistemas no existe retroalimentación, o en otras palabras, una señal de salida no se convierte en la nueva señal de entrada al sistema. Ajustando estas señales a nuestro sistema, encontramos que la señal de salida que ofrece el robot es su posición actual respecto a la de referencia. Esto es, que la posición de la que está informando es la posición respecto al último movimiento. En este sistema, el usuario es el que realiza el ajuste para el control de la posición.
- Lazo cerrado: en estos sistemas sí existe retroalimentación. La señal de salida se convierte en la nueva entrada al sistema. Utilizando esta filosofía, el robot es capaz de ajustar su posición respecto a una inicial conocida.

Como se ha introducido en el primer párrafo de este apartado, el diseño elegido ha sido en lazo abierto. Se ha escogido esta opción por su sencillez (menos carga computacional) y por su bajo precio, ya que si se hubiese optado por realizar un diseño en lazo cerrado, hubiese sido necesario incorporar algún sistema de posicionamiento adicional, como unas balizas, o integrar un sistema odométrico más preciso, lo cual incrementaría considerablemente su precio.

Así, *ValdBot* tiene la capacidad de posicionarse (siempre con una posición de referencia), pero el ajuste de esta posición es controlada por el usuario. A modo de ejemplo, el robot puede moverse en línea recta con suficiente precisión, pero el trazado de esta línea recta es controlado por el usuario. Es posible detenerse en cualquier punto que se desee, además de poder cambiar de sentido en cualquier momento. Este diseño, aparte de ser más sencillo, aporta una gran flexibilidad, que con un diseño en lazo cerrado no hubiera sido posible.

## 2.3. ARQUITECTURA

En este apartado se presentan los componentes de los que está formado *VladBot*, haciendo un recorrido más extenso por sus características y funcionalidades. Se hace un recorrido comenzando por los dispositivos más sencillos (plataforma móvil, soporte del micrófono) hasta los más sofisticados (comunicaciones inalámbricas). Este recorrido sigue un orden cronológico, o sea, el orden en el que se han ido agregando los diferentes elementos del sistema.

### 2.3.1. PLATAFORMA MÓVIL

Existen numerosas plataformas móviles comerciales, cada una con diferentes propiedades. Estas plataformas se fabrican de diversos materiales: madera, aluminio, plástico, etc. Pero el estudio se ha centrado en las plataformas de aluminio, ya que son las más resistentes y ligeras. Por otro lado, debido a que la plataforma carga con un micrófono, requiere estabilidad, y con materiales más ligeros esto no se consigue.

- Plataforma móvil DF Robot 4WD

La plataforma compatible con Arduino DFRobot 4WD es ampliamente utilizada en el mundo de las plataformas autónomas. Su tracción permite salvar obstáculos y pendientes con un fuerte chasis de aluminio. Las partes acrílicas tienen grosor extra para eliminar posibles fragilidades por fallos en el material. La potencia de los motores permite unos movimientos ágiles y rápidos, estando especialmente adaptados para hierba, gravilla, arena y pavimento con pendiente.

Es muy apta para competiciones de robots y otros proyectos de desarrollo. Sus agujeros de montaje son compatibles con una amplia variedad de sensores. En su interior hay espacio suficiente para instalar la batería. Está pensado para facilitar futuras expansiones como cámaras o brazos robóticos [6].

## Especificaciones

- Voltaje: de 4.5V a 6V
- Motor tipo: 130
- Velocidad de giro: 10000rpm
- Relación reductora: 1:120
- Velocidad máxima: 68 cm/s
- Diámetro de rueda: 65mm
- Tamaño: largo 230mm ancho 185mm alto 110mm
- Peso: 614 gramos (sin batería)
- Carga máxima: 800g



Figura 15: Plataforma DF Robot 4WD



- Plataforma móvil DF Robot 2WD

Esta plataforma móvil es pequeña, de bajo coste e ideal para su uso con un microcontrolador Arduino estándar. El kit incluye dos motores de tracción, ruedas (y la bola loca trasera), bastidor y todo el hardware de montaje. El cuerpo es de aleación de aluminio, muy resistente. Las ruedas de goma flexible hace que sea adecuado para suelos de interiores.

### **Especificaciones**

- Incluye 2 motores DC (mismos motores que la plataforma DF Robot 4WD)
- Bola loca
- Chasis con hardware de montaje
- Dimensiones (diámetro de la base): 170mm
- Peso: 400g



**Figura 16: Plataforma móvil DF Robot 2WD**

- Otras plataformas



Figura 17: Pololu Round Robot



Figura 18: Plataforma móvil Nexus Robot 2WD



Figura 19: Plataforma móvil Nexus Robot 3WD



Figura 20: Odyssey 6WD

La plataforma con la que finalmente se ha diseñado a *VladBot* ha sido la plataforma móvil de 2 ruedas de DF Robot (figura 16), por su precio y su funcionalidad. Su capacidad de poder realizar giros en estático, gracias a la bola loca, para orientarse es altamente útil. Posee una amplia plataforma donde colocar el microcontrolador, la batería y el soporte para el micrófono. Además de su robustez y su ligero peso, posee unas perforaciones en las que poder incrustar todos los componentes necesarios para moverla, así como cualquier tipo de sensor.

Esta plataforma ha sido ligeramente modificada. Se han añadido unos refuerzos de aluminio en su parte delantera para evitar oscilaciones al colocar el soporte del micrófono y el micrófono. Estos refuerzos la hacen más estable.

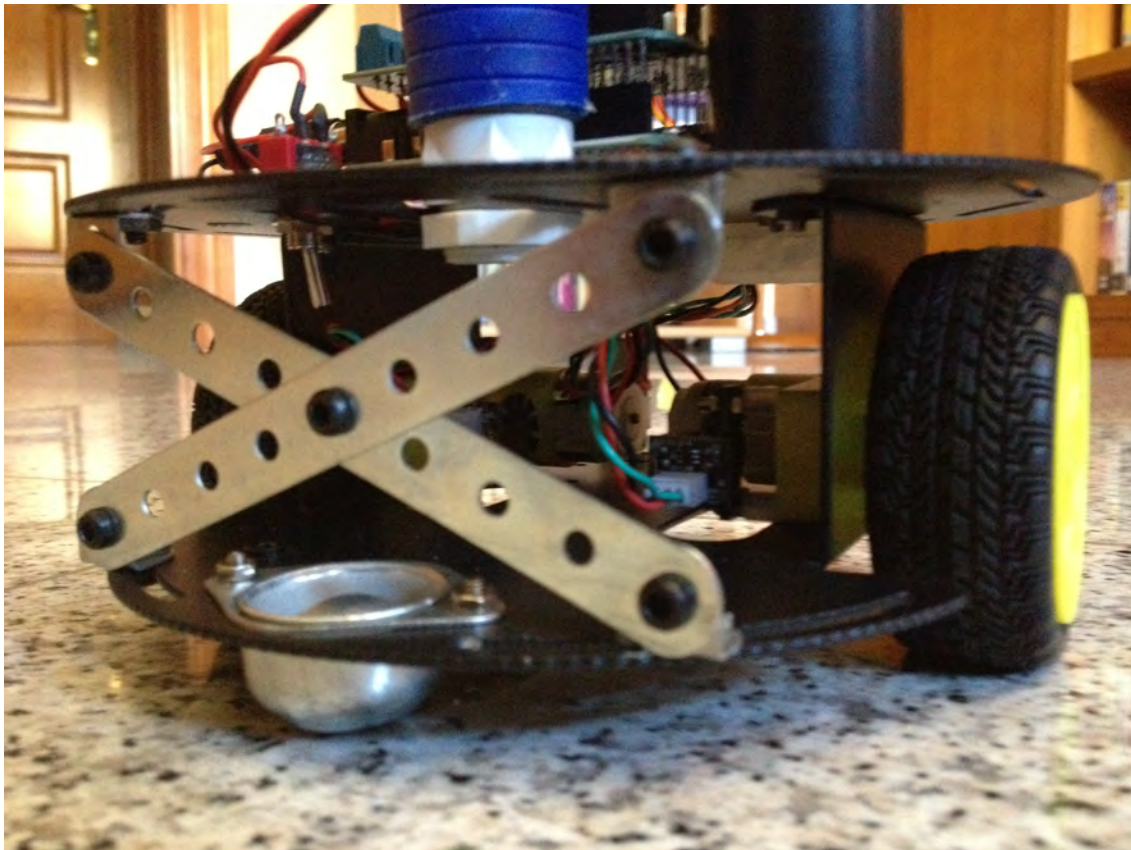


Figura 21: Refuerzo del chasis

- Otras arquitecturas

Existen otro tipo de arquitecturas, como los robots hexápodos o los sumo. Pero no se ajustan a los objetivos que ha de cumplir *VladBot*. Los hexápodos poseen un movimiento complejo y poco estable. Con este tipo de diseño sería prácticamente imposible mantener el micrófono moviéndose sobre una línea recta. Los robots sumo, aparte de ser más pesados, consumen más energía al llevar más ruedas motrices. Posee un movimiento más complejo, y existen arquitecturas con la misma funcionalidad a menor precio y más sencillas.



Figura 22: Plataforma de 6 brazos Lynxmotion



Figura 23: Plataforma sumo Zumo Robot



### 2.3.2. SOPORTE DE MICRÓFONO

La estructura que soporta el micrófono de medida consiste en un cilindro extensible de aluminio, con un brazo embutido que permite la sujeción del micrófono. Se ha anclado a la plataforma con una rosca de plástico, lo que permite se desmontaje para un cómodo traslado del robot.



**Figura 24: Brazo para la fijación del micrófono a el soporte**



Figura 25: Soporte extensible

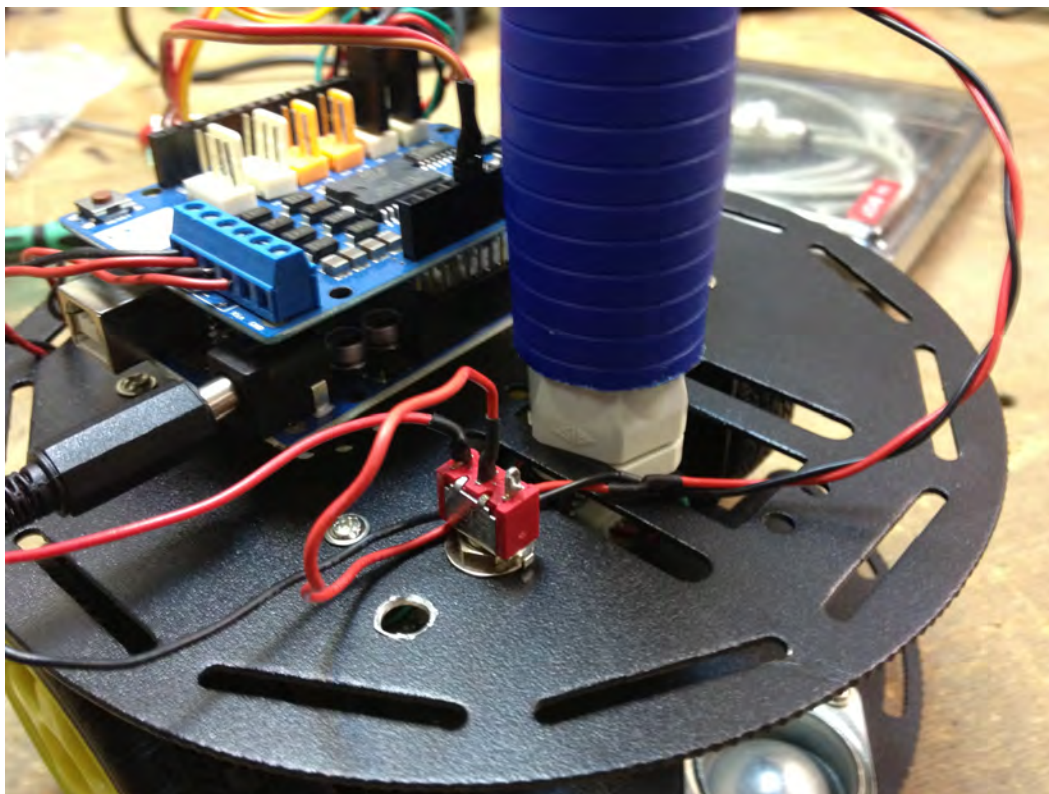


Figura 26: Pieza de sujeción a la plataforma

Aporta la capacidad de posicionar el micrófono en el eje Z, además de poder ser colocado con una orientación de 360 grados. Posee un peso muy ligero, ideal para que la plataforma móvil pueda cargar con él, siendo más relevante el peso del micrófono que el del propio soporte.

Su mecanismo de funcionamiento es muy sencillo. El giro de la rosca hace que la extensión más fina del cilindro se mueva libremente por la sección más gruesa. El nuevo giro del cilindro más fino ajusta una rosca interna que fija el soporte en la posición deseada.

### 2.3.3. MOTORES

En el campo de la robótica se pueden encontrar, fundamentalmente, 2 tipos de motores: motores de corriente continua (DC, *Direct Current*), y motores paso a paso (*Stepper Motors*).



- Motores DC

Su mecanismo de funcionamiento consiste en aplicar una tensión de alimentación entre sus bornes, de manera que se haga girar su eje. Si se desea invertir el sentido del giro de su eje basta con invertir su alimentación, y el motor comenzará a girar en sentido opuesto. Estos motores giran a una determinada velocidad, no siendo posible enclavarlos en una posición específica. Esta es la principal diferencia con los motores paso a paso, los cuales si pueden fijarse en una posición deseada.

Un motor de corriente continua esta compuesto por:

- Rotor: es la parte que proporciona movimiento a la carga. En la figura 13 se puede observar las partes que lo constituyen.

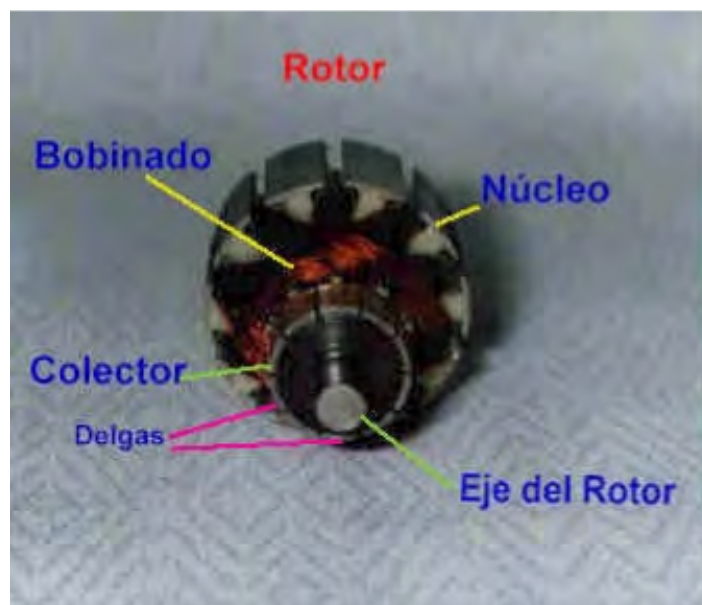


Figura 27: Rotor de un motor DC

- Estator: este elemento no se mueve, y es el encargado de suministrar el flujo magnético necesario al bobinado para realizar el movimiento giratorio.



Figura 28: Estator de un motor DC

- Motores paso a paso

Estos motores poseen la característica de que su movimiento no es continuo, si no que gira en ángulos discretos, es decir, por cada pulso que se le aplique, giran un determinado ángulo. Poseen la capacidad de quedarse enclavados en una posición determinada, por lo que son ideales para mecanismos donde se requiera una elevada precisión.

Su funcionamiento se debe a un rotor sobre el que van aplicados distintos imanes permanentes, y por un cierto número de bobinas excitadoras bobinadas en su estator. Las bobinas son parte del estator y el rotor es un imán permanente. Toda la conmutación (o excitación de las bobinas) es manejada por un controlador.

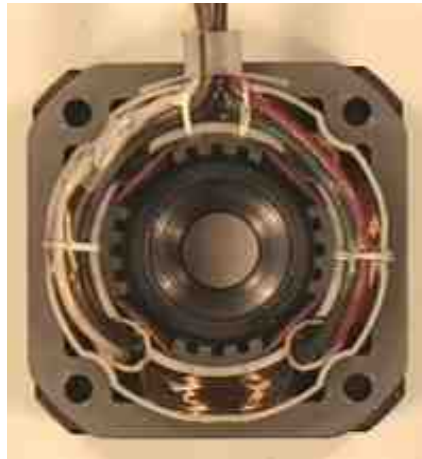


Figura 29: Estator de 4 bobinas

Existen 2 tipos de motores paso a paso de imán permanente, bipolares y unipolares:

- Bipolares: tienen generalmente cuatro cables de salida. Requieren el cambio de la dirección del flujo de corriente a través de las bobinas en la secuencia apropiada para realizar un movimiento. Cada inversión de la polaridad provoca el movimiento del eje en un paso, cuyo sentido de giro está determinado por la secuencia seguida.

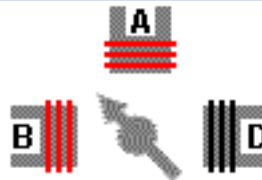
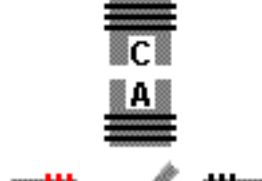
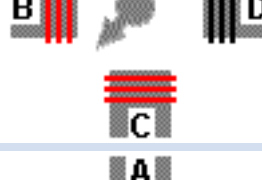

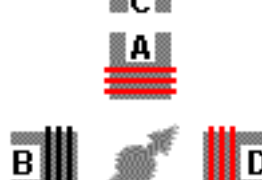
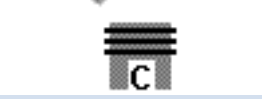



Tabla 1: Secuencia de manejo de motores paso a paso bipolares

PASO	TERMINALES			
	A	B	C	D
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

- Unipolares: Estos motores suelen tener 6 o 5 cables de salida, dependiendo de su conexionado interno. Utilizan un cable común a la fuente de alimentación y posteriormente se colocan las otras líneas a tierra en un orden específico para generar cada paso. Son más sencillos de controlar. Existen 3 secuencias para controlar este tipo de motores:

1. Secuencia Normal: el motor avanza un paso por vez y debido a que siempre hay al menos dos bobinas activadas, se obtiene un alto torque de paso y de retención.

Tabla 2: Funcionamiento secuencia normal

PASO	Bobina A	Bobina B	Bobina C	Bobina D	
1	ON	ON	OFF	OFF	
					
					
2	OFF	ON	ON	OFF	
					
					
3	OFF	OFF	ON	ON	
					
					
4	ON	OFF	OFF	ON	

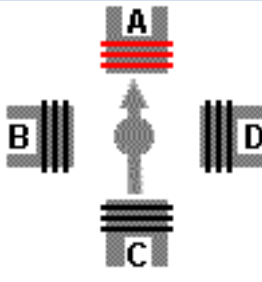
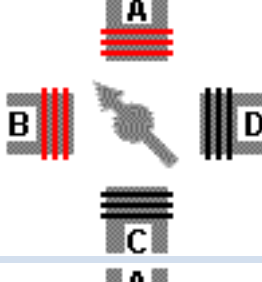
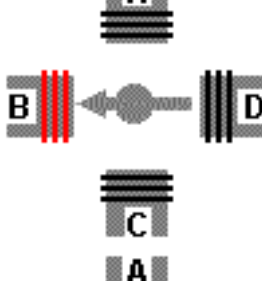
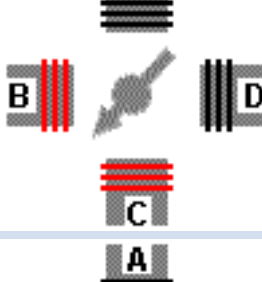
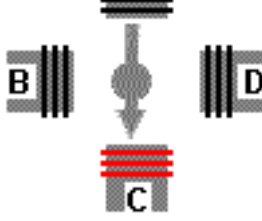
2. Secuencia *Wave Drive*: solo se activa una bobina a la vez, ofreciendo un movimiento más suave. Torque de paso y retención menor.

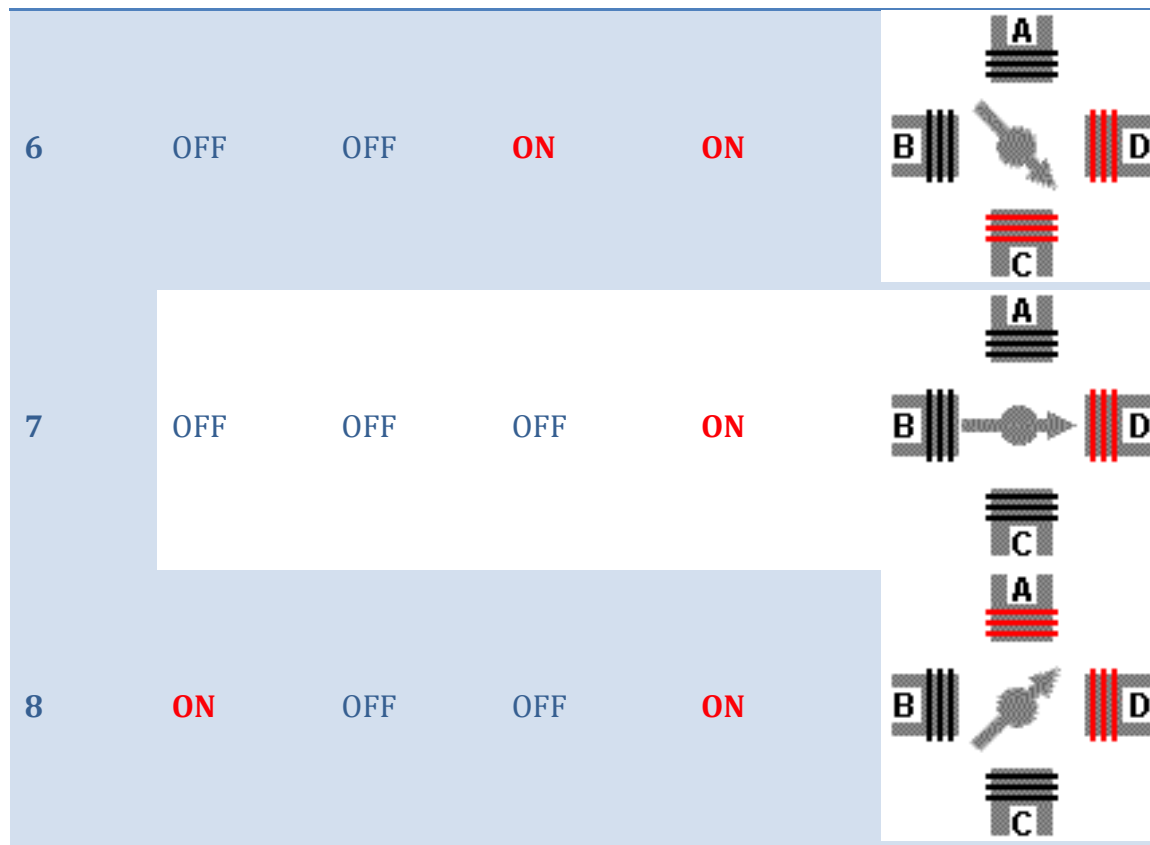
Tabla 3: Funcionamiento secuencia *Wave Drive*

PASO	Bobina A	Bobina B	Bobina C	Bobina D
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON

3. Secuencia Medio Paso: movimiento la mitad del paso real. Se activan 2 bobinas y luego 1, así sucesivamente.

Tabla 4: funcionamiento secuencia de medio paso

PASO	Bobina A	Bobina B	Bobina C	Bobina D	
1	ON	OFF	OFF	OFF	
2	ON	ON	OFF	OFF	
3	OFF	ON	OFF	OFF	
4	OFF	ON	ON	OFF	
5	OFF	OFF	ON	OFF	



Vistos los diferentes tipos de motores, los diferentes modos de funcionamiento, y comparando funcionalidades entre los 2 tipos (de paso y de corriente continua), se ha decidido dotar a VladBot de un par de motores DC (corriente continua). Esta elección se ha fundamentado básicamente en el coste de esta tecnología y su sencilla implementación. Hay que recordar que se trata de un proyecto *low cost*, y por tanto, un menor precio siempre ha sido un factor determinante a la hora de tomar una decisión. Por otro lado, la precisión que se puede conseguir con los motores DC es más que suficiente para el menester que preocupa al robot. Un motor de paso sería más apropiado, por ejemplo, para accionar un brazo robótico, cuya precisión debe ser milimétrica. Además, estos motores se venden en un kit con la plataforma elegida, siendo su integración en ésta perfecta. No hay necesidad de modificaciones ni problemas de dimensiones. Sencillamente encajan a la perfección, ya que ambos, plataforma y motores, han sido diseñados para el mismo propósito.



Figura 30: Motor DC de *VladBot*

Sus especificaciones técnicas son las siguientes:

- Relación reductora: 1:120
- Velocidad sin carga (3V): 100RPM
- Velocidad sin carga (6V): 200RPM
- Corriente sin carga (3V): 60mA
- Corriente sin carga (6V): 71mA
- Corriente de parada (3V): 260mA
- Corriente de parada (6V): 470mA
- Par motor (3V): 1.2 Kgcm
- Par motor (6V): 1.92 Kgcm
- Tamaño: 55mm x 48.3mm x 23mm
- Peso: 45 gramos



#### 2.3.4. ARDUINO

*“Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos [4].”*



Figura 31: Logotipo de Arduino

Un micro-controlador Arduino posee varios pines de entradas de los que toma información. También tiene otros pines de salida con los que actuar con su entorno. A estos micro-controladores se les pueden añadir sensores, luces, motores, etcétera, que, conectados a estos pines, pueden interactuar con su alrededor, creando un determinado sistema o herramienta. Estos proyectos pueden ejecutarse sin necesidad de estar conectados a un ordenador, aunque existe la posibilidad de hacerlo si el proyecto demanda la comunicación con algún software como puede ser Processing.

Estas placas pueden ser compradas, o hechas a mano, ya que toda la información referente a su implementación es de libre acceso (código abierto). Todo el software también se puede descargar de forma gratuita. Los micro-controladores se programan mediante el lenguaje de programación Arduino (implementado en Wiring) y su entorno de desarrollo. Para más información se puede consultar la página web de los desarrolladores [4] ya que no es objeto un estudio exhaustivo de la plataforma.

Arduino ha sido elegido para la implementación de *VladBot* debido a las numerosas funcionalidades que aportan estas sencillas placas. Aparte de ser la opción más versátil, ya que se le pueden añadir *shields* que aumenten sus capacidades, es afortunadamente de las más económicas. Gracias al IDE gratuito que ofrece la plataforma, y a la “comunidad Arduino”, se puede programar la placa sin unos extensos conocimientos de programación, ya que se trata de un lenguaje sencillo y existen innumerables bibliotecas adaptadas a cualquier necesidad.

La placa escogida para el propósito del robot a sido la Arduino Uno (figura 34). Existen numerosas placas Arduino, pero la Arduino UNO es la que mejor se ha adaptado a las necesidades de *VladBot*. Algunas de estas placas no están diseñadas para añadir *shields* (figura 32), y esto es un gran inconveniente, ya que el robot precisa de un controlador para los motores. Otras en cambio resultan muy potentes (incrementando su precio), cuando el proyecto no precisa procesar gran cantidad de datos (figura 33).

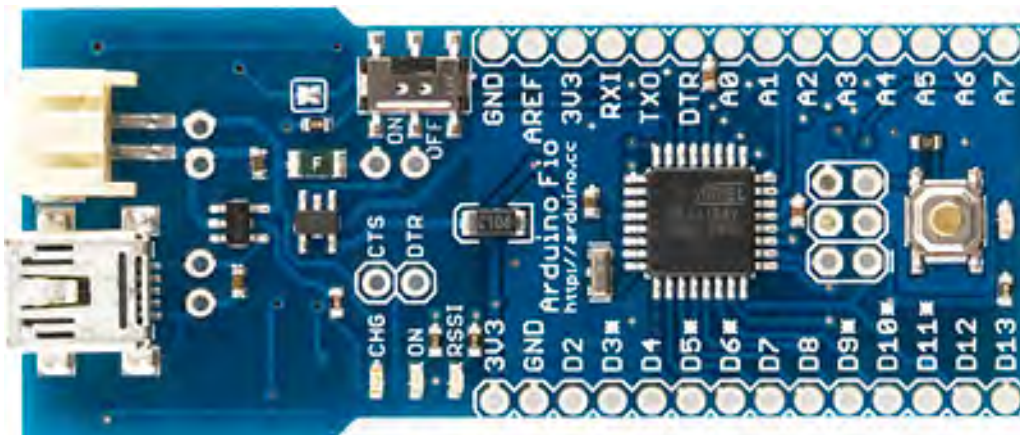


Figura 32: Arduino Fio

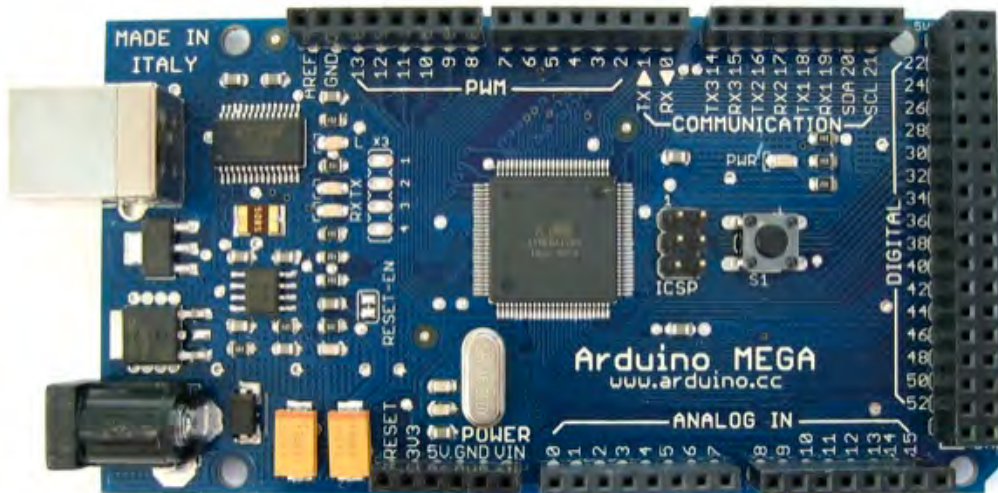


Figura 33: Arduino Mega

- Arduino Uno



Figura 34: Arduino Uno

Esta placa contiene un micro-controlador basado en el ATmega328. Puede ser alimentada por USB o con una alimentación externa (de 7 a 12V recomendados). Posee varios pines de entrada y salida tanto digitales como analógicos que le permiten su comunicación con el exterior (a través de sensores, luces, motores, etc). Además contiene unos pines con una funcionalidad específica:

- Serial: transmisión (TX) y recepción (RX), para la comunicación serie.
- Interrupciones externas: estos pines pueden ser configurados para que hagan la función de un interruptor cuando se produzca una interrupción.
- PWM: *Pulse-width Modulation* (Modulación por Ancho de Pulso).
- SPI: comunicaciones SPI.
- LED: pin para conexión de un LED.

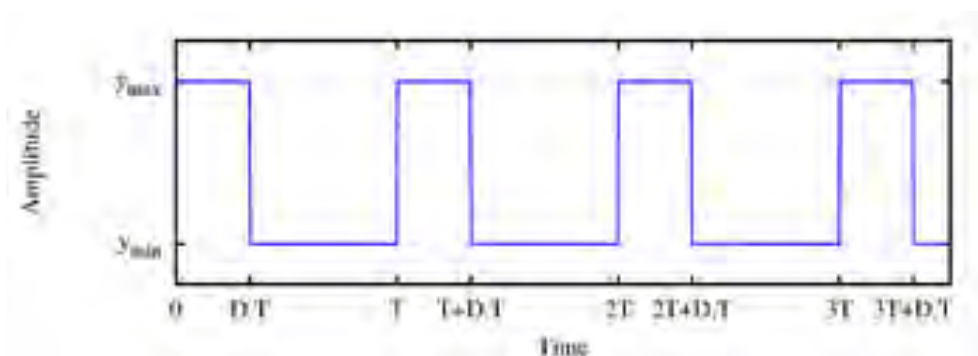


Figura 35: Ciclo PWM

Las especificaciones de la placa Arduino Uno se muestran en la tabla 5:

Tabla 5: especificaciones técnicas Arduino Uno

Microcontrolador	ATmega328
Voltaje de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límites)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Entrada Analógicas Pins	6
CC por I/O Pin	40 mA
CC para 3.3V Pin	50 mA
Memoria Flash	32 KB (ATmega328) of which 0.5 KB used by bootloader

SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad de Reloj	16 MHz

- IDE Arduino



Figura 36: Logotipo IDE de Arduino

Arduino ofrece una herramienta de programación sencilla y muy potente. Con ella se pueden crear infinidad de programas para que una placa Arduino interactúe con su entorno. Gracias a esta IDE, el usuario de *VladBot* puede reprogramar su código, dándole nueva funcionalidades o mejorando las existente. Esto aporta gran flexibilidad al proyecto.

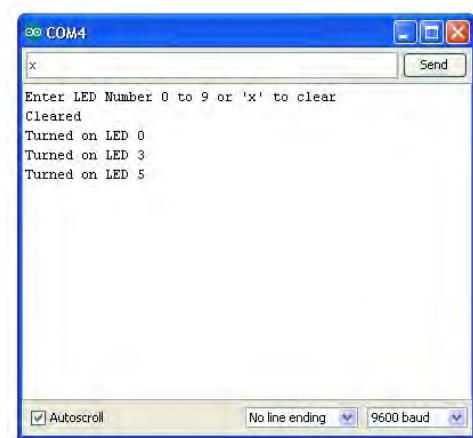


Figura 37: monitor serial IDE Arduino



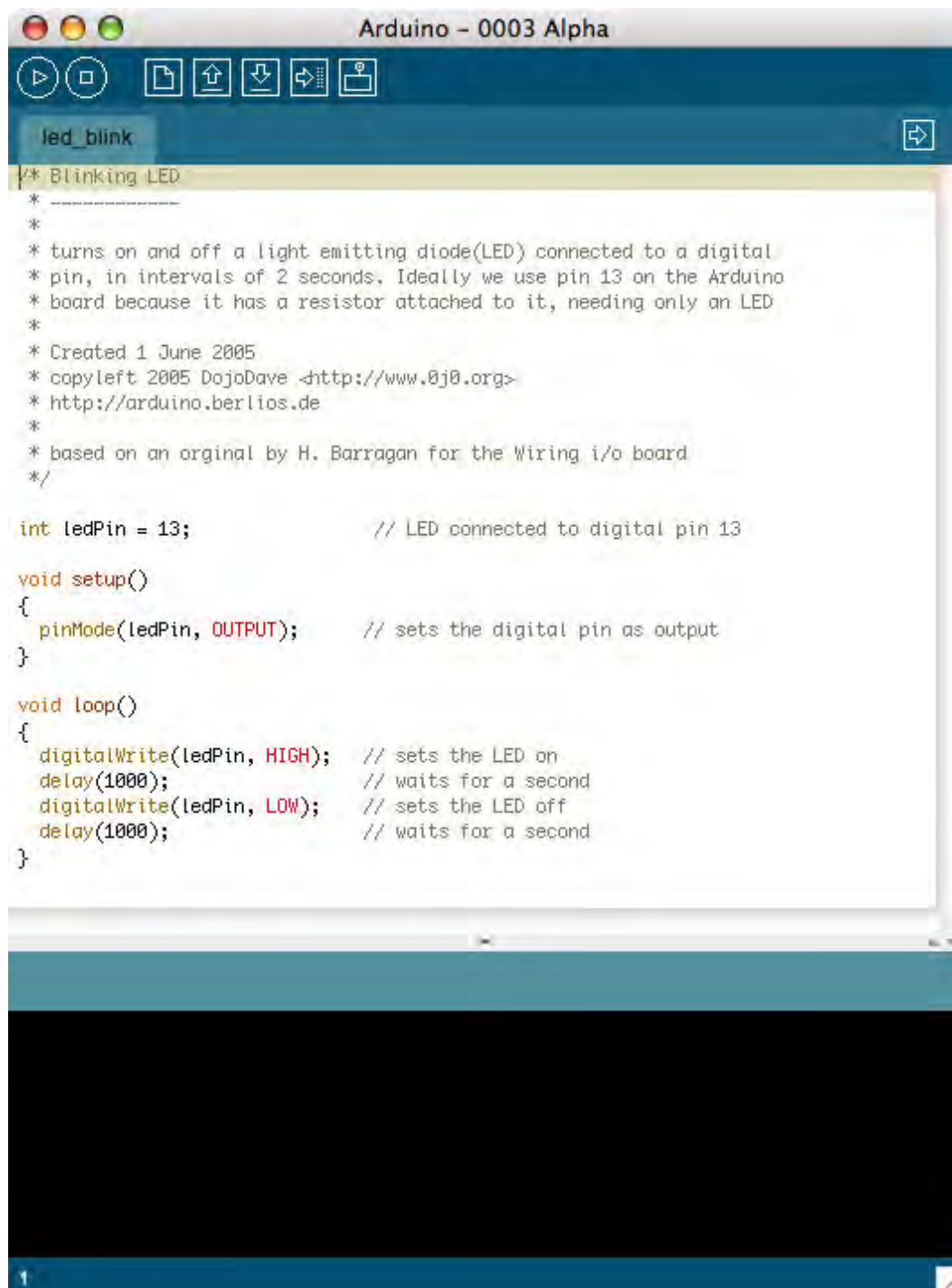


Figura 38: Interfaz IDE Arduino

### 2.3.5. CONTROLADOR DE MOTORES

Los 2 motores DC precisan de un controlador que los accione. La Arduino Motor Shield R3 ha sido la *shield* que mejor se ha adaptado a las necesidad del proyecto. Esta placa se ajusta perfectamente a el Arduino Uno, no siendo necesario ningún cableado. Ofrece un control independiente de dos motores de corriente continua o de un servomotor (motor paso a paso). Incorpora un driver de motores L298P en formato SMD, además dispone de cuatro conectores para módulos TinkerKit (necesarios para la integración de los *encoders*) y dos conectores I2C para comunicaciones.

Ha resultado idóneo el control independiente de los 2 motores DC, ya que gracias a ello es posible el cambio de orientación de la plataforma. Haciendo girar las ruedas en sentidos opuestos se consigue que la plataforma gire sobre sí misma permitiendo a *VladBot* orientarse en cualquier dirección.

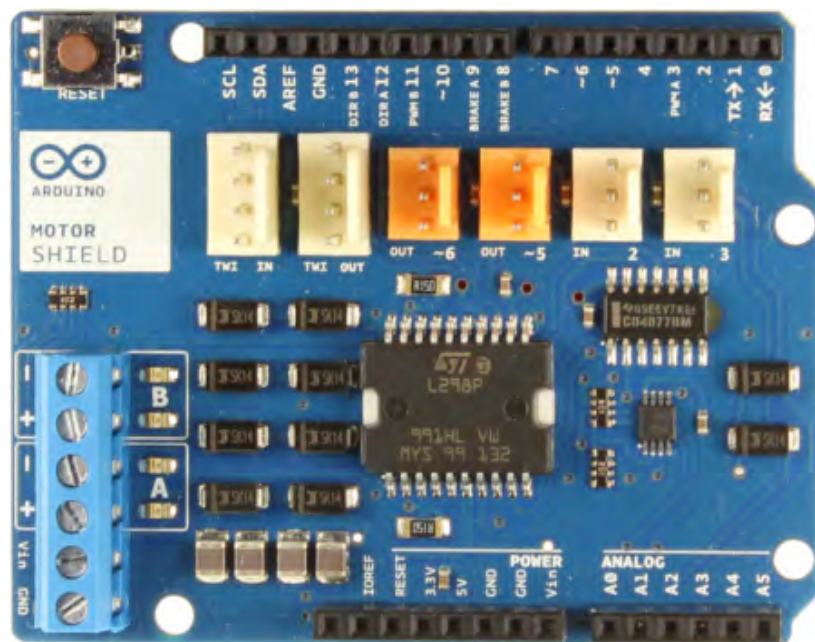


Figura 39: Arduino Motor Shield R3

### 2.3.6. ALIMENTACIÓN

La batería necesaria para mover todo el sistema es una pila de 9V recargable. Esta pila alimenta los motores de corriente continua, además de proporcionar el voltaje necesario para que la placa funcione. Aporta una de las características más importantes al robot, su autonomía, porque gracias a esta batería el robot no precisa de estar conectado a una toma de corriente. Hecho que dificultaría sus movimientos y limitaría su alcance.



Figura 40: Pila de 9V

La batería ha sido montada en el chasis del robot, en una carcasa que se ajusta a la pila. Permite un fácil acceso a ella, de manera que pueda ser sustituida fácilmente por otra, o para ser recargada.



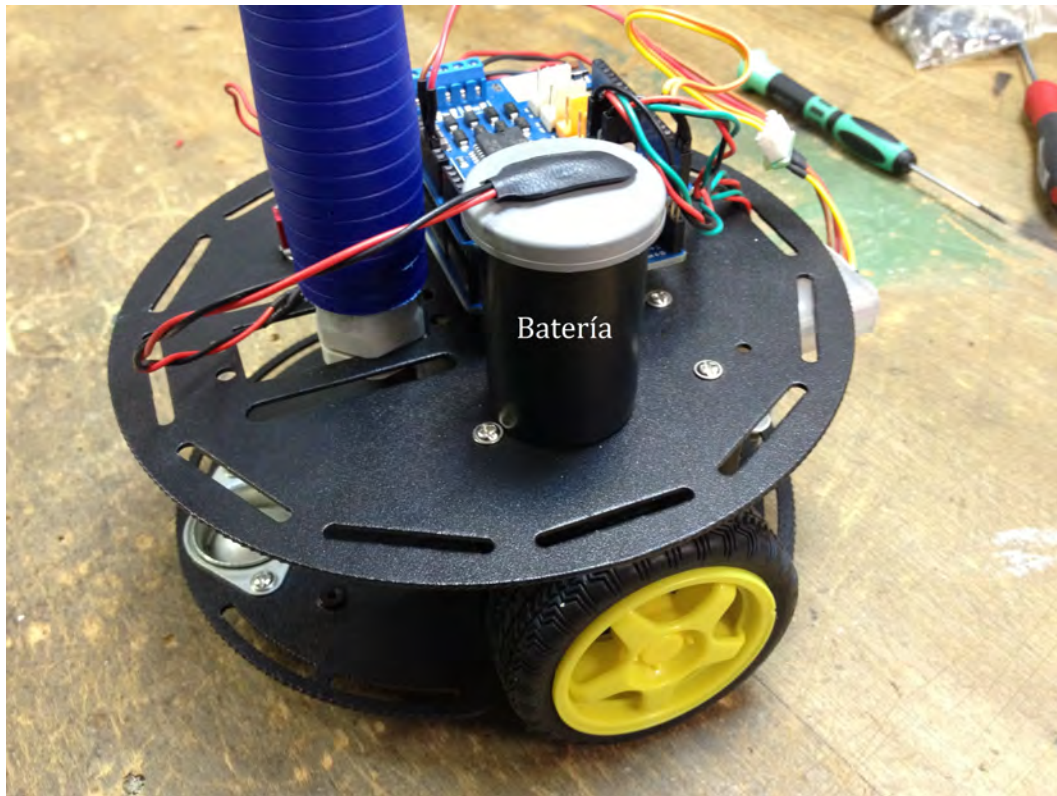


Figura 41: Ubicación de la batería en el chasis

#### 2.3.7. ENCODERS

Gracias a los *encoders*, o codificadores rotatorios, es posible utilizar la técnica de la odometría para posicionar el *VladBot*. Un *encoder* rotatorio es un dispositivo electromecánico que tiene como objetivo transformar una posición angular en un código digital. Utiliza un diodo emisor, el cual está emitiendo pulsos constantemente. Unas muescas en la rueda que se encuentra separándolo del diodo receptor deja pasar esta energía en determinados momentos, cuando la rueda gira. De esta forma, el diodo receptor puede contabilizar los pulsos emitidos, para posteriormente, con las técnicas de la odometría, posicionar el robot.

Los *encoders* utilizados para implementar el robot han sido los *encoders* de DFRobot [6]. Estos *encoders* resultan muy baratos y sencillos de integrar en la plataforma. Aparte de poseer un sencillo mecanismo de funcionamiento, consiguen una precisión más que suficiente.

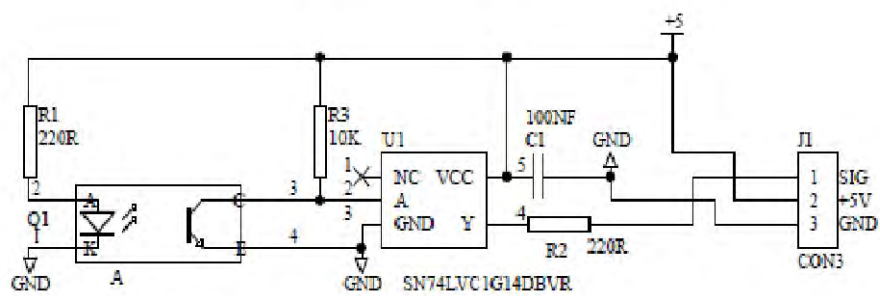


Figura 42: Esquema *encoder*

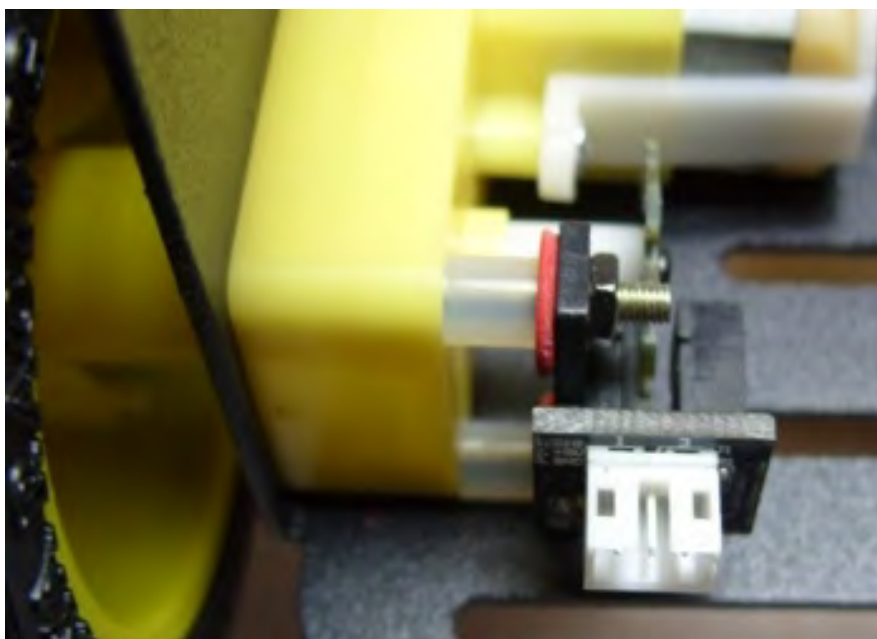


Figura 43: *Encoder* montado en la rueda

Especificaciones:

- Voltaje: +5V
- Corriente: <20 mA
- Resolución: 20 PPR
- Peso: 20 gramos

#### 2.3.8. COMUNICACIONES

*VladBot* es una plataforma móvil inalámbrica, es decir, la comunicación con el usuario se hace sin necesidad de estar conectado con un cable. Esto es una característica esencial de *VladBot*, ya que si no existiese esta comunicación inalámbrica, el desarrollo del sistema carecería de interés. Existen varias tecnologías de comunicaciones inalámbricas compatibles con el proyecto.

- XBee

ZigBee es una especificación de un conjunto de protocolos para comunicaciones inalámbricas. Está basada en la norma IEEE 802.15.4 de WPAN (*Wireless Personal Area Network*). Permite la comunicación entre dispositivos que se encuentren en la misma red. XBee, es la marca comercial de un módulo de radio que implementa estos protocolos [7].

Un módulo XBee ha presentado incompatibilidades a la hora de integrar el controlador de motores y este módulo de radio con la placa Arduino. Estas incompatibilidades se debían a que ambos módulos (Xbee y Arduino Motor Shield) comparten los mismo pines de la placa Arduino Uno, lo que hacía que sólo se pudiese integrar uno de los módulos. Y puesto que el controlador de motores resulta imprescindible, y existen tecnologías alternativas para las comunicaciones, se ha descartado XBee como tecnología para establecer las comunicaciones de *VladBot*.



Figura 44: Módulo XBee

- Bluetooth

La tecnología *bluetooth* permite comunicaciones inalámbricas entre diferentes tipos de dispositivos. Es una tecnología de comunicaciones de corto alcance, simple y segura. Numerosos dispositivos electrónicos ya vienen con esta tecnología implantada. Este hecho es muy favorable, ya que se ha querido que *VladBot* pueda ser controlado por cualquier usuario, sin necesidad de adquirir un equipo especial. De esta forma, cualquier ordenador personal (la mayoría posee un módulo de Bluetooth integrado) es suficiente para la utilización del robot.



Figura 45: Logotipo Bluetooth

Esta tecnología posee las siguientes características entre otras [8]. Se han presentado las características más relevantes o las más interesantes de cara a la comunicación que se ha deseado establecer entre el *VladBot* y el usuario.

- Espectro: desde 2,400 a 2,485 GHz (banda ISM).
- Interferencias: la tecnología Bluetooth implementa AFH (*Adaptive Frequency Hopping*), diseñado para reducir interferencias entre comunicaciones que compartan el mismo espectro.
- Rango: existen 3 clases, en las cuales se especifica el máximo alcance.
  - Clase 1: 100 metros
  - Clase 2: 10 metros
  - Clase 3: 1 metro
- Alimentación: esta tecnología consume muy poca energía, en torno a los 2,5 mW.

Debido a que es una tecnología muy económica, posee un alcance de hasta 100 metros (suficiente si se pretende controlar a VladBot desde una habitación contigua), su bajo consumo de energía y su espectro de trabajo (no interfiere con las ondas acústicas) se ha decidido que esta tecnología sea la utilizada para establecer las comunicaciones entre el robot y el usuario (ordenador). Otra razón de peso ha sido su completa integración, ya que los módems de Bluetooth disponibles en el mercado para las placas Arduino no han presentado ninguna incompatibilidad con el resto de los dispositivos integrados, a diferencia de XBee. Por otro lado se trata de una tecnología implantada en casi todos los ordenadores personales de la actualidad, por lo que no es necesario adquirir ningún equipo externo.

- Módulo de Bluetooth

De entre todos los módems Bluetooth disponible en el mercado, el módulo que se ha escogido para *VladBot* a sido el Modem Bluetooth Bluesmirf Gold. Éste módem

es muy económico y cubre perfectamente las necesidades del robot. Es ligero y de pequeño tamaño, ideal para ser integrado en la plataforma.



**Figura 46: Módulo Bluesmirf Gold**

Éste módulo funciona en modo serial (transmisor/receptor) y permite establecer una comunicación inalámbrica con una velocidad de transferencia de entre 9600 y 115200bps. Es de clase 1, por lo que el alcance aproximado es de unos 100 metros. Más que suficiente, debido a que el robot va a ser controlado desde una sala contigua. Puede ser alimentado con una tensión de alimentación de entre 3.3 y 6V, tensión que ofrece la placa Arduino. Puede conectarse aun ordenador (no directamente, sino con un conversor serie-USB).

El módulo se ha integrado en una zona visible en el chasis, para reducir las interferencias y se ha localizado dentro de una funda de plástico, que lo protege de golpes.

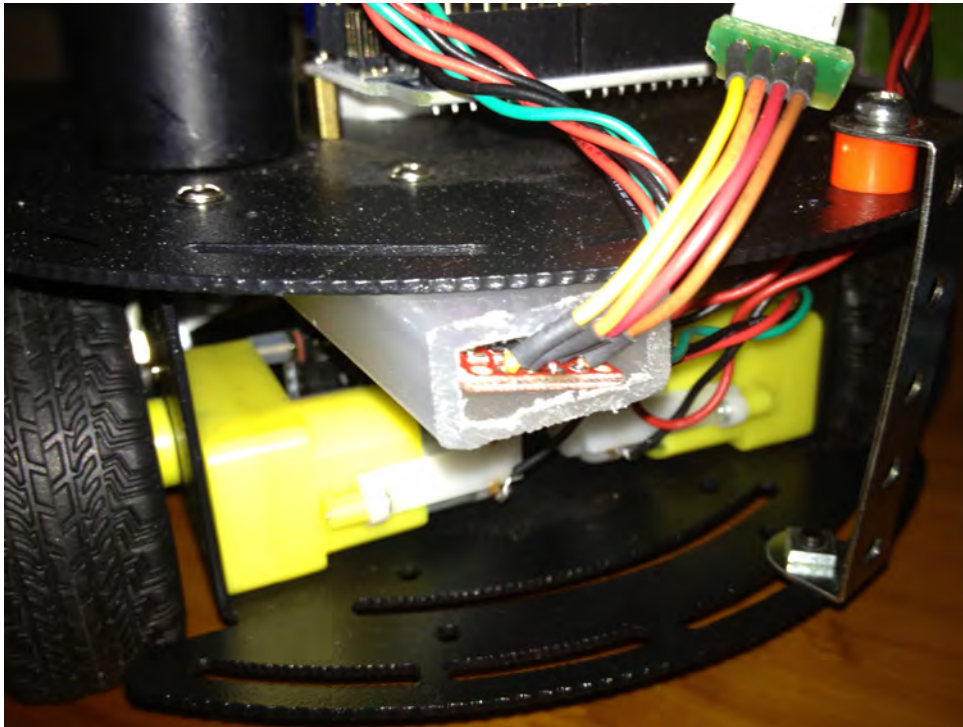


Figura 47: Ubicación del módem de Bluetooth



### 3. IMPLEMENTACIÓN

En este apartado se pretende justificar la integración de todos los elementos de diseño, así como presentar las funcionalidades de cada uno. Se muestran también porciones del código para que sea más comprensible al el lector comprender cómo se ha abordado cada funcionalidad del robot, aunque es recomendable familiarizarse antes con él. Para ello se puede consultar el anexo I.

#### 3.1. CONEXIONES

*VladBot* está alimentado con una pila de 9V, la cual suministra energía a la placa Arduino Uno, el controlador de motores, los motores, los *encoders*, y el módulo de Bluetooth. La batería se encuentra conexcionada a un conector jack-hembra de 2.1mm como el de la figura 37. Este conector está enchufado a el conector jack de la placa Arduino Uno. La placa necesita 5V para operar, pero es necesario de una fuente de alimentación de más voltaje para que los motores puedan mover toda la estructura.



Figura 48: Conector jack-hembra de 2.1 mm (centro positivo)



Se ha incorporado un interruptor de 2 posiciones (ON/OFF), para que se pueda apagar y encender el robot cuando se desee. Este interruptor se encuentra en una ubicación de fácil acceso en la plataforma. La conexión se ha realizado de forma que se desconecte el terminal positivo (posición OFF) para apagar el robot. Cuando se precisa de su funcionamiento, basta con cambiar la posición del interruptor para volver a cerrar el circuito.

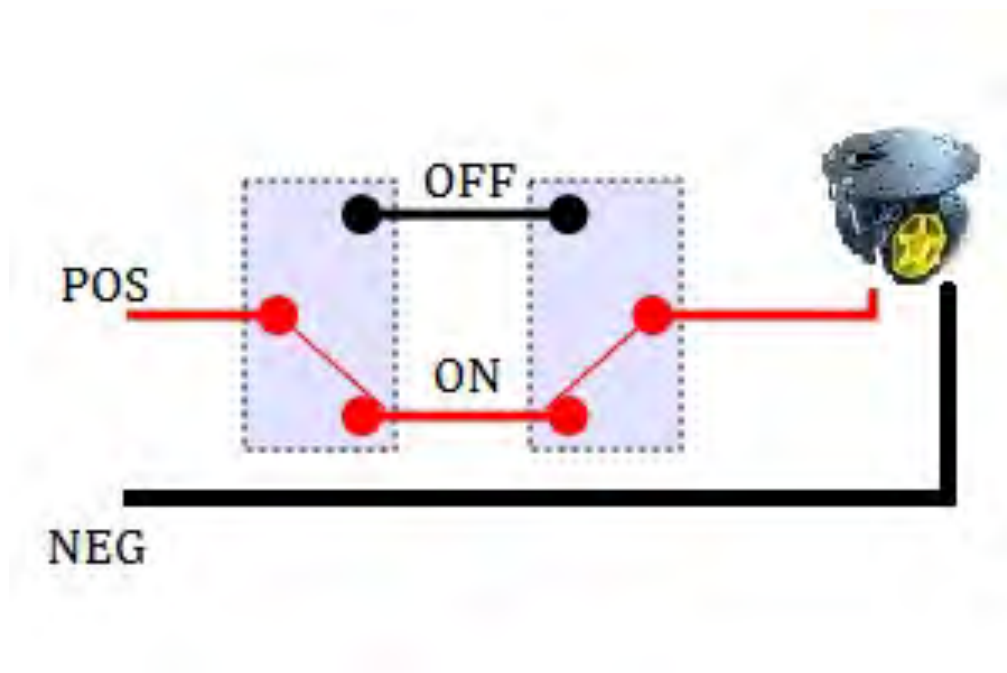


Figura 49: Diagrama de conexionado de la batería-interruptor

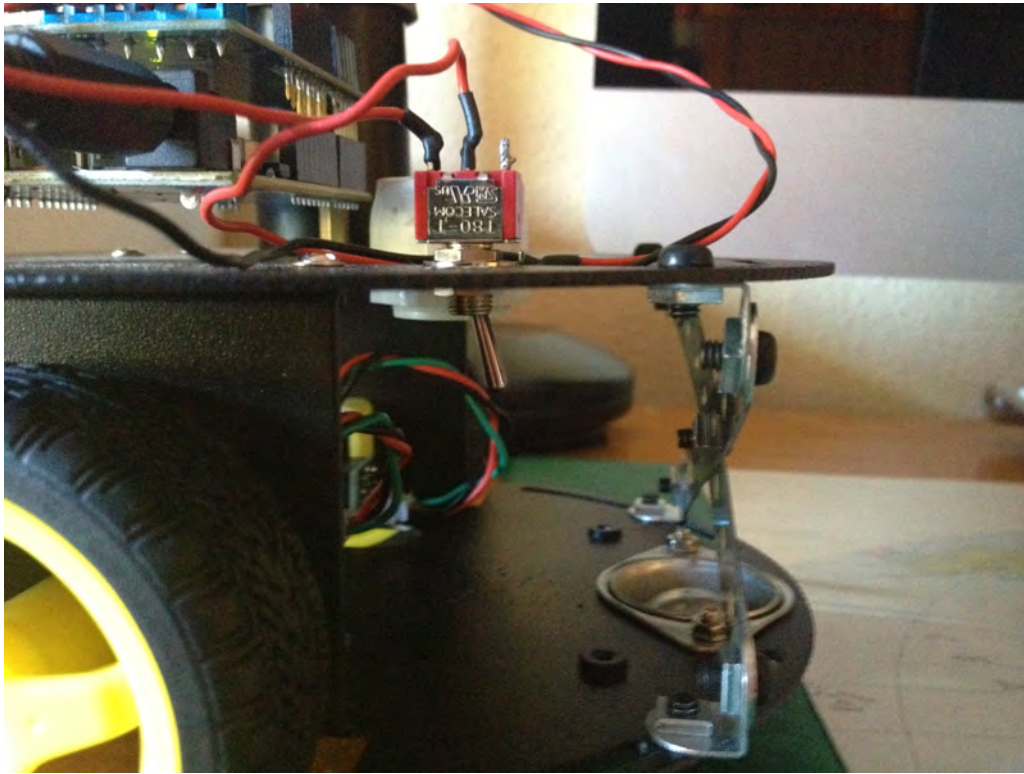


Figura 50: Posición de encendido (ON)

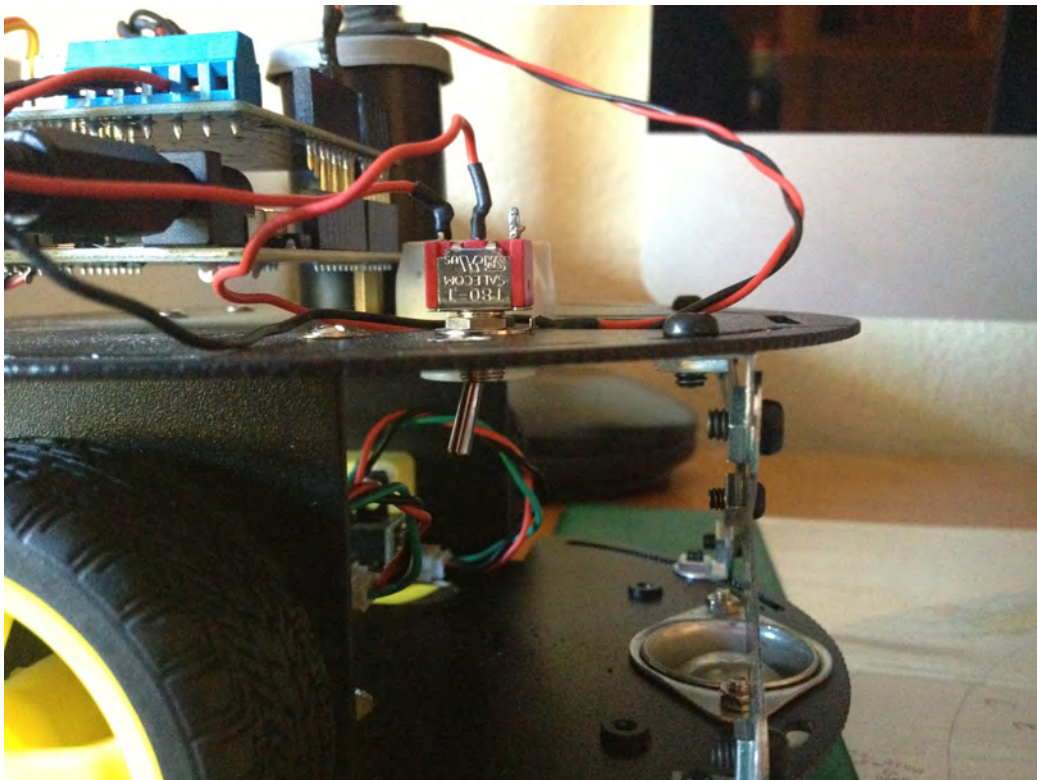


Figura 51: Posición de apagado (OFF)

Posteriormente, se ha agregado a la placa Arduino Uno el controlador de motores. No ha sido necesaria ninguna conexión, ya que se trata de un *shield* de Arduino, es decir, está fabricado expresamente para que su integración con una placa Arduino. Sólo ha sido necesario asegurarse que todos los pines entrasen correctamente donde les correspondía.

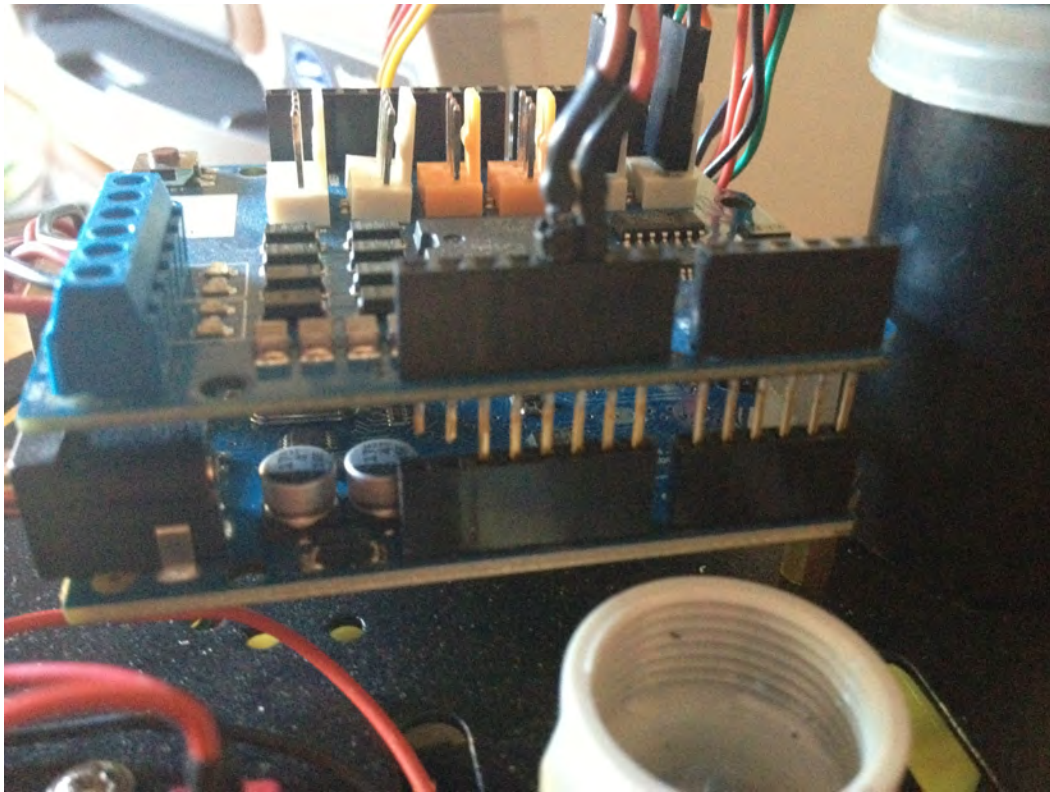


Figura 52: Conexión de la placa Arduino Uno y el controlador de motores

A continuación, se han conectado los motores. El controlador posee 2 entradas independientes (A y B), una para controlar los parámetros de cada motor. De esta forma, el motor derecho corresponde a la entrada A, quedando la entrada B para el izquierdo.



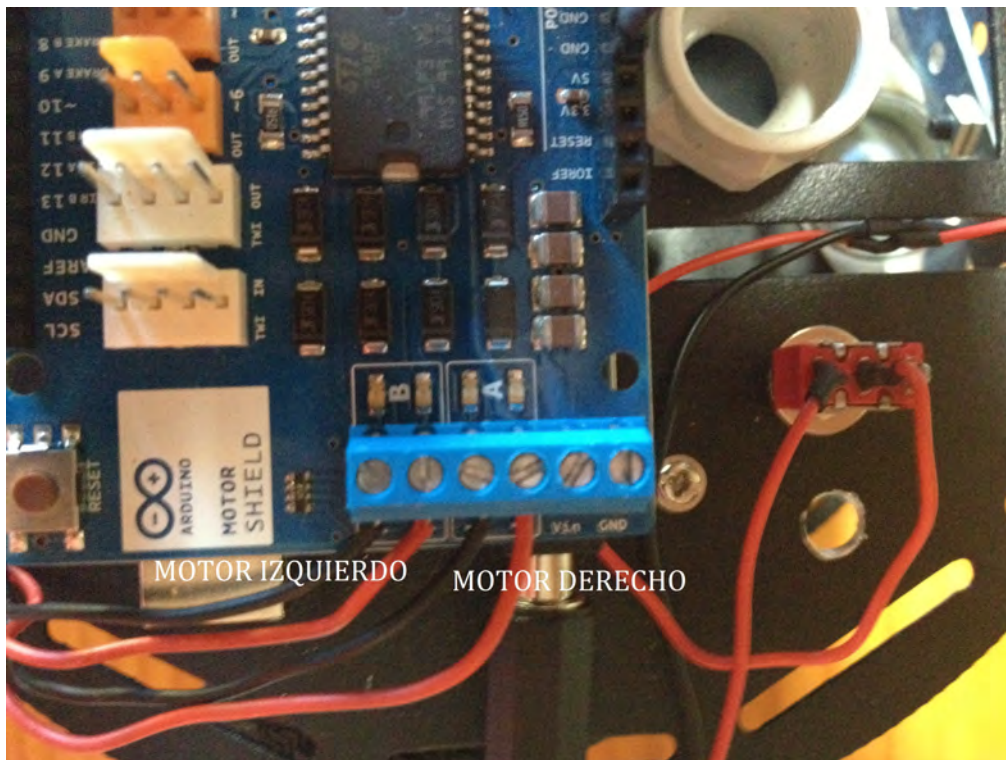


Figura 53: Conexión de los motores

El siguiente paso a sido la conexión de los *encoders*. Como se puede observar en la figura 42, los *encoders* poseen 3 salidas: SIG (señal), GND (masa o tierra) y +5V (alimentación). Se conexión se hace a través de los conectores *TinkerKit* del controlador de motores.

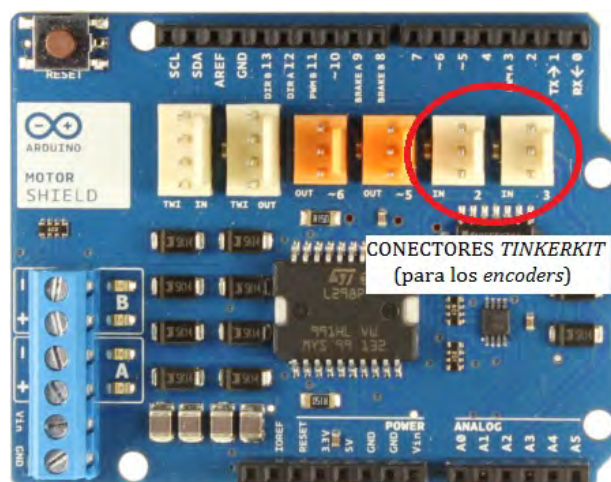


Figura 54: Ubicación de los conectores para los *encoders*

Estos conectores se encuentran directamente conectados a la alimentación (+5V, para que los *encoders* reciban el voltaje necesario para operar), a la masa de la placa (GND) y a las entradas analógicas A2 y A3 (la señal SIG de cada conector). Así, el encoder de la rueda derecha corresponde a la entrada A2, y el de la izquierda a la A3 (figura 55).

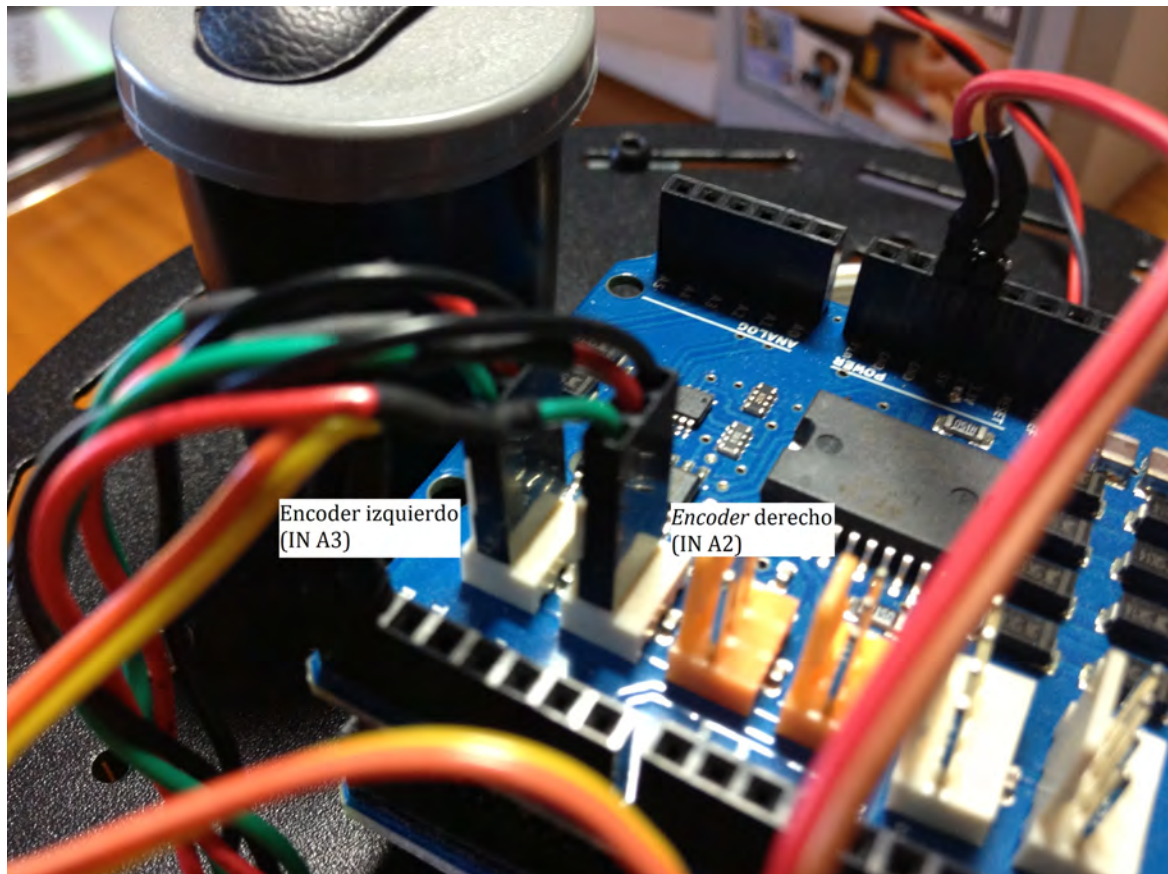


Figura 55: Conexión *encoders*

Por último, se ha integrado el módem Bluetooth. El módulo posee 6 conectores: 2 para alimentación (VCC y GND), transmisión y recepción (TX y RX) y otros 2 para señales de control (CTS y RTS) que no se han utilizado. Cabe destacar, que la conexión del receptor (RX) en la placa, debe estar conectado al transmisor del módem (TX) para que pueda establecerse la comunicación. La figura 56 muestra el diagrama del cableado.

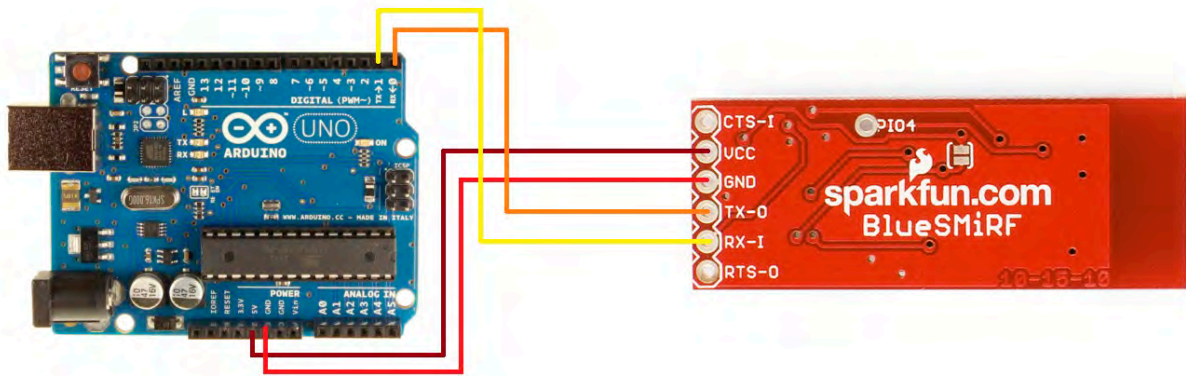


Figura 56: Diagrama de conexión del módem Bluetooth a la placa Arduino Uno

La conexión se ha hecho a través un pequeño conector, que permite retirar el módulo de una forma sencilla. Esto se ha diseñado así debido a que la programación de la placa se hace a través de su puerto serial (conector USB del Arduino Uno), mismo puerto que usa el módulo para establecer la comunicación. Al intentar cargar un programa en la placa, se produce un conflicto en el puerto entre el módem Bluetooth y el USB , de manera que resulta imposible la carga de ningún programa mientras el módulo de Bluetooth se encuentra conectado. Por esta razón es necesario retirar el módulo cuando se desee hacer alguna modificación en el *software* de *VladBot*.

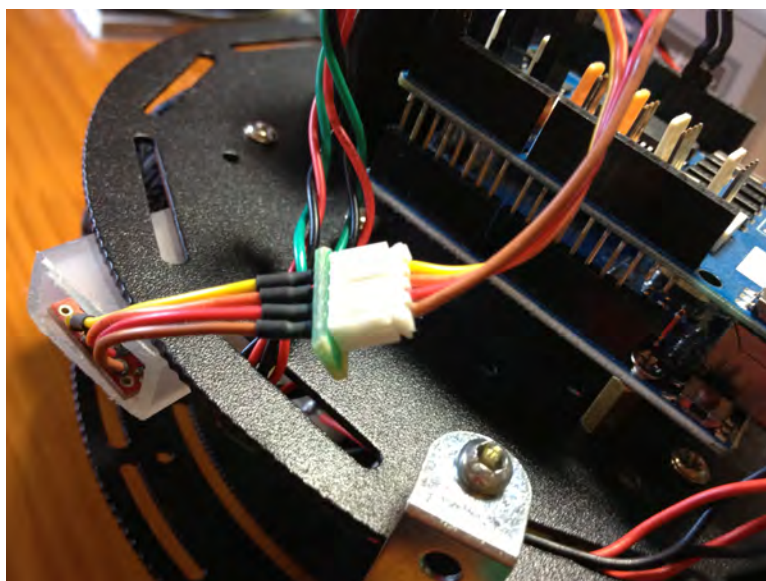


Figura 57: Conexión del módem Bluetooth



### 3.2. ODOMETRÍA

La técnica elegida para el posicionamiento del robot ha sido la odometría. Este método utiliza los *encoders* para medir la rotación de las ruedas o su orientación, y poder posicionar a *VladBot*. Debido a que la plataforma posee dos ruedas móviles, son necesarios dos *encoder*, uno para cada rueda. Los *encoders* cuentan las revoluciones de cada rueda por separado. Cada función, una para el *encoder* derecho y otra para el izquierdo, incrementa en uno la posición del *encoder*. Estas funciones cuentan los pulsos de onda que son emitidos por los diodos. Gracias a ellas es posible determinar cuantos pulsos de onda, ha avanzado cada rueda.

```
void contarIzq(){  
    EncoderPosIzq++;  
}  
  
void contarDer(){  
    EncoderPosDer++;  
}
```

Fragmento de código 1: Funciones de los *encoders*

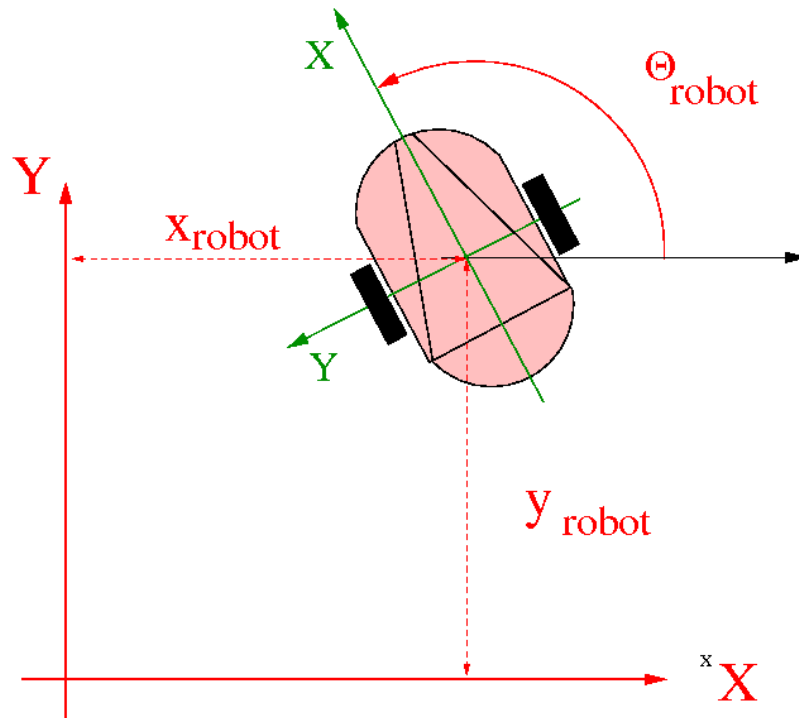


Figura 58: Posicionamiento odometría

El inconveniente de esta técnica es que el error de la medida se incrementa a medida que el vehículo se aleja de su referencia. Pero como se ha expuesto en apartados anteriores, el diseño se ha hecho en lazo abierto, o dicho de otra manera, en cada iteración (movimiento o giro) se establece una nueva referencia. Esto reduce considerablemente el error, haciendo viable esta técnica.

Los movimientos que se le permiten a *VladBot* son: de avance (adelante o hacia atrás) y de rotación (orientación en una dirección). Con estos dos tipos de movimiento se puede acceder a cualquier punto de una sala. No se han contemplado las trayectorias curvilíneas, ya que no son de gran utilidad, puesto que en el mundo de la acústica no es de interés medir niveles de presión sobre una trayectoria curva. Además, esto reduce considerablemente la carga computacional del robot, haciendo que su respuesta sea más rápida.

Los *encoders* utilizados para tal propósito son unos sencillos *encoders* ópticos que producen un cierto número de pulsos de onda cuadrada por revolución. Estos pulsos son recogidos por la placa Arduino Uno, que transforma estos pulsos a su correspondiente distancia o distancia angular.





Figura 59: *Encoders*

En este apartado se hace énfasis en cómo se ha aplicado esta técnica al robot. No se pretende un estudio sobre la odometría, aunque si el lector quiere ampliar información sobre el tema puede consultar [2].

Se han diferenciado dos movimientos, avance y giro, los cuales se procesan de manera distinta.

- Avance

El usuario introduce los centímetros que quiere que el *VladBot* avance. La placa Arduino Uno hace de traductor entre los *encoders*, los cuales entienden sólo de pulsos, y el usuario, que conoce las distancias en centímetros. Se ha elegido que la distancia se mida en centímetros debido a que este es el máximo error que introduce el sistema. En apartados posteriores se justifica esta decisión.

```
//case "go"
case 0:
    Serial.print( "Moviendome hacia adelante " );
    Serial.print(val);
    Serial.println( " centimetros.." );
    while((EncoderPosIzq)<New_ticks){
        adelante();
    }
```

**Fragmento de código 2: Caso de movimiento hacia adelante**

El fragmento de código 2 muestra el caso en el que *VladBot* ha de moverse hacia delante. Se le indica una cantidad que ha de avanzar, “New\_ticks”, y el robot se estará moviendo hacia delante hasta que el *encoder* izquierdo alcance esta nueva posición. “New\_ticks” es la cantidad en pulsos de onda, ya que *VladBot* no “entiende” de distancias en metros, si no de pulsos de onda, es necesaria una transformación. En el caso del que el robot tenga que moverse hacia atrás, simplemente se cambia el sentido de giro de los motores. El funcionamiento es exactamente el mismo.

```

void adelante (){//MOVIMIENTO HACIA ADELANTE
//Se establecen las direcciones de los motores
digitalWrite(Direccion_DER, HIGH);
digitalWrite(Direccion_IZQ, HIGH);
//Se deshabilita la parada de los motores
digitalWrite(Parada_DER, LOW);
digitalWrite(Parada_IZQ, LOW);
//PID
int velIniIzq=70;
int velIniDer=70;
int velIzq=velIniIzq;
int velDer=velIniDer;

if(EncoderPosIzq<EncoderPosDer){
    velIzq=velIzq+10;
}
if(EncoderPosIzq>EncoderPosDer){
    velDer=velDer+10;
}
//Se giran las ruedas a determinada velocidad
analogWrite(Velocidad_DER, velDer);
analogWrite(Velocidad_IZQ, velIzq);

}

```

**Fragmento de código 3: Movimiento hacia adelante**

Mediante la fórmula 1 se establece la transformación:

$$\left( \text{Centímetros} / \frac{(\text{Diámetro} \cdot \pi)}{\text{Pulsos}} \right) + 0,5 \quad (1)$$

Donde:

- Centímetros: número de centímetros que el usuario quiere que se avancen.
- Diámetro: es el máximo diámetro de la rueda, el que hace contacto con la superficie. En el *VladBot* son 6,6 cm.
- Pulsos: es el número de pulsos por revolución de los *encoders*. Los *encoders* del robot generan 10 pulsos por revolución. Éste es el parámetro culpable de la precisión del sistema.
- El 0,5 que se suma es debido al *cast* de Arduino. El procesador de la placa siempre trunca los resultados, así, para mayor precisión se suma esta cantidad para que redondee el resultado y no lo trunque.

Con la distancia traducida a pulsos, la placa Arduino Uno ya sabe cuántos pulsos de los *encoders* son necesarios para avanzar la distancia exigida. El robot avanza hasta que se complete este número de ciclos. El fragmento de código 4 muestra cómo se ha implementado la fórmula 1 en la placa Arduino Uno.

```
//Transforma de centímetros a numero de ticks (pulsos)
int cmToticks(int num){
    float diametro=6.6; //centímetros
    int ticks_vuelta=10;
    float pi=3.14;

    float cm_por_tick = (diametro*pi)/ticks_vuelta;
    int res = (int) (num/cm_por_tick + 0.5);
    //El +0,5 es para redondear, ya que el cast de Arduino trunca
    return res;
}
```

Fragmento de código 4: Implementación de la fórmula 1

Como complemento adicional, se ha introducido un pequeño control de PID (Proporcional Integral Derivativo). El PID es un mecanismo de retroalimentación que calcula el error entre el valor medido y valor esperado. Se introduce en un sistema para aplicar una acción correctora que ajuste el proceso. Se da en tres parámetros distintos: el proporcional, el integral y el derivativo. En el caso del robot, este mecanismo se aplica para asegurar que el avance del vehículo es completamente recto.

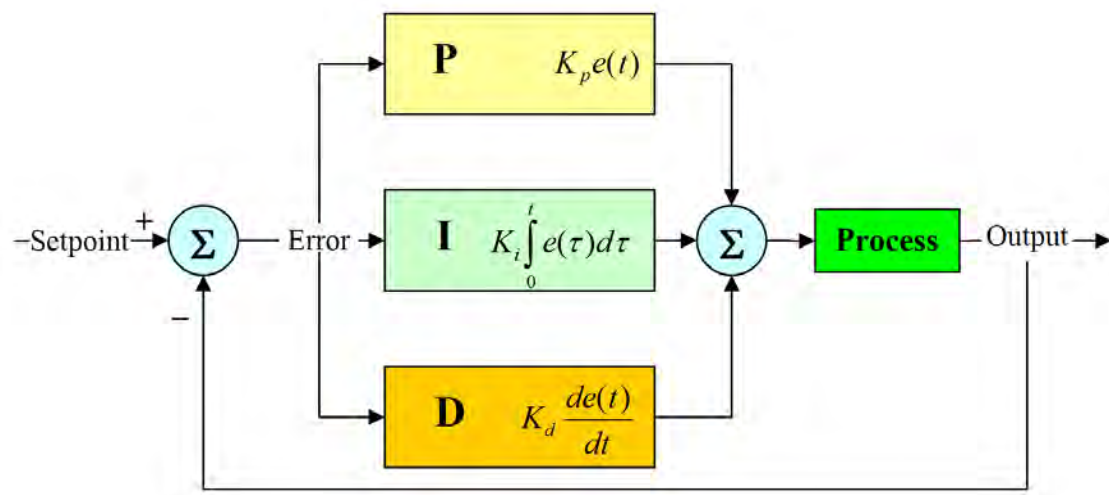


Figura 60: Control PID

El mecanismo de funcionamiento se basa mantener los *encoders* siempre con el mismo número de pulsos emitidos. En cuanto uno de los encoders posee una cuenta de pulsos inferior/superior al otro (esto quiere decir que el robot estaría dibujando una trayectoria curva) enseguida se aumenta/disminuye la velocidad de la rueda contraria hasta que se alcanza una medida deseada. Esta acción es la encargada de corregir el error producido.

```
//PID
int velIniIzq=70;
int velIniDer=70;
int velIzq=velIniIzq;
int velDer=velIniDer;

if(EncoderPosIzq<EncoderPosDer){
    velIzq=velIzq+10;
}
if(EncoderPosIzq>EncoderPosDer){
    velDer=velDer+10;
}
```

Fragmento de código 5: Implementación PID

- Giro

El giro de la plataforma se puede considerar un caso especial de avance. La rotación del robot se ha conseguido haciendo girar una de las ruedas hacia delante y la otra hacia atrás. Dependiendo de si se trata de un giro hacia la derecha o hacia la izquierda el sentido de rotación de los motores es uno u otro.

De la misma forma que en el caso del avance, se ha establecido una traducción entre el usuario y la placa. En este caso el usuario introduce los grados que quiere que la plataforma gire, en un sentido deseado, y la placa Arduino Uno se encarga de traducirlo a pulsos de *encoder*. Los giros permitidos por el robot van de 15 en 15 grados, hasta un máximo de 360 grados (una vuelta completa). Nuevamente esta limitación la establecen los *encoders*. Pero es más que suficiente para realizar cambios de sentido o giros de 90 grados con la plataforma, no siendo necesario más precisión en este tipo de movimiento ya que no se pretende que el robot realice complejas maniobras.

```

//case "right"
case 3:
    Serial.print( "Girando hacia la derecha " );
    Serial.print(val);
    Serial.println( " grados..." );
    if (val>360){
        Serial.println( "Introduzca un valor menor de 360 grados" );
        break;
    }
    ticks=((val/15) +0.5);
    New_ticks = ticks + EncoderPosIzq;

    while(EncoderPosIzq<New_ticks){
        derecha();
    }

```

Fragmento de código 6: Caso de giro hacia la derecha

Con la fórmula 2 se ha conseguido esta transformación:

$$\left(\frac{\text{Grados}}{15}\right) + 0,5 \quad (2)$$

```

void derecha()
{
    int velIniIzq=70;
    int velIniDer=70;
    int velIzq=velIniIzq;
    int velDer=velIniDer;

    //Se establecen las direcciones de los motores
    digitalWrite(Direccion_DER, LOW);
    digitalWrite(Direccion_IZQ, HIGH);

    //Se deshabilita la parada de los motores
    digitalWrite(Parada_DER, LOW);
    digitalWrite(Parada_IZQ, LOW);

    //Se giran las ruedas a determinada velocidad
    analogWrite(Velocidad_DER, velDer);
    analogWrite(Velocidad_IZQ, velIzq);
}

```

Fragmento de código 7: Función de giro a la derecha

### 3.3. INTERRUPCIONES

Arduino posee una herramienta llamada interrupciones. En la placa Arduino Uno existen dos fuentes de interrupciones externas, asociadas a los pines digitales 2 y 3 (conectados con los *encoders*). Funcionan de la siguiente manera: cada vez que se produzca un evento (el cual puede programarse) el microcontrolador deja la tarea actual que se esté ejecutando para saltar al código asociado a esa interrupción. Las interrupciones son muy útiles para automatizar operaciones. *VladBot* utiliza esta capacidad de la placa Arduino Uno para contar las revoluciones de las ruedas, es decir, contar el número de pulsos de onda cuadrada emitido por los *encoders*. La plataforma Arduino ofrece una biblioteca, “*attachInterrupt*” que implementa esta funcionalidad:



`attachInterrupt (interrupción, función, modo)`

```
//Interrupciones  
  
PCintPort::attachInterrupt(ENCODER_IZQ, &contarIzq, RISING);  
  
PCintPort::attachInterrupt(ENCODER_DER, &contarDer, RISING);
```

**Fragmento de código 8: Declaración de interrupciones**

- Interrupción: es el número de la interrupción, o sea, la interrupción 0 (pin digital 2) o la interrupción 1 (pin digital 3). En *VladBot* cada una de estas interrupciones corresponden a una rueda. Hay un aspecto importante que señalar y es que el pin digital 3 es exclusivamente utilizado por el controlador de motores para establecer la velocidad de la rueda derecha. Esto no ha sido un problema, ya que también en la plataforma Arduino existe una biblioteca que permite cambiar los pines de las interrupciones: “*pinChangeInt*”. Con esta utilidad, las interrupciones han quedado resueltas de la siguiente manera:
  - Interrupción rueda derecha: pin 16 (pin analógico 2).
  - Interrupción rueda izquierda: pin 17 (pin analógico 3).
- Función: incrementar el número de pulsos, dependiendo de cada rueda.
- Modo: existen 4 modos de funcionamiento

- LOW: dispara la interrupción cuando el pin se encuentra en valor bajo (LOW).



Figura 61: Modo de funcionamiento LOW

- CHANGE: dispara la interrupción cada vez que el pin cambia de valor.

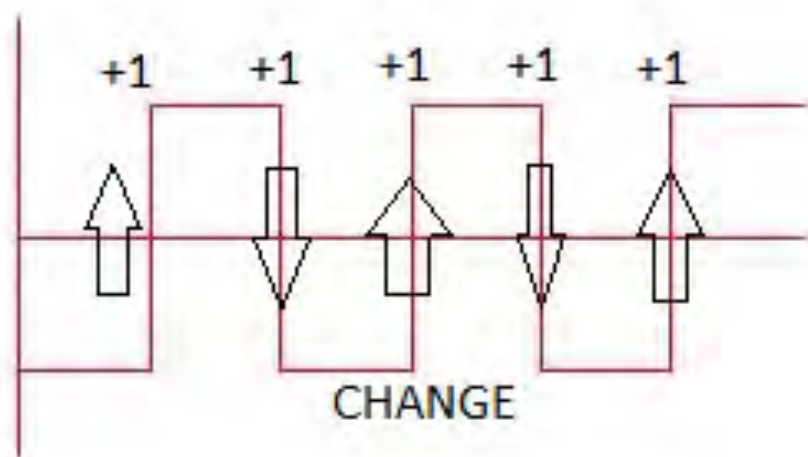


Figura 62: Modo de funcionamiento CHANGE

- FALLING: dispara la interrupción cuando el pin pasa de un valor alto (HIGH) a bajo (LOW).

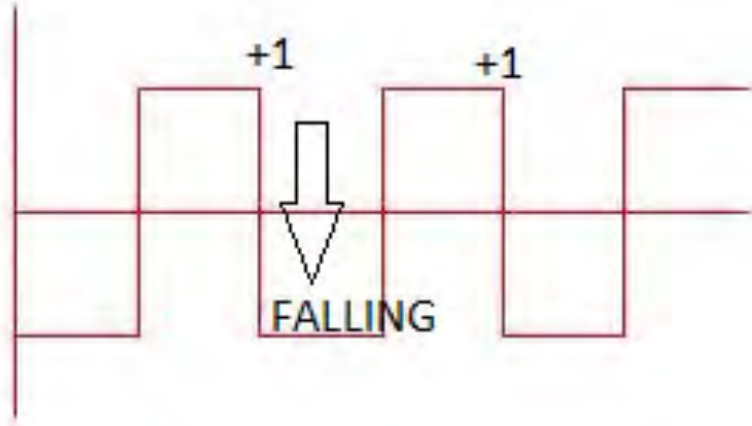


Figura 63: Modo de funcionamiento FALLING

- RISING: dispara la interrupción cuando el pin pasa de un valor bajo (LOW) a alto (HIGH).

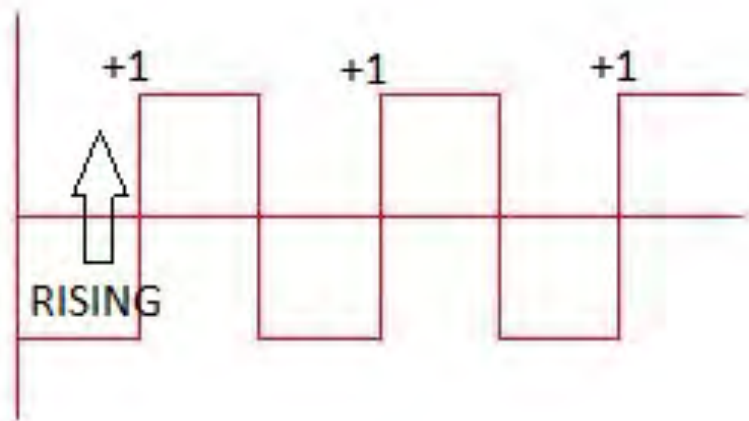


Figura 64: Modo de funcionamiento RISING

Este último modo de funcionamiento (RISING) ha sido el elegido para el robot, ya que de esta forma la cuenta de los pulsos es más precisa. Con el modo LOW se podrían dar situaciones en las que no se contasen los pulsos, y con el modo CHANGE se podría perder la cuenta de alguno, ya que los cambios son más rápidos. El modo FALLING se podría haber utilizado indistintamente.

### 3.4. PROTOCOLOS

Ha sido necesario encapsular la información con un protocolo de comunicaciones: JSON (*JavaScript Object Notation*). Este protocolo permite el intercambio de información entre el ordenador del usuario y la placa Arduino Uno, que controla a *VladBot*. Incluso podría comunicarse, gracias a este protocolo, con otro un *software* externo como podría ser Processing o un servidor web.

JSON es un protocolo de comunicaciones que permite esta funcionalidad, codificando y decodificando los mensajes que emiten los nodos. JSON es una alternativa más sencilla al intercambio de información estructurada que XML. La principal ventaja de JSON sobre XML es su simplicidad a la hora de escribir un *parser* (analizador sintáctico) [9]. Por eso JSON es una herramienta sencilla de implementar y una forma eficiente de transmitir datos.

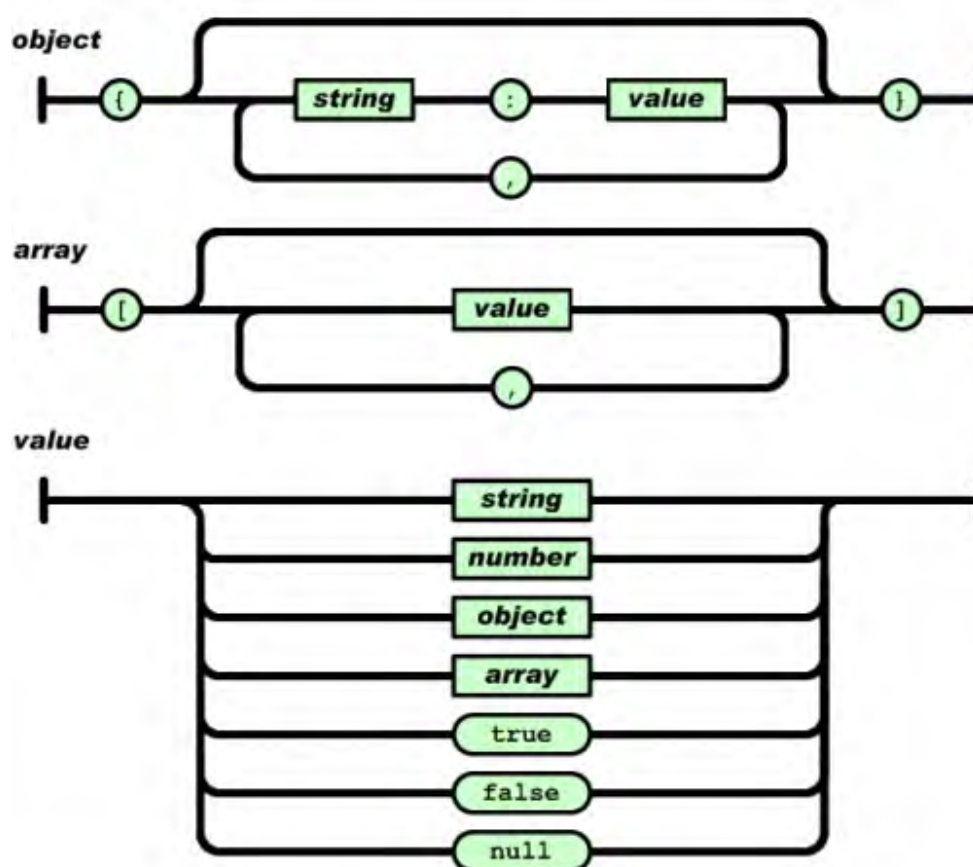


Figura 65: JSON [9]

Arduino ofrece una biblioteca que permite implementar este protocolo en la placa Arduino Uno: `aJson`. Por esta razón (fácil y sencilla integración) aparte de las funcionalidades que puede ofrecer cualquier protocolo de comunicaciones, es por la que se ha decidido que JSON sea el protocolo con el que se comunique *VladBot*.

Cada mensaje que se envía desde el ordenador al robot es un objeto `aJson`. Este objeto contiene la información del movimiento (cadena de caracteres) y el valor de éste (valor numérico). El protocolo encapsula el mensaje que va a ser enviado por el puerto serial (Bluetooth) al robot. Una vez recibido en la placa, el microcontrolador se encarga de *parsear* este mensaje para extraer la información que le interesa: el tipo de movimiento y su cantidad (distancia de avance o grados de giro). Finalmente, una vez se ha realizado con éxito la orden, el robot envía un mensaje de confirmación (otro objeto `aJson`) con el fin de informar al usuario de que a concluido la orden enviada.

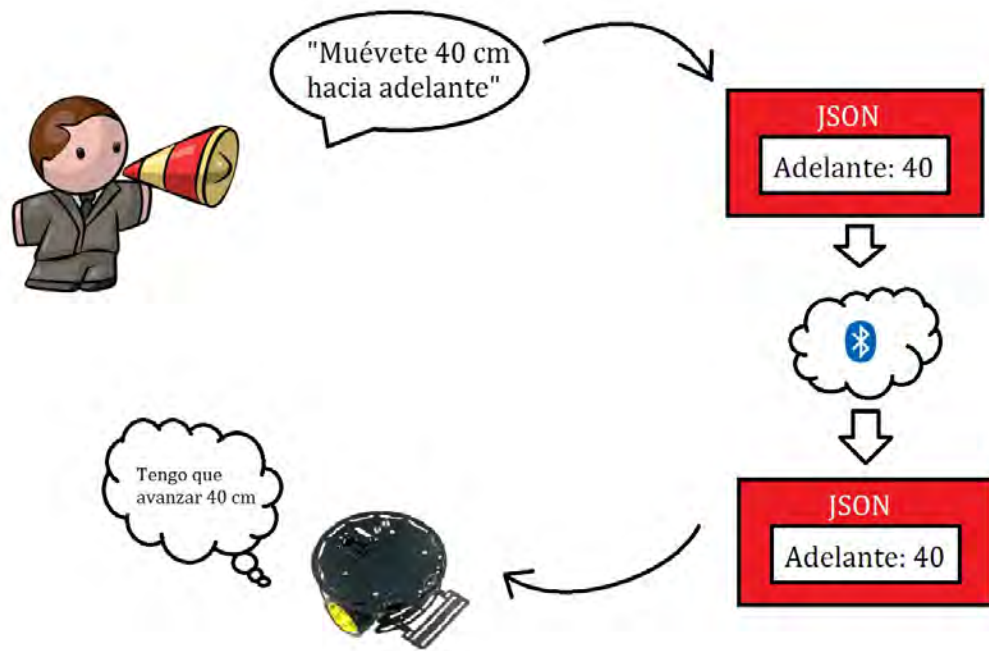


Figura 66: Esquema de funcionamiento de las comunicaciones

Por tanto, las órdenes que se pueden enviar al robot desde el monitor serial del IDE de Arduino (o cualquier emulador de terminal, por ejemplo, Zterm para MacOS) se escriben de la siguiente manera:

```
{“orden”:cantidad}
```

Siendo la órdenes que puede interpretar *VladBot*:

- “go”: avance hacia delante.
- “back”: avance hacia atrás.
- “left”: giro hacia la izquierda.
- “right”: giro hacia la derecha.

A modo de ejemplo, si se quiere avanzar hacia atrás 2 metros, hay que mandarle la orden de movimiento ("back") y la cantidad en centímetros (200). Siempre entre corchetes y separados por dos puntos. Las órdenes se escriben entre comillas.

```
{"back":200}
```

De la misma manera, si se desea girar 90 grados a la izquierda, se escribe el sentido del giro ("left") y la cantidad de grados que se quiere girar (90).

```
{"left":90}
```

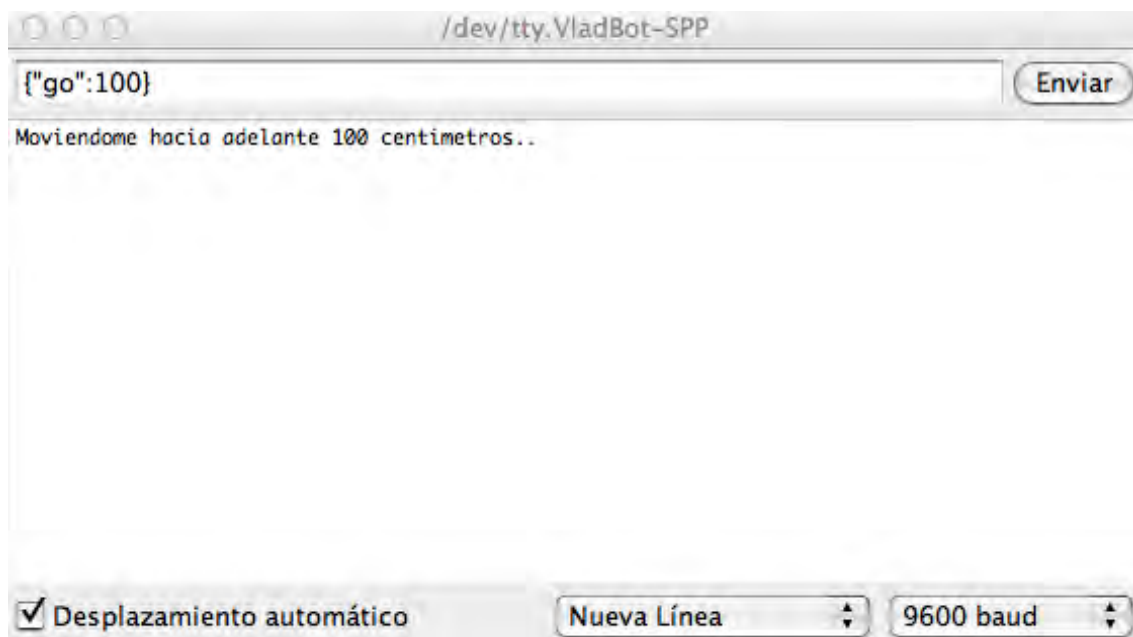


Figura 67: Monitor serial

En el fragmento de código 9 se puede observar el funcionamiento detallado del protocolo. En primer lugar, un mensaje del tipo de la figura 67 es parseado, es decir, extrae la información que contiene (una cadena, "pstr", con el tipo de movimiento, y un valor, "pval", cantidad de esta movimiento). Se comprueba que la cadena sea alguna de las 4 posibilidades de movimiento y estos valores se pasan a



la función "cmdProcess" (procesamiento de comandos). Esta función es la encargada de decidir el tipo de movimiento que se efectúa y la cantidad del mismo.

```
aJsonObject* root = aJson.parse(msg); //Mensaje del tipo:
                                     {"go":200}

aJsonObject *pval;
char *pstr;

for(int j = 0; j < 4; j++) { //Lo que hace este bucle es ir
                             "preguntando" si se ha introducido
                             alguna de las palabras clave para
                             indicar la direccion de movimiento.
                             Si no es ninguna, se sale del bucle, y
                             borra el objeto aJson

    if(j == 0) {
        pstr = "go" ;
    } else if(j == 1) {
        pstr = "back" ;
    } else if(j == 2) {
        pstr = "left" ;
    } else if(j == 3) {
        pstr = "right" ;
    }
    pval = aJson.getObjectItem(root, pstr);
    if(pval) {
        cmdProcess(j, pval->valueint); //Procesamiento de comandos, se
                                     le pasa el valor de "j",
                                     correspondiente al movimiento

        break;
    }
}
aJson.deleteItem(root); //Al final se elimina el objeto
```

Fragmento de código 9: Funcionamiento del protocolo JSON

### 3.5. PROGRAMACIÓN

La programación de la placa se ha realizado con el IDE de Arduino. En este apartado se dan a conocer algunos de los aspectos más relevantes del código, pero para más detalle se puede consultar el anexo I, dónde se encuentra íntegramente y detallado.

El código de la placa cuenta con dos funciones indispensables:

- `Setup()`: establece los pines de entrada y los de salida.
- `Loop()`: es la función que se repite en bucle.

Antes de establecer los pines como entradas/salidas, se han añadido las bibliotecas. Es de especial relevancia ya que, como se ha comentado en apartados anteriores, se usan dos bibliotecas externas. La biblioteca para el cambio de los pines de interrupción (consultar apartado 3.3.) y la biblioteca necesaria para el uso del protocolo Json.

```
#include <PinChangeInt.h>
#include <stdio.h>
#include <stdlib.h>
#include <aJSON.h>
```

**Fragmento de código 10: Bibliotecas utilizadas**

A continuación, para que le sea más intuitivo al programador, se le ha dado nombre a los pines. Si el pin encargado de parar el motor derecho es el 9, se define el pin 9 con el nombre *Parada\_DER*.

```

//Se establecen los pines
#define ENCODER_DER 16//interrupcion pin analogico A2
#define ENCODER_IZQ 17//interrupcion pin analogico A3
volatile long EncoderPosDer = 0;
volatile long EncoderPosIzq = 0;
//Volatile: siempre que su valor pueda ser modificado por algo más
allá de la sección del código en el que aparece, por ejemplo
interrupciones

#define BUFLLEN 32 //Se define el tamaño del buffer en el que se
almacenan los datos de entrada

int Direccion_DER = 12; //Direccion del motor derecho en el pin 12
int Direccion_IZQ = 13; //La del izquierdo en el 13

int Velocidad_DER = 3;
int Velocidad_IZQ = 11;

int Parada_DER = 9;
int Parada_IZQ = 8;

```

**Fragmento de código 11: Declaración de los pines**

En la función `setup()` se han llevado a cabo 4 acciones:

- Establecer las salidas: de los pines que controlan los motores.
- Inicializar el puerto serie: 9600 baudios es la velocidad con la que trabaja la comunicación entre el ordenador del usuario y el robot. Es muy importante que en ambos nodos esta velocidad sea la misma.
- Establecer las interrupciones.
- Inicializar las variables que controlan los *encoders*.

```

void setup(){
    //Se establecen las salidas
    pinMode(Direccion_DER, OUTPUT);
    pinMode(Direccion_IZQ, OUTPUT);

    pinMode(Velocidad_DER, OUTPUT);
    pinMode(Velocidad_IZQ, OUTPUT);

    pinMode(Parada_DER, OUTPUT);
    pinMode(Parada_IZQ, OUTPUT);

    //Se inicializa el puerto serial
    Serial.begin(9600);

    //Interrupciones
    PCintPort::attachInterrupt(ENCODER_IZQ, &contarIzq, RISING);
    PCintPort::attachInterrupt(ENCODER_DER, &contarDer, RISING);

    //Inicializacion de las variables de los encoders
    int EncoderPosIzq=0;
    int EncoderPosDer=0;
    int ticks=0;

}

```

Fragmento de código 12: Función setup()

Para hacer el código más sencillo, y que pueda ser modificado, las funciones del robot se han aislado. Se ha definido una función independiente para cada acción que debe llevar a cabo el micro-procesador. Si el usuario desea ampliar o modificar el código, puede hacerlo fácilmente sin necesidad de modificar estas funciones primitivas. Así pues, las funciones que implementa *VladBot* son:

- contarIzq(): cuenta la posición del *encoder* izquierdo.
- contarDer(): cuenta la posición del *encoder* derecho.

```
void contarIzq(){  
    EncoderPosIzq++;  
}  
void contarDer(){  
    EncoderPosDer++;  
}
```

**Fragmento de código 13:** Funciones para contar posición de los *encoders*

- adelante(): se encarga del avance hacia delante. En esta función se incluye el control PID.

```
void adelante (){//MOVIMIENTO HACIA DELANTE

//Se establecen las direcciones de los motores
digitalWrite(Direccion_DER, HIGH);
digitalWrite(Direccion_IZQ, HIGH);

//Se deshabilita la parada de los motores
digitalWrite(Parada_DER, LOW);
digitalWrite(Parada_IZQ, LOW);

//PID
int velIniIzq=70;
int velIniDer=70;
int velIzq=velIniIzq;
int velDer=velIniDer;

if(EncoderPosIzq<EncoderPosDer){
    velIzq=velIzq+10;
}
if(EncoderPosIzq>EncoderPosDer){
    velDer=velDer+10;
}
//Se giran las ruedas a determinada velocidad
analogWrite(Velocidad_DER, velDer);
analogWrite(Velocidad_IZQ, velIzq);
}
```

Fragmento de código 14: Función para moverse hacia adelante

- `atras()`: se encarga del avance hacia atrás. En esta función se incluye el control PID también.

```
void atras()//MOVIMIENTO HACIA ATRAS
{
    //Se establecen las direcciones de los motores
    digitalWrite(Direccion_DER, LOW);
    digitalWrite(Direccion_IZQ, LOW);

    //Se deshabilita la parada de los motores
    digitalWrite(Parada_DER, LOW);
    digitalWrite(Parada_IZQ, LOW);

    //PID
    int velIniIzq=70;
    int velIniDer=70;
    int velIzq=velIniIzq;
    int velDer=velIniDer;

    if(EncoderPosIzq<EncoderPosDer){
        velIzq=velIzq+10;
    }
    if(EncoderPosIzq>EncoderPosDer){
        velDer=velDer+10;
    }
}
```

Fragmento de código 15: Función para moverse hacia atrás



- derecha(): giro a la derecha.

```
void derecha()
{
    int velIniIzq=70;
    int velIniDer=70;
    int velIzq=velIniIzq;
    int velDer=velIniDer;

    //Se establecen las direcciones de los motores
    digitalWrite(Direccion_DER, LOW);
    digitalWrite(Direccion_IZQ, HIGH);

    //Se deshabilita la parada de los motores
    digitalWrite(Parada_DER, LOW);
    digitalWrite(Parada_IZQ, LOW);

    //Se giran las ruedas a determinada velocidad
    analogWrite(Velocidad_DER, velDer);
    analogWrite(Velocidad_IZQ, velIzq);
}
```

Fragmento de código 16: Función para girar a la derecha

- izquierda(): giro a la izquierda.

```
void izquierda()
{
    int velIniIzq=70;
    int velIniDer=70;
    int velIzq=velIniIzq;
    int velDer=velIniDer;

    //Se establecen las direcciones de los motores
    digitalWrite(Direccion_DER, HIGH);
    digitalWrite(Direccion_IZQ, LOW);

    //Se deshabilita la parada de los motores
    digitalWrite(Parada_DER, LOW);
    digitalWrite(Parada_IZQ, LOW);

    //Se giran las ruedas a determinada velocidad
    analogWrite(Velocidad_DER, velDer);
    analogWrite(Velocidad_IZQ, velIzq);
}
```

Fragmento de código 17: Función para girar a la izquierda

- stop(): parada del robot.

```
boolean stop(){  
    //Se paran los motores  
    digitalWrite(Parada_DER, HIGH);  
    digitalWrite(Parada_IZQ, HIGH);  
    return true;  
}
```

Fragmento de código 18: Función para detener el movimiento

- cmToticks: realiza la conversión de centímetros a pulsos de onda (encoders).

```
//Transforma de centímetros a numero de ticks (pulsos)  
int cmToticks(int num){  
    float diametro=6.6; //centímetros  
    int ticks_vuelta=10;  
    float pi=3.14;  
  
    float cm_por_tick = (diametro*pi)/ticks_vuelta;  
    int res = (int) (num/cm_por_tick + 0.5);  
    //El +0,5 es para redondear, ya que el cast de Arduino trunca  
    return res;  
}
```

Fragmento de código 19: Función que transforma centímetros a pulsos

Además existen 2 funciones especiales, responsables de la comunicación entre el *VladBot* y el usuario:

- `cmdProcess()`: procesa los comandos introducidos por el usuario. Esta función se encarga de realizar una acción u otra, dependiendo del tipo de movimiento (y la cantidad de éste) que el usuario haya introducido.

```
void cmdProcess(int num_str, int val) {
    int ticks = cmToticks(val);
    //En New_ticks se almacena la nueva distancia acumulada que quiere
    alcanzarse
    int New_ticks = ticks + EncoderPosIzq;

    switch(num_str){

        //case "go"
        case 0:
            Serial.print( "Moviendome hacia adelante " );
            Serial.print(val);
            Serial.println( " centimetros.." );
            while((EncoderPosIzq)<New_ticks){
                adelante();
            }

            if(stop()){
                //Se envia mensaje de terminacion
                mensajeFin();
            }
            Serial.flush();
            delay(1000);
            break;
    }
}
```

Fragmento de código 20: Función que procesa los comandos (I)

```

//case "back"
case 1:
    Serial.print( "Moviendome hacia atras " );
    Serial.print(val);
    Serial.println( " centimetros..." );
    while((EncoderPosIzq)<New_ticks){
        atras();
    }

    if(stop()){
        //Se envia mensaje de terminacion
        mensajeFin();
    }
    Serial.flush();
    delay(1000);
    break;

//case "left"
case 2:
    Serial.print( "Girando hacia la izquierda " );
    Serial.print(val);
    Serial.println( " grados..." );
    if (val>360){
        Serial.println( "Introduzca un valor menor de 360 grados" );
        break;
    }

    //Estadisticamente: cada 2 ticks son 15º, asi se calcula el
    numero de veces que hay que girar 15º
    y se redondea, por eso el 0,5
    ticks=((val/15) +0.5);
    New_ticks = ticks + EncoderPosIzq;

    while((EncoderPosIzq)<New_ticks){
        izquierda();
    }

```

**Fragmento de código 21: Función que procesa los comandos (II)**

```

    if(stop()){
        //Se envia mensaje de terminacion
        mensajeFin();
    }
    //Se ponen los encoders a 0
    EncoderPosIzq=0;
    EncoderPosDer=0;
    Serial.flush();
    delay(1000);
    break;

//case "right"
case 3:
    Serial.print( "Girando hacia la derecha " );
    Serial.print(val);
    Serial.println( " grados..." );
    if (val>360){
        Serial.println( "Introduzca un valor menor de 360 grados" );
        break;
    }
    ticks=((val/15) +0.5);
    New_ticks = ticks + EncoderPosIzq;

    while(EncoderPosIzq<New_ticks){
        derecha();
    }

    if(stop()){
        //Se envia mensaje de terminacion
        mensajeFin();
    }
    //Se ponen los encoders a 0
    EncoderPosIzq=0;
    EncoderPosDer=0;
    Serial.flush();
    delay(1000);
    break;
}
}

```

Fragmento de código 22: Fragmento de código 23: Función que procesa los comandos (III)

- `mensajeFin()`: una vez se haya completado la orden con éxito, el robot envía un mensaje de confirmación al usuario.

```
void mensajeFin(){
    aJsonObject* fin=aJson.createObject();
    aJson.addStringToObject(fin, "END" , "OK" );

    //Se manda por el puerto serie y se imprime
    aJsonObject* imp;
    imp = aJson.getObjectItem(fin, "END" );
    Serial.println(imp->valuestring);
    aJson.deleteItem(fin);
}
```

Fragmento de código 24: Función que envía mensaje de terminación

Por último, en la función `loop()` se esta constantemente leyendo del puerto serie (hasta que se introduce un retorno de carro), almacenando en un búfer los datos que llegan por este puerto. El micro-controlador parsea el mensaje almacenado (objeto `aJson`) para extraer los parámetros necesarios (movimiento y cantidad) que necesita la función `cmdProcess()`.

```
void loop(){

    char msg[BUFLen];
    int c= 0;
    int i = 0;
    //Bucle para inicializar el buffer donde se almacenan los
    datos
    for(i = 0; i < BUFLen; i++){
        msg[i]=0;
    }

    i = 0; //Se asegura que i sea 0
```

Fragmento de código 25: Función `loop()` (I)



```

//Este bucle almacena los datos en el buffer (msg)
while(true) {
    while (Serial.available() < 1) {
        delay(1);
    }
    c = Serial.read();
    if(c == 10) break; //Si c es un retorno de carro '/r' (10 en
                        Ascii), sale del bucle
    msg[i++] = c; //Se incrementa la posicion de buffer
} //Fin while (true)
//Serial.println(msg);
aJsonObject* root = aJson.parse(msg); //Mensaje del tipo:
                                     {"go":200}

aJsonObject *pval;
char *pstr;

for(int j = 0; j < 4; j++) { //Lo que hace este bucle es ir
                            "preguntando" si se ha introducido
                            alguna de las palabras clave para
                            indicar la direccion de movimiento.

    if(j == 0) {
        pstr = "go" ;
    } else if(j == 1) {
        pstr = "back" ;
    } else if(j == 2) {
        pstr = "left" ;
    } else if(j == 3) {
        pstr = "right" ;
    }
    pval = aJson.getObjectItem(root, pstr);
    if(pval) {
        //Para depuración:
        //Serial.print(pstr);
        //Serial.println(pval->valueint);
        cmdProcess(j, pval->valueint); //Procesamiento de comandos, se
                                     le pasa el valor de "j",
                                     correspondiente al movimiento

        break;
    }
}
aJson.deleteItem(root); //Al final se elimina el objeto
} //FIN LOOP

```

Fragmento de código 26: Función loop() (II)

## 4. PRUEBAS REALIZADAS

En la sección 1.4. se ha introducido que las pruebas realizadas ocupan dos grandes bloques: el posicionamiento y las comunicaciones. En este apartado se describen las fases de pruebas realizadas en *VladBot* de forma cronológica para una mejor comprensión del lector. Así en este punto, las pruebas se agrupan en dos bloques:

- Pruebas de código.
- Pruebas de precisión.

### 4.1. PRUEBAS DE CÓDIGO

Con las pruebas de código se ha comprobado que todos los elementos funcionan correctamente. Se han realizado pruebas con pequeños códigos para probar cada elemento por separado. Las pruebas realizadas han sido las siguientes.

- Prueba Arduino Uno: se ha cargado en la placa un código de ejemplo ("*blink*") que enciende y apaga un LED. Con esta prueba se ha comprobado el correcto funcionamiento de la placa. Además, se ha comprobado que todo el *software* funciona correctamente: instalación y configuración del IDE de Arduino, controladores USB, cable USB, etc. Es importante mencionar que el IDE debe estar correctamente configurado para trabajar con la placa Arduino Uno:

Herramientas→Tarjeta→Arduino Uno

De la misma forma, es necesario establecer el puerto de comunicación:

Herramientas→Puerto Serial→ /dev/tty.usbmodemfa131

Este puerto es necesario seleccionarlo cuando se desee establecer la comunicación mediante el cable USB, indispensable para la carga del código. Un aspecto importante es que la conexión del USB ha de hacerse primero en *VladBot* y posteriormente al ordenador, si no, el ordenador no reconoce la placa.

- Prueba Motores: se ha introducido un pequeño código a la placa que probase todos los tipos de movimientos que ofrece la plataforma. En esta fase de pruebas se han establecido la rueda derecha y la izquierda, definiendo la parte frontal del robot. También se han decidido los dos tipos de movimientos que ejecuta el robot.
- Prueba Bluetooth: en esta prueba se han comprobado las conexiones del módem Bluetooth a la placa, además de la configuración *software*. En primer lugar se ha enlazado, mediante el asistente Bluetooth del sistema operativo, con el ordenador de pruebas. A continuación se ha cargado un sencillo código para probar que la transmisión por Bluetooth funciona correctamente. Este código ha consistido en la transmisión de sencillas instrucciones y mensajes de depuración. En esta situación, la comunicación se ha realizado mediante el módulo Bluetooth, así que en la configuración del IDE se ha establecido el puerto correspondiente a éste. Ésta es la configuración necesaria para trabajar con *VladBot*.

Herramientas→Puerto Serial→/dev/tty.VladBot-SPP

- Prueba Interrupciones: se ha llevado a cabo un test a los *encoders*. En primer lugar se ha comprobado, mediante un osciloscopio, que efectivamente se generan pulsos de onda cuadrada. Una vez comprobados e integrados al sistema, se ha ejecutado un ligero código en la placa que comprabase que se capturaban estas interrupciones.
- Prueba *Encoders*: en esta fase de pruebas se ha establecido la conversión entre los centímetros que introduce el usuario y los pulsos de onda que entiende el robot. Se han realizado diversas pruebas, tanto para el ajuste del movimiento de avance como para el de giro.
- Prueba PID: se ha añadido al código de pruebas de los *encoders* el control PID. Esta pequeña modificación se ha ajustado hasta que se han conseguido los resultados deseados.

- Prueba Json: esta prueba ha consistido en una modificación del código de pruebas de Bluetooth para la transmisión de mensajes con este protocolo. En esta fase se ha estudiado su comportamiento y la forma de implementarlo en el código definitivo del robot.
- Prueba Final: en esta prueba se ha definido el código definitivo (*VladBot 1.0*). Se han integrado todas aquellas partes de código que han pasado las pruebas satisfactoriamente. Este es el código que se presenta en el anexo I.

## 4.2. PRUEBAS DE PRECISIÓN

En las pruebas de precisión se ha determinado el error que introduce el sistema de posicionamiento. Asumiendo este error, se ha definido la precisión del sistema. En concreto se han realizado dos tipos de pruebas:

- Prueba de giro

En este tipo de pruebas se ha determinado el ángulo mínimo de giro que ofrece una precisión satisfactoria. Se ha hecho girar el robot en incrementos de 15 grados, hasta completar una vuelta completa. Se ha escogido este número de grados ya que es el correspondiente a dos pulsos de onda cuadrada. En una primera aproximación, se ha probado con un sólo pulso de onda, que a primera vista ofrece más precisión. Pero se ha comprobado que su precisión es muy inestable. Sin embargo, al aumentar a dos pulsos, el sistema arroja frecuentemente el mismo resultado.

En las tablas 6, 7 y 8 se muestran los tests realizados. Con la ayuda de una brújula digital, la cual se ha montado en la plataforma, se ha podido establecer este conjunto de datos.

Tabla 6: Prueba de giro. Test 1

Grados	Incremento
9	9
24	15
38	14
55	17
62	7
72	10
95	23
113	18
134	21
146	12
161	15
169	8
185	16
195	10
219	24
239	20
253	14
268	15
289	21
304	15
317	13
335	18
349	14
360	11
PROMEDIO	15
DESVEST	4,6

Tabla 7: Prueba de giro. Test 2

Grados	Incremento
12	12
23	11
39	16
54	15
63	9
77	14
93	16
110	17
124	14
142	18
160	18
170	10
185	15
197	12
215	18
230	15
248	18
260	12
280	20
295	15
313	18
323	10
343	20
360	17
PROMEDIO	15
DESVEST	3,2

Tabla 8: Prueba de giro. Test 3

Grados	Incremento
15	15
29	14
38	9
49	11
66	17
78	12
96	18
111	15
125	14
143	18
156	13
168	12
185	17
196	11
214	18
233	19
245	12
257	12
270	13
290	20
306	16
324	18
340	16
360	20
PROMEDIO	15
DESVEST	3,1

Se observa que el promedio es siempre 15 grados, con una desviación estándar media de 3,7. Esta precisión es suficiente para el trabajo que va a realizar el robot. Ofrece la posibilidad de cambiar de sentido, y cambios de dirección de 15 grados. Esto le otorga a la plataforma la capacidad de definir rutas e itinerarios con suficiente precisión.



- Prueba de avance

Este banco de pruebas se ha realizado para comprobar la fiabilidad del posicionamiento en una línea recta. Con el sistema PID integrado en el código y el soporte del micrófono y el micrófono cargado en la plataforma, se ha hecho avanzar en línea recta a *VladBot* y se ha observado que lo hace de manera satisfactoria, de forma completamente recta en distancias no superiores a las decenas de metros, más que suficiente para abarcar una sala. *VladBot* no está diseñado para recorrer largas distancias, si no que su trabajo consiste en mover en intervalos de unos pocos centímetros el micrófono de medida. Como se ha establecido en el apartado 3.2., los *encoders* utilizados limitan esta precisión a 2 cm, este es el error máximo que se ha obtenido. Por tanto, el robot puede resolver movimientos de avance de un mínimo de 2 centímetros, y avanzar completamente en línea recta unos pocos metros con un error que no supera los 2 centímetros en ninguna ocasión.

## 5. PRESUPUESTO

En esta sección se muestra el presupuesto detallado. Debido a que el proyecto ha sido un producto sin precedentes, el trabajo de investigación ha aumentado considerablemente su precio. Pero se puede observar que el sistema por sí solo cumple uno de los objetivos principales: bajo precio. De esta manera el desarrollo cada nuevo *ValdBot* tendrá un precio de 181,24 euros.

Tabla 9: Presupuesto *VladBot*

COMPONENTE	EUROS
Kit DFRobot 2WD Mobile Platform	30,74
Soporte de micrófono	10,00
Arduino Uno + cable USB	29,90
Controlador de motores	28,61
Encoders	9,75
Modem Bluetooth BlueSMIRF Gold	72,24
TOTAL	181,24

Tabla 10: Presupuesto diseño

HORAS	EUROS/HORA
123	31,02
TOTAL	3815,46

Tabla 11: Presupuesto total

TOTAL VLADBOT	3996,70 EUROS
---------------	---------------

## 6. LÍNEAS FUTURAS DE INVESTIGACIÓN

Debido a que el desarrollo de *VladBot* ha sido realizado con una plataforma abierta, se abre un abanico de posibilidades de rediseño del proyecto. Se han abierto a priori, cuatro vías de investigación que se pueden desarrollar en el robot para aumentar sus funcionalidades. Estas mejoras pueden potenciar considerablemente su manejo así como aumentar sus aplicaciones actuales.

- Programación vía Bluetooth

Como se ha descrito en el apartado 3.1, actualmente resulta imposible cargar un código por Bluetooth, sino que ha de hacerse por el cable USB. Consiguiendo programar la placa vía Bluetooth se podrían realizar cambios en el código del robot inalámbricamente. Una ventaja bastante útil.

- Integración con otro *software*

Otra línea de investigación útil sería la posibilidad de integrar la comunicación del robot con otro tipo de *software* (por ejemplo, Processing). Se podrían programar rutas e itinerarios con este otro programa externo. Gracias a el protocolo de comunicación, Json, esta implementación es posible. De hecho, ha sido uno de las razones por la que el diseño de las comunicaciones se ha desarrollado con este protocolo. Podría resultar muy útil al usuario establecer una ruta determinada por donde ir recogiendo los datos de las medidas automáticamente.

- Sistema de medida inalámbrico

En el diseño actual el robot carga con el micrófono de medida, con su cable y con su pre-amplificador. Esto supone mucho gasto de energía a la plataforma, y en consecuencia un cambio continuo de la batería, acortando sus horas de uso. Por esta razón resulta razonable la integración de un sistema de medida inalámbrico.

Además, aumentaría considerablemente la capacidad de automatización de las medidas, ya que, desarrollando anteriormente la integración del sistema actual con otro *software*, se podría programar la adquisición de datos. Esto aportaría una completa automatización en el proceso de medidas. Como se ha introducido al principio de este documento, *VladBot* es un prototipo de sistema de transporte, y no se preocupa de realizar las medidas. Pero esto no resta interés en que en versiones futuras esto se pueda implementar en la plataforma.

- Reducción/filtración de ruido

Actualmente, el ruido que emiten los motores de corriente continua al moverse resulta molesto para realizar medidas acústicas. Por esta razón, actualmente, el sistema no puede recoger datos mientras está realizando un movimiento, sino que se ha de esperar a que la plataforma se mueva al punto deseado para poder realizar la medida. Reduciendo este ruido, se añadiría una nueva aplicación al *VladBot*: recoger datos en movimiento. También se podría solventar este inconveniente detectando y caracterizando este ruido característico, para su posterior filtración.

## 7. CONCLUSIONES

*VladBot* cumple con éxito el objetivo para el cual ha sido diseñado. Tiene la capacidad de posicionar un micrófono de medida en tres dimensiones en cualquier punto de una sala, con una precisión satisfactoria. Esta precisión es el error máximo que se puede cometer a la hora de posicionar la plataforma, y como se ha expuesto en capítulos anteriores, dicho error no supera los 2 centímetros. Dado que *VladBot* tiene una aplicación directa en el campo de la acústica, esta precisión es más que suficiente, ya que la longitud de las ondas del espectro audible tienen una magnitud comparable a dicho error. Desde los 20 Hz (17 metros de longitud de onda) que comienza el espectro, hasta los 20 kHz (1,7 cm de longitud de onda), el sistema ofrece un error que hace factible su utilización. A favor de *VladBot*, en la mayoría de los ensayos acústicos, el estudio del espectro audible llega hasta los 8 kHz (4,25 cm de longitud de onda), lo que hace que este error adquiera el adjetivo de satisfactorio.

Además, ha sido desarrollado con tecnologías de bajo coste (181,24 euros de presupuesto) consiguiendo que su diseño le aporte versatilidad, y esto es, cualquier usuario puede desarrollar una nueva aplicación para el robot.

La idea de automatizar medidas acústicas con una plataforma robótica móvil se hace factible. Este prototipo, al implementar tecnologías de bajo coste, ofrece una precisión satisfactoria, pero con un mayor presupuesto podría llegar a tener una precisión milimétrica. Está condicionado a trabajar sobre una superficie lisa y sin obstáculos, pero al igual que ocurre con su precisión, son las tecnologías de bajo coste las que imponen este condicionamiento. Con un sistema de posicionamiento externo, y sensores para detectar obstáculos se puede solucionar este problema.

Por otro lado, la comunicación Bluetooth es uno de los puntos fuertes del sistema, ya que ofrece una comunicación fiable a través de un protocolo de comunicación (JSON). Permite la interacción con cualquier dispositivo que integre Bluetooth (PC, Smartphone, Tablet, etc), y esto, sumado a la plataforma con la que se ha diseñado (Arduino) y la posibilidad de rediseño (reprogramación del código), abre un extenso abanico de posibilidades. Además la distancia de separación entre los terminales (*VladBot* y ordenador) es superior a los 100 metros, haciendo posible

que se pueda establecer una comunicación con el robot desde una habitación contigua.

Es una plataforma ligera, de un tamaño manejable y modular, es decir, se pueden desmontar sus diversos componentes para que sea más sencillo su transporte y no se produzcan daños en el robot. Su interruptor de encendido y la pila que lo alimenta son de fácil acceso, no siendo necesario ninguna herramienta adicional para encenderlo o cambiar la batería.

Por estas razones, el primer prototipo de *VladBot* cumple las expectativas planteadas en los objetivos: un sistema de transporte para un micrófono de medida capaz de establecer una comunicación fiable con el usuario, todo ello implementado con tecnologías de bajo coste. Además su diseño ha sido orientado para tener la capacidad de seguir evolucionando y alcanzar nuevas funcionalidades.



Figura 68: *VladBot* con micrófono

## 8. REFERENCIAS

- [1] I. Cortés de la Vega, *Robot autónomo como nodo móvil de redes de sensores*, Escuela Politécnica Superior Universidad Autónoma de Madrid, 2011
- [2] J. Boresteing, H.R. Everett, L. Feng, *Where am I? Sensors and Methods for Mobile Robot Positioning*, University of Michigan, 1996.
- [3] G. McComb, *The Robot Builder's Bonanza*, McGraw-Hill, 2001.
- [4] Arduino, <http://arduino.cc> [consulta: 29 de agosto de 2013].
- [5] S. Elvira Díaz, *Redes de sensores inalámbricas aplicadas a robótica corporativa*, Escuela Politécnica Superior Universidad Autónoma de Madrid, 2009.
- [6] DFRobot, <http://www.dfrobot.com> [consulta: 29 de agosto de 2013].
- [7] XBee, <http://www.xbee.cl> [consulta: 29 de agosto de 2013].
- [8] Bluetooth, <http://www.bluetooth.com>, [consulta: 29 de agosto de 2013].
- [9] JSON, <http://www.json.org>, [consulta: 29 de agosto de 2013].



## 9. ANEXO I

### 9.1. CÓDIGO

```
#include <PinChangeInt.h>

#include <stdio.h>

#include <stdlib.h>

#include <aJSON.h>

//Se establecen los pines

#define ENCODER_DER 16//interrupcion pin analogico A2

#define ENCODER_IZQ 17//interrupcion pin analogico A3

volatile long EncoderPosDer = 0;

volatile long EncoderPosIzq = 0;

//Volatile: siempre que su valor pueda ser modificado por algo más
allá de la sección del código en el que aparece, por ejemplo
interrupciones

#define BUFLLEN 32 //Se define el tamaño del buffer en el que se
                        almacenan los datos de entrada
```

```

int Direccion_DER = 12; //Direccion del motor derecho en el pin 12

int Direccion_IZQ = 13; //La del izquierdo en el 13


int Velocidad_DER = 3;

int Velocidad_IZQ = 11;


int Parada_DER = 9;

int Parada_IZQ = 8;


void setup(){

    //Se establecen las salidas

    pinMode(Direccion_DER, OUTPUT);

    pinMode(Direccion_IZQ, OUTPUT);


    pinMode(Velocidad_DER, OUTPUT);

    pinMode(Velocidad_IZQ, OUTPUT);


    pinMode(Parada_DER, OUTPUT);

    pinMode(Parada_IZQ, OUTPUT);


    //Se inicializa el puerto serial

    Serial.begin(9600);


    //Interrupciones

```

```

PCintPort::attachInterrupt(ENCODER_IZQ, &contarIzq, RISING);

PCintPort::attachInterrupt(ENCODER_DER, &contarDer, RISING);


//Inicializacion de las variables de los encoders

int EncoderPosIzq=0;

int EncoderPosDer=0;

int ticks=0;


}


//Funcion que se ejecuta cuando ocurre una interrupcion.
//Se incrementa la cuenta del encoder

void contarIzq(){

    EncoderPosIzq++;

}

void contarDer(){

    EncoderPosDer++;

}

/*void imprime(int ticks){

```

```

    Serial.println("-----NUEVA ITERACION-----
    -----");

    //Muestra la posicion de los encoders

    Serial.print("Numero de cambios encoder IZQUIERDO: ");

    Serial.println(EncoderPosIzq);


    Serial.print("Numero de cambios encoder DERECHO: ");

    Serial.println(EncoderPosDer);


    //Pulsos, se contabilizan los flancos de subida

    Serial.print("TICKS: ");

    Serial.println(ticks);
}*/ /*Para depuracion*/


void adelante ()//MOVIMIENTO HACIA ADELANTE
{

    //Se establecen las direcciones de los motores

    digitalWrite(Direccion_DER, HIGH);

    digitalWrite(Direccion_IZQ, HIGH);


    //Se deshabilita la parada de los motores

    digitalWrite(Parada_DER, LOW);

    digitalWrite(Parada_IZQ, LOW);

```

```

//PID

int velIniIzq=70;

int velIniDer=70;

int velIzq=velIniIzq;

int velDer=velIniDer;


if(EncoderPosIzq<EncoderPosDer){

    velIzq=velIzq+10;

}

if(EncoderPosIzq>EncoderPosDer){

    velDer=velDer+10;

}


//Se giran las ruedas a determinada velocidad

analogWrite(Velocidad_DER, velDer);

analogWrite(Velocidad_IZQ, velIzq);

}


void atras()//MOVIMIENTO HACIA ATRAS

{

    //Se establecen las direcciones de los motores

    digitalWrite(Direccion_DER, LOW);

    digitalWrite(Direccion_IZQ, LOW);

```

```

//Se deshabilita la parada de los motores

digitalWrite(Parada_DER, LOW);

digitalWrite(Parada_IZQ, LOW);


//PID

int velIniIzq=70;

int velIniDer=70;

int velIzq=velIniIzq;

int velDer=velIniDer;


if(EncoderPosIzq<EncoderPosDer){

    velIzq=velIzq+10;

}

if(EncoderPosIzq>EncoderPosDer){

    velDer=velDer+10;

}


//Se giran las ruedas a determinada velocidad

analogWrite(Velocidad_DER, velDer);

analogWrite(Velocidad_IZQ, velIzq);

}

void derecha()

```

```

{

    int velIniIzq=70;

    int velIniDer=70;

    int velIzq=velIniIzq;

    int velDer=velIniDer;


    //Se establecen las direcciones de los motores

    digitalWrite(Direccion_DER, LOW);

    digitalWrite(Direccion_IZQ, HIGH);


    //Se deshabilita la parada de los motores

    digitalWrite(Parada_DER, LOW);

    digitalWrite(Parada_IZQ, LOW);


    //Se giran las ruedas a determinada velocidad

    analogWrite(Velocidad_DER, velDer);

    analogWrite(Velocidad_IZQ, velIzq);

}


void izquierda()

{

    int velIniIzq=70;

    int velIniDer=70;

    int velIzq=velIniIzq;

```

```

int velDer=velIniDer;

//Se establecen las direcciones de los motores

digitalWrite(Direccion_DER, HIGH);

digitalWrite(Direccion_IZQ, LOW);

//Se deshabilita la parada de los motores

digitalWrite(Parada_DER, LOW);

digitalWrite(Parada_IZQ, LOW);

//Se giran las ruedas a determinada velocidad

analogWrite(Velocidad_DER, velDer);

analogWrite(Velocidad_IZQ, velIzq);

}

boolean stop(){

    //Se paran los motores

    digitalWrite(Parada_DER, HIGH);

    digitalWrite(Parada_IZQ, HIGH);

    return true;

}

//Transforma de centimetros a numero de ticks (pulsos)

```



```

int cmToticks(int num){

    float diametro=6.6; //centimetros

    int ticks_vuelta=10;

    float pi=3.14;


    float cm_por_tick = (diametro*pi)/ticks_vuelta;

    int res = (int) (num/cm_por_tick + 0.5);

    //El +0,5 es para redondear, ya que el cast de Arduino trunca

    return res;

}


void loop(){

    char msg[BUFLEN];

    int c= 0;

    int i = 0;


    //Bucle para inicializar el buffer donde se almacenan los datos

    for(i = 0; i < BUFLen; i++){

        msg[i]=0;

    }


    i = 0; //Se asegura que i sea 0

```

```

//Este bucle almacena los datos en el buffer (msg)

while(true) {

    while (Serial.available() < 1) {

        delay(1);

    }

    c = Serial.read();

    if(c == 10) break; //Si c es un retorno de carro '/r' (10 en
                        Ascii), sale del bucle

    msg[i++] = c; //Se incrementa la posicion de buffer
}

//Fin while (true)

//Serial.println(msg);

JsonObject* root = Json.parse(msg); //Mensaje del tipo:

                                {"go":200}


JsonObject *pval;

char *pstr;


for(int j = 0; j < 4; j++) { //Lo que hace este bucle es ir

                                "preguntando" si se ha introducido

                                alguna de las palabras clave para

                                indicar la direccion de movimiento.

                                Si no es ninguna, se sale del bucle, y

                                borra el objeto Json

    if(j == 0) {

```

```

        pstr = "go" ;
    } else if(j == 1) {
        pstr = "back" ;
    } else if(j == 2) {
        pstr = "left" ;
    } else if(j == 3) {
        pstr = "right" ;
    }

    pval = aJson.getObjectItem(root, pstr);
    if(pval) {
        //Para depuración:

        //Serial.print(pstr);

        //Serial.println(pval->valueint);

        cmdProcess(j, pval->valueint); //Procesamiento de comandos, se
                                     le pasa el valor de "j",
                                     correspondiente al movimiento

        break;
    }
}

aJson.deleteItem(root); //Al final se elimina el objeto
} //FIN LOOP

void cmdProcess(int num_str, int val) {
    int ticks = cmToticks(val);

```

```

//En New_ticks se almacena la nueva distancia acumulada que quiere
alcanzarse

int New_ticks = ticks + EncoderPosIzq;

switch(num_str){

    //case "go"

    case 0:

        Serial.print( "Moviendome hacia adelante " );

        Serial.print(val);

        Serial.println( " centimetros.." );

        while((EncoderPosIzq)<New_ticks){

            adelante();

        }

        if(stop()){

            //Se envia mensaje de terminacion

            mensajeFin();

        }

        Serial.flush();

        delay(1000);

        break;

    //case "back"

```

```

case 1:

    Serial.print( "Moviendome hacia atras " );

    Serial.print(val);

    Serial.println( " centimetros..." );

    while((EncoderPosIzq)<New_ticks){

        atras();

    }

    if(stop()){

        //Se envia mensaje de terminacion

        mensajeFin();

    }

    Serial.flush();

    delay(1000);

    break;

//case "left"

case 2:

    Serial.print( "Girando hacia la izquierda " );

    Serial.print(val);

    Serial.println( " grados..." );

    if (val>360){

        Serial.println( "Introduzca un valor menor de 360 grados" );

        break;

```

```

    }

    //Estadisticamente: cada 2 ticks son 15º, asi se calcula el
    //numero de veces que hay que girar 15º, y se redondea, por eso el 0,5
    ticks=((val/15) +0.5);

    New_ticks = ticks + EncoderPosIzq;

    while((EncoderPosIzq)<New_ticks){

        izquierda();

    }

    if(stop()){

        //Se envia mensaje de terminacion

        mensajeFin();

    }

    //Se ponen los encoders a 0

    EncoderPosIzq=0;

    EncoderPosDer=0;

    Serial.flush();

    delay(1000);

    break;

//case "right"

```

```

case 3:

    Serial.print( "Girando hacia la derecha " );

    Serial.print(val);

    Serial.println( " grados..." );

    if (val>360){

        Serial.println( "Introduzca un valor menor de 360 grados" );

        break;

    }

    ticks=((val/15) +0.5);

    New_ticks = ticks + EncoderPosIzq;

    while(EncoderPosIzq<New_ticks){

        derecha();

    }

    if(stop()){

        //Se envia mensaje de terminacion

        mensajeFin();

    }

    //Se ponen los encoders a 0

    EncoderPosIzq=0;

    EncoderPosDer=0;

    Serial.flush();

```

```

        delay(1000);

        break;

    } //FIN Switch

}

void mensajeFin(){

    aJsonObject* fin=aJson.createObject();

    aJson.addStringToObject(fin, "END" , "OK" );

    //Se manda por el puerto serie y se imprime

    aJsonObject* imp;

    imp = aJson.getObjectItem(fin, "END" );

    Serial.println(imp->valuestring);

    aJson.deleteItem(fin);

}

```



## 10. ANEXO II

### 10.1. MANUAL DE USUARIO PARA MAC OS X

En primer lugar es necesario instalar el software de Arduino. Para ello se va a la página:

<http://www.arduino.cc>

Y se hace *click* en la sección “Downloads”, donde se puede descargar la última versión del IDE de Arduino.

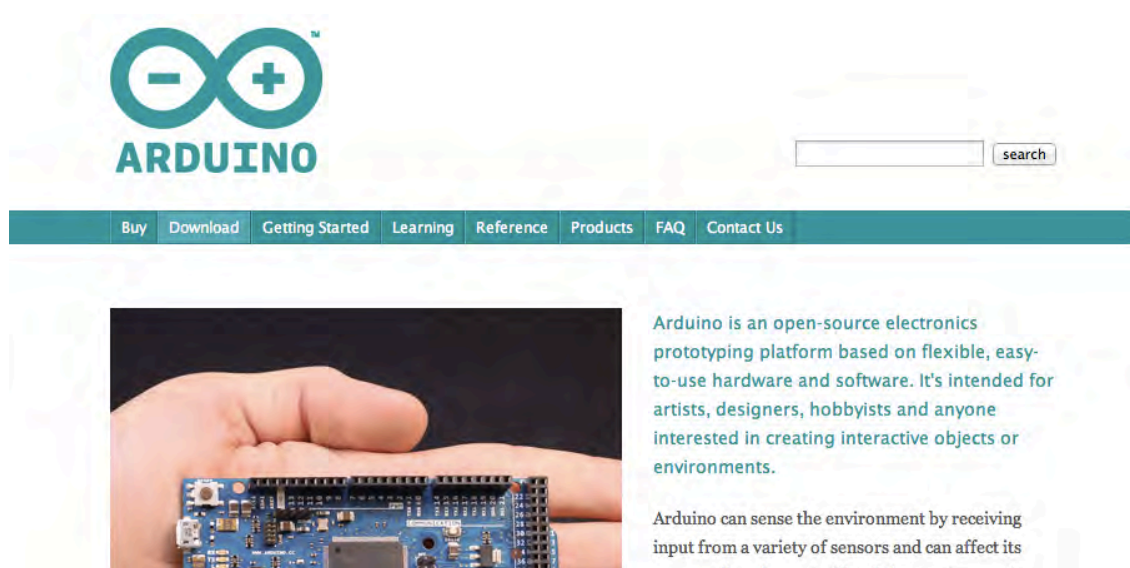


Figura 69: Sección de descargas

Se selecciona el sistema operativo, en este caso: Mac OS X. Si el usuario posee otro sistema operativo simplemente ha de escoger el suyo y seguir las instrucciones de instalación.

A continuación comenzará la descarga de la aplicación del IDE de Arduino. Se descomprime una vez descargado y se instala siguiendo las instrucciones del

asistente de instalación (depende del sistema operativo). Para Mac OS X basta con arrastrar la aplicación a la carpeta de “Applications”. Se abre el archivo descargado “.dmg” y aparecerá la pestaña que muestra la figura 70, donde se podrá realizar esta acción.

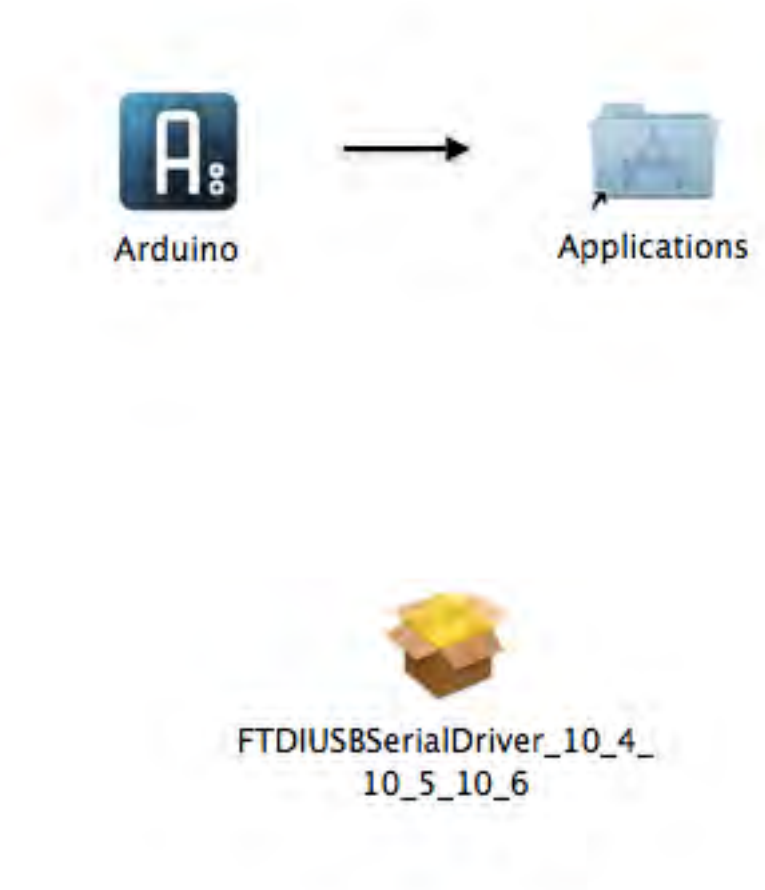
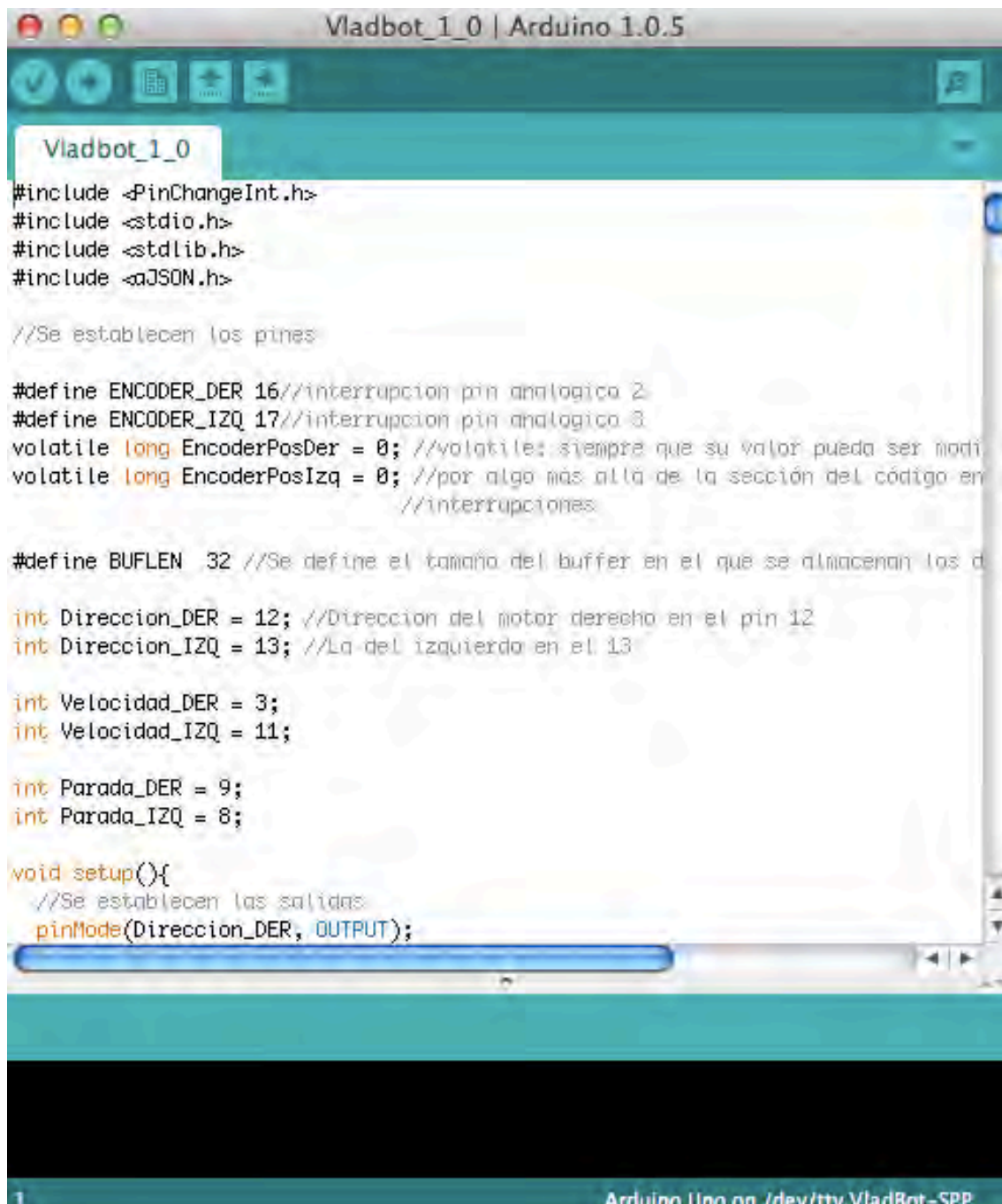


Figura 70: Instalación IDE de Arduino

También es necesario instalar los controladores FTDI USB para poder utilizar el cable USB. En el mismo archivo “.dmg” en el que se encuentra la aplicación se ubica un paquete de instalación para estos controladores (“FTDIUSBSerialDriver\_x”). Se abre y se sigue el asistente de instalación.

Ya con el IDE instalado, se abre el archivo “VladBot\_1\_0.ino” que es el programa que lleva instalado *VladBot*.



```
Vladbot_1_0 | Arduino 1.0.5

Vladbot_1_0

#include <PinChangeInt.h>
#include <stdio.h>
#include <stdlib.h>
#include <aJSON.h>

//Se establecen los pines

#define ENCODER_DER 16//interrupcion pin analogico 2
#define ENCODER_IZQ 17//interrupcion pin analogico 3
volatile long EncoderPosDer = 0; //volatile: siempre que su valor pueda ser modi
volatile long EncoderPosIzq = 0; //por algo más allá de la sección del código en
//interrupciones

#define BUFLen 32 //Se define el tamaño del buffer en el que se almacenan los d

int Direccion_DER = 12; //Direccion del motor derecho en el pin 12
int Direccion_IZQ = 13; //La del izquierdo en el 13

int Velocidad_DER = 3;
int Velocidad_IZQ = 11;

int Parada_DER = 9;
int Parada_IZQ = 8;

void setup(){
  //Se establecen las salidas
  pinMode(Direccion_DER, OUTPUT);
```

Figura 71: Sketch VladBot\_1\_0

No es necesario cargar el programa en la placa, ya que se encuentra precargado, pero en el caso de que el usuario decidiera hacer alguna modificación, se ha de proceder de la siguiente manera. En primer lugar, se desconecta el módem de Bluetooth, y a continuación se conecta el cable USB en *VladBot*. Después de esto ya se puede conectar el otro terminal del cable USB al puerto USB del ordenador. Es importante hacerlo en este orden, ya que si no podría darse algún problema en el *software*.

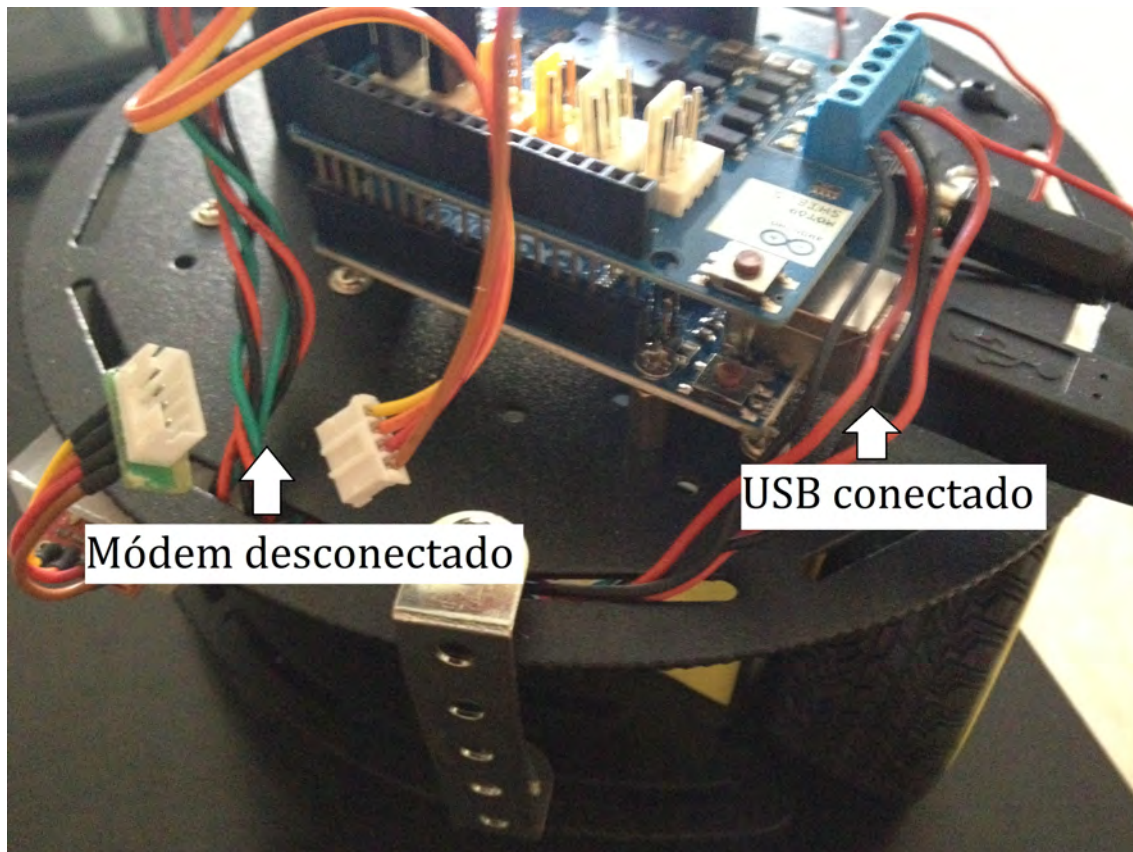


Figura 72: Conexiones previas antes de cargar un *sketch*

A continuación, seleccionamos la placa Arduino Uno (figura 73) y el puerto USB (figura 74).

Con esta configuración, se hace *click* en la flecha superior derecha, ver figura 75 ("Cargar"), y el nuevo programa comenzará a descargarse en la placa.



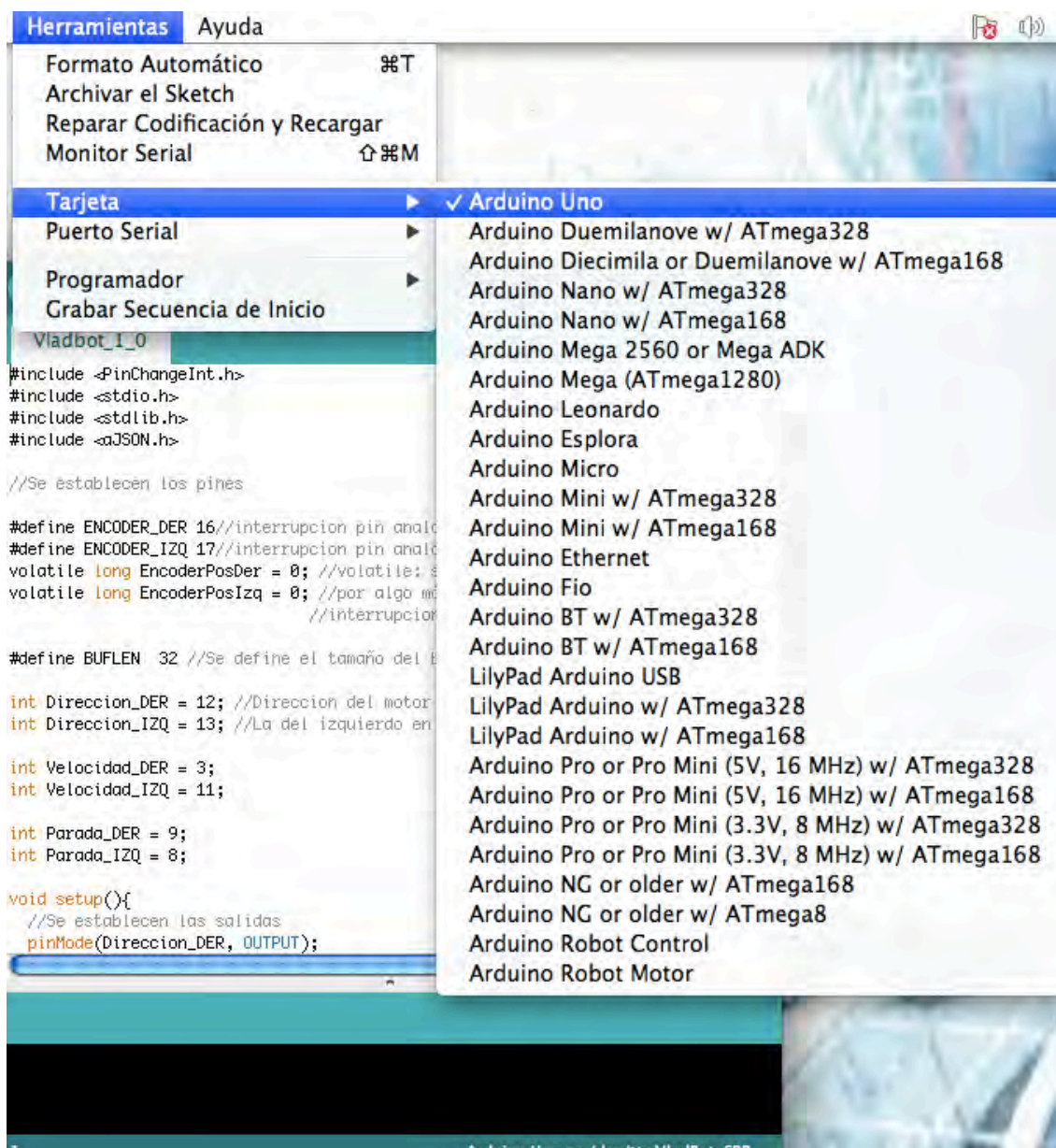


Figura 73: Selección de la tarjeta

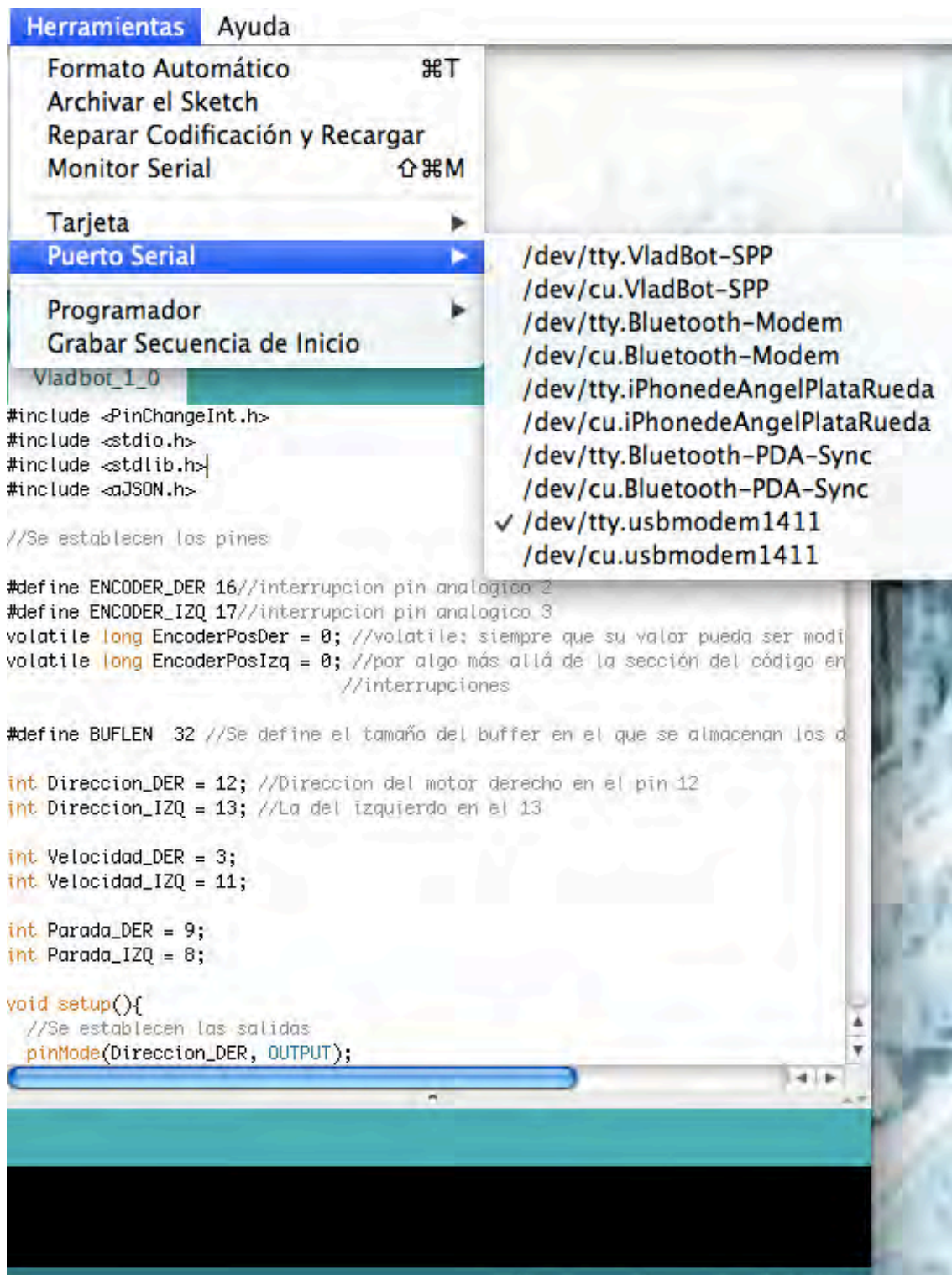


Figura 74: Selección del puerto

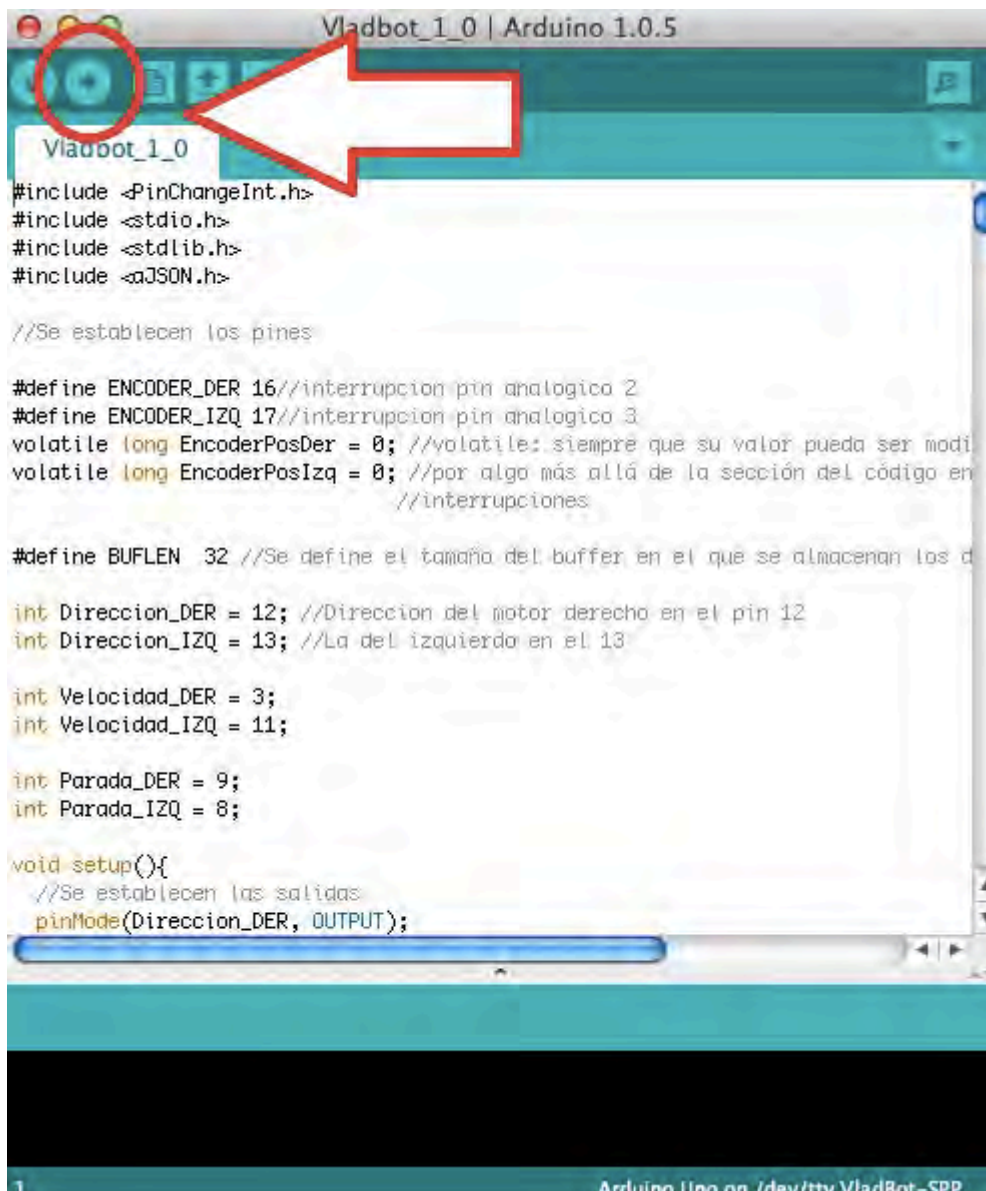


Figura 75: Cargar sketch

Ya con el programa cargado en la placa, se procede a enlazar el módem Bluetooth con el ordenador. Para ello, lo primero es reconectar el módem (si anteriormente se ha utilizado el puerto USB para cargar alguna modificación), si no, simplemente basta con poner el interruptor en posición de encendido y la luz roja del módem USB comenzará a parpadear.



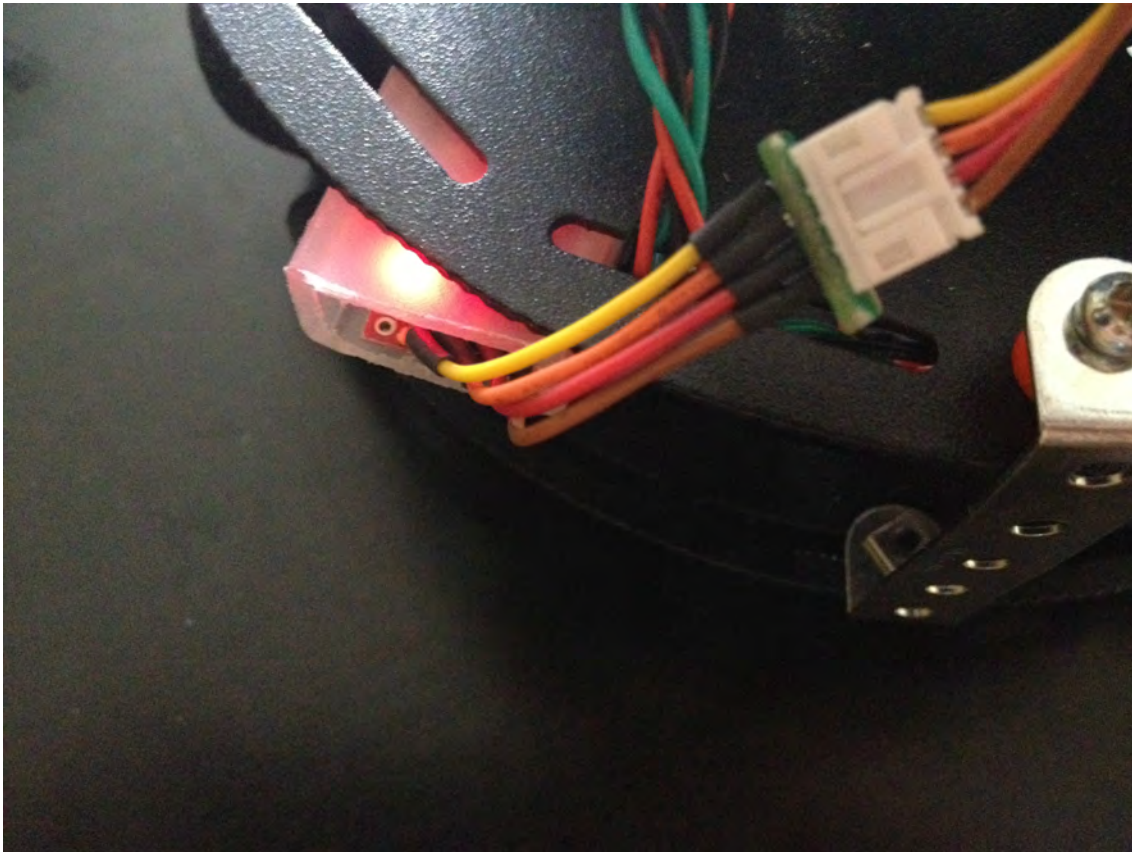


Figura 76: Luz roja del módem Bluetooth

Ahora hay que abrir el asistente de Bluetooth del ordenador en el que se vaya a utilizar. Se accede a configurar un nuevo dispositivo Bluetooth.

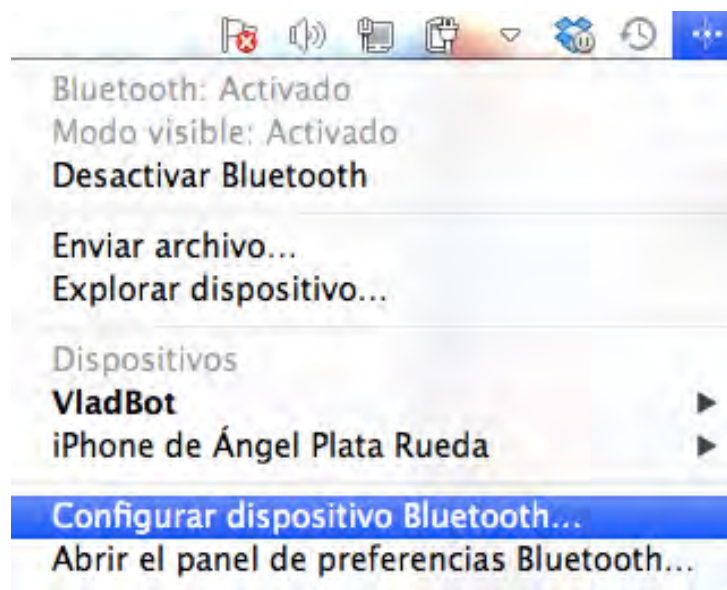


Figura 77: Asistente Bluetooth



Aparecerá el dispositivo con el nombre *VladBot*. Se selecciona y se da a continuar.

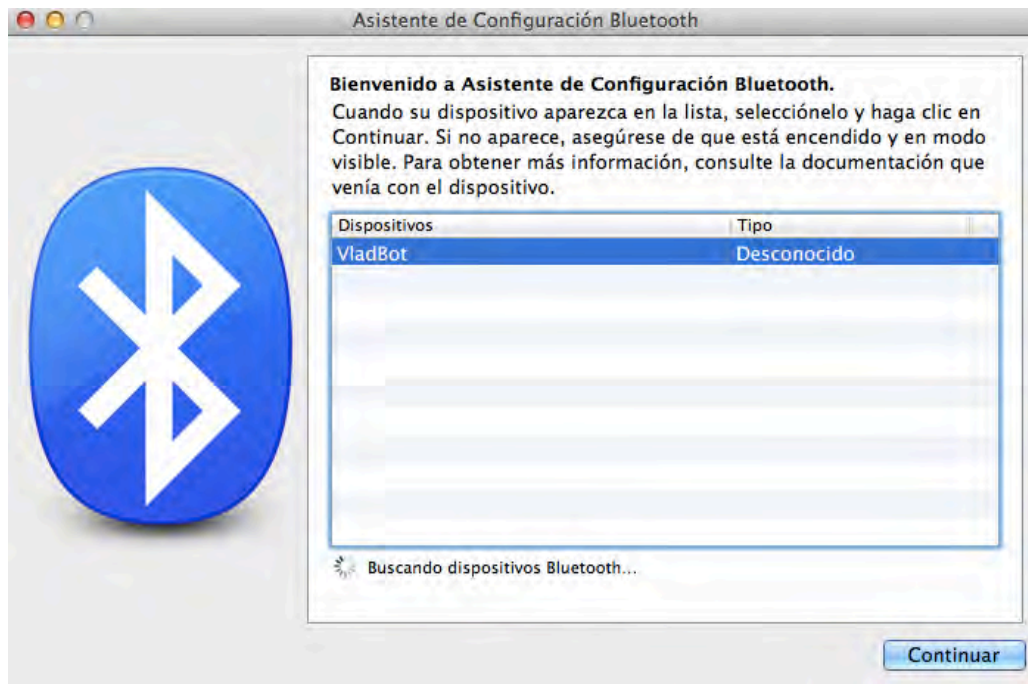


Figura 78: Primera ventana de la configuración del módem Bluetooth

Aparecerá la pantalla de la figura 79, se hace *click* en “Opciones de código...” y se selecciona “No utilizar un código con este dispositivo”.



Figura 79: Segunda ventana de la configuración del módem Bluetooth



Figura 80: Opciones de código

Se mostrará una pantalla de confirmación, donde se selecciona salir para finalizar el asistente. *VladBot* ya está enlazado con el ordenador.



Figura 81: Ventana de confirmación de enlace

De nuevo en el IDE de Arduino, se selecciona el nuevo puerto serial (figura 81).

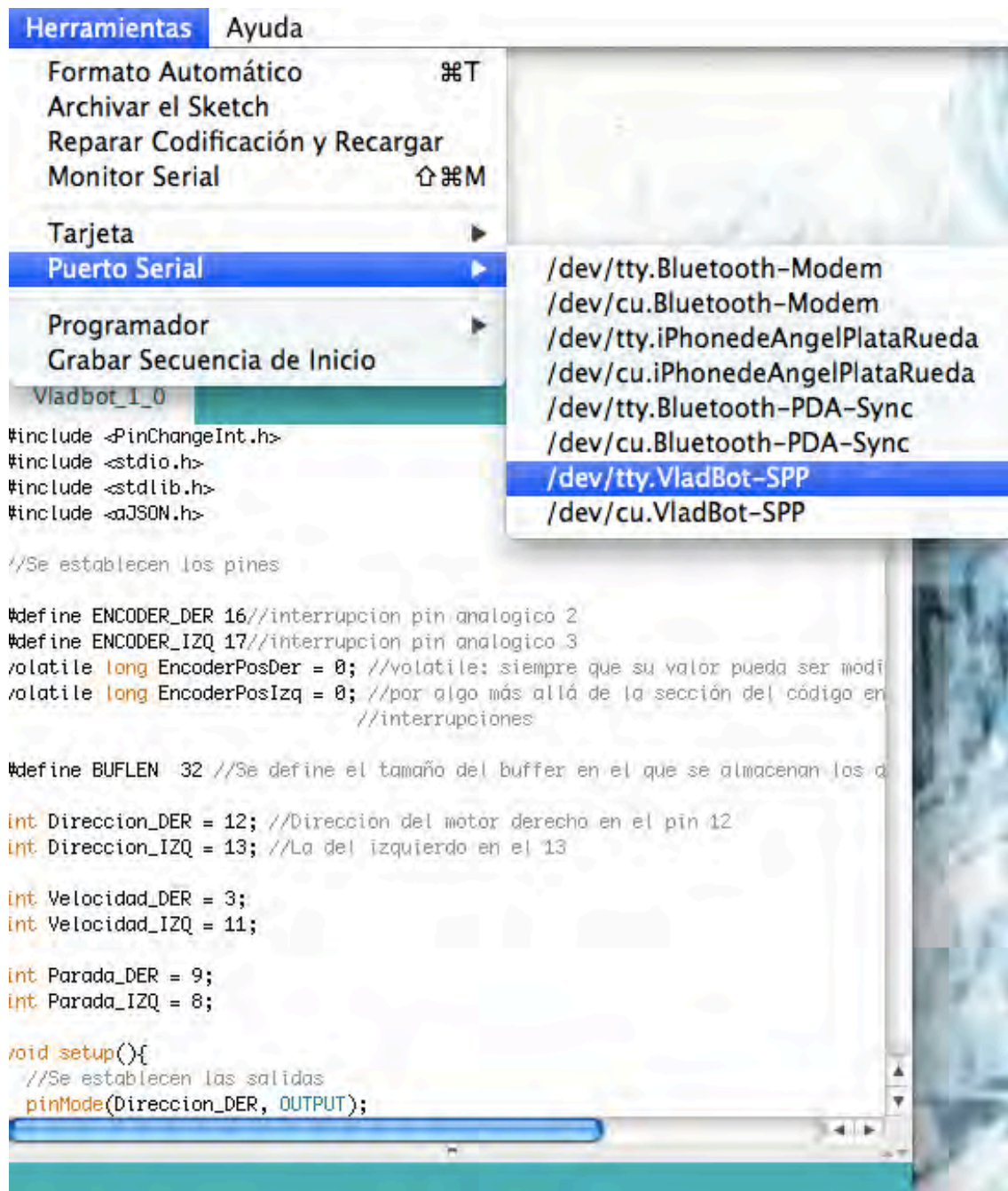


Figura 82: Selección del puerto serial (Bluetooth)

Se hace click en “Monitor Serial” (figura 83) y la luz del módem Bluetooth quedará de color verde (sin parpadeos, figura 84). Debe estar configurado a 9600 baudios y en “Nueva Línea” (figura 85). En este momento *VladBot* se encuentra listo para ser utilizado. Ya sólo hay que mandarle las instrucciones:

```
{“orden”:cantidad}
```

Siendo la órdenes que puede interpretar *VladBot*:

- “go”: avance hacia delante.
- “back”: avance hacia atrás.
- “left”: giro hacia la izquierda.
- “right”: giro hacia la derecha.

Y la cantidad en centímetros, si se trata de un movimiento hacia delante o hacia atrás, o en grados, si es un movimiento de giro. Ejemplo en la figura 86.

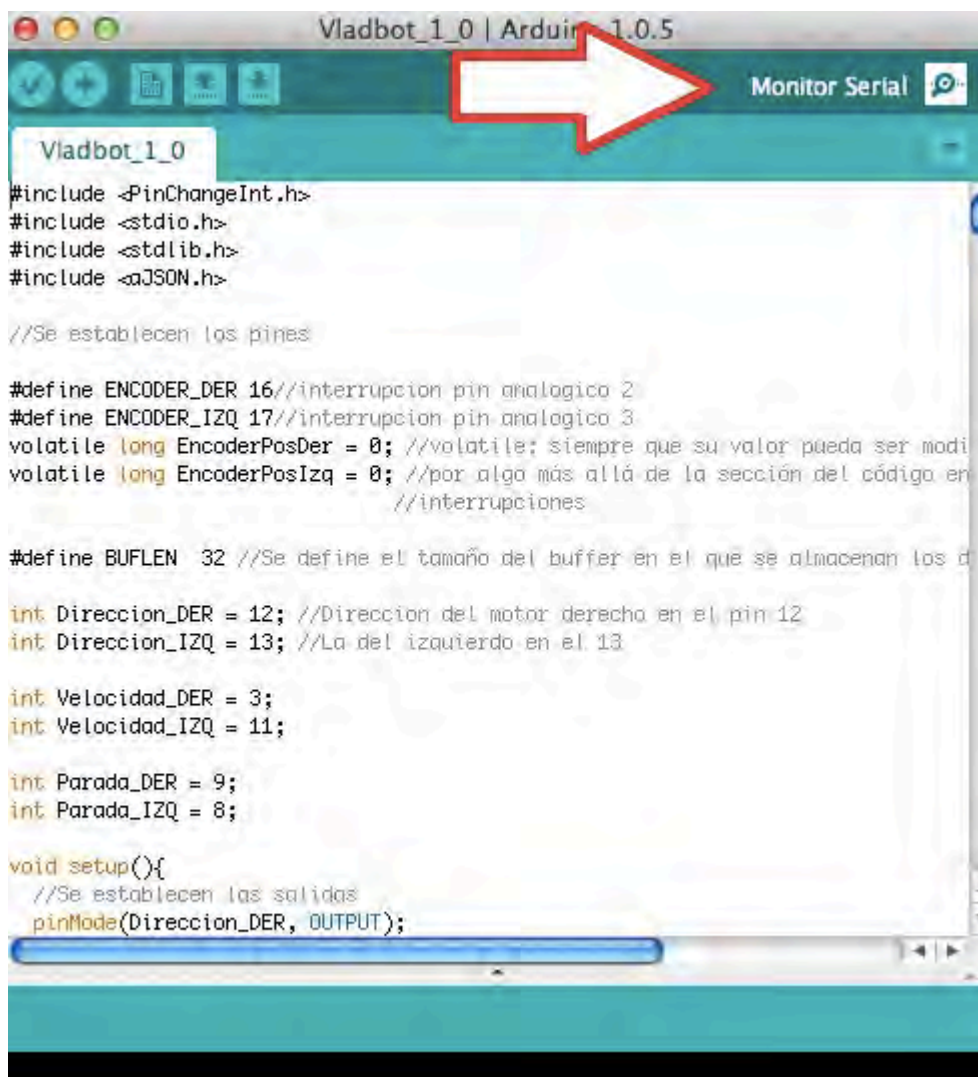


Figura 83: Monitor Serial



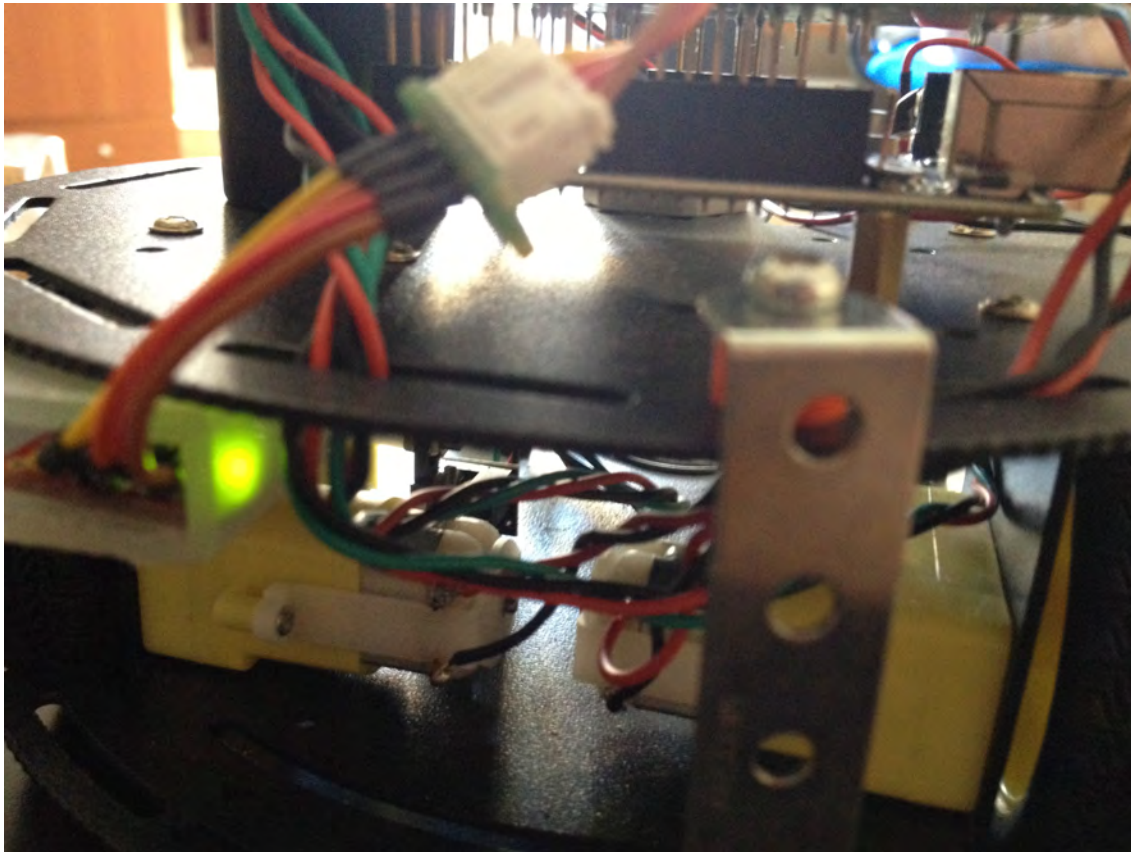


Figura 84: Luz verde sin parpadeos en el módem Bluetooth

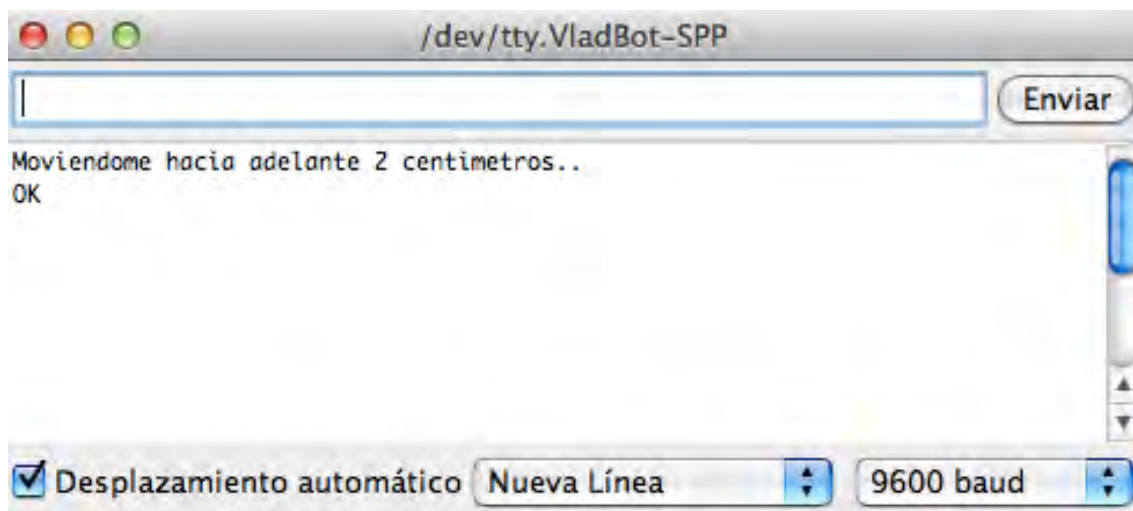


Figura 85: Configuración Monitor Serial

*VladBot* envía el mensaje "OK" cuando termina de realizar un movimiento.



Figura 86: Ejemplo de uso de *VladBot*