ESCUELA DE INGENIERÍA MINERA E INDUSTRIAL DE ALMADÉN

**TRABAJO FIN DE GRADO**

# DISEÑO DE UN ROBOT MÓVIL CON CONTROL DIFUSO POR VOZ

UNIVERSIDAD DE CASTILLA-LA MANCHA

GRADO EN INGENIERÍA ELÉCTRICA

**Autor:** Juan Antonio Serrano Jaime
**Director:** Dr. Javier A. Albusac Jiménez
**Curso: 2013-2014**

"Los ideales que iluminan mi camino y una y otra vez me han dado coraje para enfrentar la vida con alegría han sido: la amabilidad, la belleza y la verdad."

Albert Einstein.

## AGRADECIMIENTOS Y DEDICATORIAS

Me gustaría comenzar este apartado de agradecimientos, dándole las gracias a mi director de proyecto, al Doctor D. Javier Alonso Albusac Jiménez, por ofrecerme la posibilidad de hacer este trabajo fin de grado y manifestarme su interés en dirigirlo, por su confianza, colaboración, consejos y apoyo en mi proceso de realización puesto que sin su ayuda no habría podido ni imaginar por dónde empezar a desarrollar y escribir el trabajo. También agradecer al personal de la Escuela de Ingeniería Minera e Industrial de Almadén por el apoyo mostrado e interés de sus conocimientos para resolver cualquier duda durante la realización de este trabajo fin de grado.

Por último, agradecer a mi familia y seres más queridos, por estar ahí en cada momento dándome ánimos y confiando en mí siempre. Cabe destacar la figura de mi novia Brenda, mi hermana Mª Carmen y mis padres Vicente y Manoli por apoyarme y aguantarme este tiempo y este trabajo va dedicados a ellos.

Dedicar este trabajo para mí es muy importante ya que simboliza que la presentación de él da por finalizada esta carrera y ha sido posible gracias a ellos. Primero a mi padre por darme sus consejos y no dejar que me relaje durante la realización del proyecto e incluso durante toda la carrera. A mi madre por confiar en mí y hacer posible que estos cuatro años de carrera haya llegado a su fin. A mi hermana por estar ahí cuando la he necesitado y por apoyarme en todo momento. También dedico este proyecto a mi novia por apoyarme y animarme en todo momento y por no pasar más tiempo con ella durante la realización de él. A mis abuelos Antonio, Pilar y Juan Antonio que aunque ya no están siempre van conmigo y me acuerdo mucho de ellos. Por último, dedicárselo al más pequeño de la familia; mi sobrino Iker, ya que en los momentos difíciles durante el desarrollo del proyecto, me alegraba jugando conmigo.

A todos aquellos les doy gracias por hacer posible llevar a cabo este proyecto.

Almadén, Mayo de 2014

Juan Antonio Serrano Jaime

## RESUMEN

El objetivo de este proyecto está basado en robótica, ya que consiste en el diseño y construcción de un robot móvil que sea controlado por voz.

Para ello se ha empleado Arduino y componentes compatibles con él. Una vez construido el robot se hizo necesario modelar su comportamiento mediante un lenguaje de programación, como es Arduino (lenguaje empleado en los grados de la propia E.I.M.I.A.). Además, es necesario un segundo lenguaje de programación para la construcción del reconocedor de voz utilizando Matlab (lenguaje también empleado en la E.I.M.I.A.).

La comunicación entre el robot (Arduino) y PC (Reconocedor de voz en Matlab) se hizo por radiofrecuencia sin necesidad de conectar ambos mediante cables. Para que la comunicación sea posible, es necesario programar las librerías de comunicación entre Matlab y Arduino.

Para definir el comportamiento del robot se ha utilizado Inteligencia Artificial. Está basado principalmente en el modelo matemático de la lógica difusa. De esta manera el comportamiento del robot es adaptativo y se busca una conducción suavizada en cuanto a aceleración y frenado similar a la de un humano, dándole una mayor suavidad de control y una respuesta no tan prefijada a la hora de mandar las órdenes.

El uso de estos robots móviles puede ser de gran utilidad en tareas de vigilancia inteligente o para entornos en los que no es aconsejable la presencia humana a un posible riesgo para su integridad física.

Además, la realización de este proyecto abre una puerta interesante a los alumnos de la E.I.M.I.A, ya que todo el material desarrollado podrá ser empleado para un futuro campeonato de carreras con coches construidos con Arduino. De esta forma, no necesitarán invertir tiempo en solucionar los aspectos básicos de control de robot y podrán centrarse en el mejoramiento del robot.

Palabras clave: Robótica Móvil, Reconocimiento de Voz, Lógica Difusa, Órdenes, Señal de voz.

## ABSTRACT

The objective of this project is based on robotics. It consists of the design and construction of a mobile robot controlled by voice.

The construction of the robot is made from scratch. It was searched that the design be as economical as possible.

Arduino and other components for the construction of the robot is used. For the robot to have a certain behavior, you need to program it. The programming language used is Arduino. This type of language is used in EIMIA. For the speech recognizer is used Matlab. This language is also used in the EIMIA. Orders to recognize are: forward, stop, left turns, faster, slower right turns, reversing.

Communication between Arduino and PC is made by radiofrequency. For communication to work you have to program the communication libraries.

Artificial intelligence is used to define the behavior of the robot. Is based mainly on the mathematical model of fuzzy logic. Thus a smoothed condución is achieved. The acceleration and braking of the robot is almost like driving a car. The answer depends on the distance to the obstacle.

The "faster" order, for large distances from the obstacle, the greater the increase in speed. The "slow" order, for small distances to the obstacle, the greater the increase and speed reduction will be higher.

From the PC you can see the environment surrounding the robot captured by camera mobile phone.

The use of this robot can be very useful in surveillance or dangerous places. The realization of this project is interesting for students of EIMIA for future projects that can be done. A future project may be a car racing championship with Arduino. Using this project, students do not need to invest time in creating the basic moves and can use it to improve the robot.

Keywords: Mobile robots, Speech Rcognition, Fuzzy Logic, Orders, Speech Signal.

# ÍNDICE DE ILUSTRACIONES

# ÍNDICE DE TABLAS

# ÍNDICE DE GRÁFICOS DE RESULTADOS

# CAPÍTULO 1: INTRODUCCIÓN

# 1 INTRODUCCIÓN

Siempre se ha pretendido que el ser humano haga el menor esfuerzo posible y no poner en riesgo su salud, su integridad física y aumentar la rapidez en los trabajos que se realizan. Para conseguir esto se han llevado a cabo numerosas investigaciones y proyectos para la implementación de tecnologías a los puestos de trabajo, para que aumente la seguridad y fiabilidad de estos.

La evolución de estas tecnologías ha ido aumentando conforme han pasado los años, comenzando con construcciones básicas de mecanismos que disminuían los esfuerzos realizados por el trabajador, como puede ser una simple polea o palancas. Luego las máquinas fueron capaces de sustituir formas naturales de energía renovable, tales como el viento, mareas, o un flujo de agua por energía humana, hasta ir consiguiendo avances que revolucionaron la industria como puede ser la implementación de sistemas de automatización.

En 1801, la patente de un telar automático utilizando tarjetas perforadas fue dada a *Joseph Marie Jacquard*, quien revolucionó la industrial del textil, haciendo conseguir patrones en la tela, permitiendo que hasta los usuarios más inexpertos pudieran elaborar complejos diseños. Así como esta máquina se han ido desarrollando otras muchas, con tan solo mecanismos mecánicos hasta la aparición de los componentes electrónicos.

Gracias a la electrónica se ha conseguido una avanzada automatización al incorporar computadoras al control de tareas simples, repetitivas, tareas semiespecializadas y especializadas. Para la realización de dichas tareas apareció la robótica, que no es más que un conjunto de mecanismos y circuitos electrónicos previamente programados.

La Robótica es relevante en los estudios de ingeniería hoy en día, debido a que el robot puede realizar trabajos peligrosos, monótonos o precisos. Los humanos al cabo de muchas horas de trabajo disminuyen su velocidad de trabajo y concentración, sin embargo los robots pueden permanecer durante mucho tiempo trabajando a velocidad constante y la precisión no disminuye. No obstante, cabe decir que hay trabajos que un robot no podría hacer, ya que una máquina está muy limitada en la realización de trabajos que impliquen un grado de creatividad o improvisación, tienen unos procesos predefinidos y no pueden

salirse de ellos por sí mismos. Es decir, tienen un comportamiento definido que repiten una y otra vez; repiten el comportamiento para el que han sido diseñadas.

La existencia de robots en la actualidad está en todas partes, desde la industria, como puede ser un brazo robot para coger piezas a elevadas temperaturas, hasta en las viviendas, como pueden ser las aspiradoras que se están comercializando en los últimos años. Aunque estos tipos de robot se utilicen en distintas áreas de trabajo y sectores, al fin y al cabo están compuestos por las mismas unidades. Las unidades de un robot se componen de:

- **Unidades de procesamiento**: son los que procesan los datos de entrada para obtener los datos de salida.

- **Unidades de entrada:** son los datos del exterior, que son percibidos para el ingreso de información para su posterior procesamiento. Estas unidades se llaman sensores, pudiendo ser externos o internos.

- **Unidades de salida:** son los datos que se encargan de comunicar el resultado del procesamiento al usuario u operador. Estas unidades se llaman actuadores, como pueden ser motores, Led, displays, etc.

Una de las unidades más importantes es la de procesamiento, ya que es como bien se ha dicho anteriormente la que procesa los datos, por lo tanto debe de estar correctamente desarrollada para que el comportamiento del robot sea satisfactorio. Para esta unidad se suele utilizar en la actualidad microcontroladores, haciendo los robots más compactos y rápidos, ya que cuando no existían los microcontroladores los robots se hacían implementando circuitos electrónicos muy complejos y costosos de hacer.

En la actualidad se está utilizando mucho los microcontroladores integrados que les permiten conectar entradas y salidas sin necesidad de cablear el microcontrolador ni soldar los terminales. Existes varios tipos de estas placas, una de las que más se están utilizando son las placas Arduino, ya que su bajo coste lo hace accesible a estudiantes de ingeniería y hace posible la realización de trabajos con microcontroladores.

Como todos los microcontroladores se tienen que programar con un lenguaje de programación determinado según el tipo que sea. Por lo tanto al programarlo se permiten una mayor amplitud de funciones que se pueden controlar en comparación con los antiguos robot hechos a base de circuitos electrónicos, dándole al robot una mayor inteligencia.

La inteligencia del robot no es más que la forma de programar que se le da al robot para que actúe de una forma determinada ante unas situaciones dadas. Actualmente se están llevando a cabo robots con inteligencia artificial (IA).

La IA es un área multidisciplinaria que, a través de ciencias, tales como la informática, la lógica y la filosofía, estudia la creación y diseño de entidades capaces de razonar por sí mismas utilizando como paradigma la inteligencia humana, creando así maquinas que pueden llegar a pensar. A veces también se hace necesario la interacción entre el robot y el ser humano, por lo tanto haciendo esto aumenta la inteligencia.

Existen varias formas de interactuar con el ser humano, ya puede ser mediante un ordenador, mandos de videojuegos o incluso por la voz. La forma de interactuar ha llegado a un punto que se están desarrollando proyectos muy interesantes como puede ser la captación de los movimientos de las distintas partes del cuerpo.

Una de las maneras más interesantes de interactuar con un robot es mediante el habla, poder decirle órdenes por voz y que el robot obedezca a esas órdenes. El habla es una de las partes más importantes de la expresión humana, ya que es una de las expresiones esenciales para el hombre, por lo tanto, hace muy interesante esta interacción.

Llegados a este punto, decir que la intención de este proyecto está basado en robótica, ya que **consiste en el diseño y construcción de un robot móvil que sea controlado por voz** y que sea capaz de visualizar el entorno que le rodea. Este proyecto lo podrá utilizar gente con minusvalía (al ser controlado por voz), incluso la automatización que se utiliza en el proyecto se puede llevar a cabo en una silla de ruedas para que estas personas puedan controlar sus desplazamientos mediante el habla, para esto solo haría falta sustituir el chasis del robot y motores por la silla de ruedas y unos motores adecuados, ya que el resto de elementos hardware y software no haría falta modificaciones importantes y sería fácil

de aplicar. El proyecto también puede ser utilizado en vigilancia y para la realización de tareas que impliquen un peligro.

Por último, decir que para la realización de dicho proyecto los conocimientos se han ido adquiriendo a lo largo de estos 4 años **en la obtención del título de Grado en Ingeniería Eléctrica, en la Escuela de Ingeniería Minera e Industrial de Almadén,** como son conocimientos de informática (programación del robot, creación de interfaz de reconocimiento de voz), conocimientos de electrónica y electricidad (microcontroladores, sensores, motores y baterías) en las siguientes asignaturas:

- **Informática** (programación y realización de programas para el control del robot)
- **Sistemas de Comunicación en Edificios** (al llevar control inalámbrico y sistema de video inalámbrico)
- **Máquinas Eléctricas** (a la hora de elección del motor más adecuado para este proyecto)
- **Electrónica** (para la elección de sensores, componentes electrónicas y conexión de ellos)
- **Ciencia de los Materiales** (en la elección de los materiales más adecuados)
- **Teoría de Mecanismos y Estructuras** (para la construcción de la forma más eficiente)
- **Tecnología Eléctrica y Teoría de Circuitos** (a la hora de hacer la el sistema de alimentación a los componentes)
- **Control de Máquinas Eléctricas** (para el control de los motores)
- **Control y regulación automática** (a la hora de regulación de velocidad)
- **Domótica** (al tener Arduino en nuestro proyecto)

# CAPÍTULO 2: OBJETIVOS

# 2 OBJETIVOS

La realización de este proyecto está enfocada a la aplicación de las competencias generales de la titulación de Ingeniería Eléctrica en la construcción de un robot móvil y en la aplicación de lenguajes de programación a casos de ingeniería.

Se trata de **construir un robot cuyo control del mismo se haga mediante comandos hablados**, es decir, que mediante la voz humana el robot obedezca la orden siempre y cuando el robot haya sido programado para dicha orden. Como alternativa del modo anterior se ofrecerá que tenga una **segunda opción de control mediante joystick** de forma manual.

Debido a que el control del robot es remoto de manera inalámbrica será necesario poder observar el entorno que rodea el robot sin estar cerca del mismo. Por ello se integrará una cámara de video al robot y un sistema de transferencia de contenido multimedia para visualizar en tiempo real el flujo de video en el computador a través del cual se utiliza el control del robot.

Para cumplir los objetivos generales expuestos anteriormente es necesario satisfacer los siguientes puntos.

1. **Diseño y construcción desde cero de un robot móvil lo más económico posible:** Se deberá elegir los microcontroladores y componentes que sean más económicos pero que a su vez tengan buenas características y se consigan los objetivos del robot, es decir, que haya un equilibrio entre calidad precio.

2. **Reconocimiento de órdenes mediante voz para el control del robot:** Se pretende controlar el robot por órdenes habladas, procesadas mediante un ordenador y que este ordenador se las comunique al robot. Para ello, el sistema de control deberá ser capaz de captar la señal de audio y analizarla para establecer similitudes con patrones prefijadas y establecidos con anterioridad. La similitud entre la señal actual y algunos de los patrones que forman parte de la base del conocimiento del sistema, implica el reconocimiento de algunas de las órdenes a las que el robot debe responder y para la que ha sido programada.

3. **Motor de inferencia difusa para definir el comportamiento del robot (Inteligencia Artificial):** El motor de inferencia o razonamiento del robot estará basado principalmente en el modelo matemático de la lógica difusa. Con esta forma de controlar el robot se busca que éste se comporte de forma distinta ajustada a la situación actual en la que se encuentre. Es decir, el comportamiento del robot es adaptativo y se busca una conducción suavizada en cuanto a aceleración y frenado similar a la de un humano, **dándole una mayor suavidad de control y una respuesta no tan prefijada** a la hora de mandar las órdenes.

La elección de controlar el robot mediante la voz es por la necesidad que se da en algunos casos de tener las manos ocupadas y no poder controlar el robot con ellas. Por lo tanto **con la voz podemos controlar el robot mientras se hace otras operaciones con las manos**, como puede ser el control de otro robot mediante joystick. La segunda opción de controlar el robot con un joystick es para asegurarnos de la utilización del robot cuando el sistema de reconocimiento de voz entre en conflicto con los ruidos del exterior, porque posiblemente habrá lugares donde haya mucho ruido o ecos muy fuertes y este sistema no funcione adecuadamente, por lo tanto, con esta opción podemos controlar el robot sin problemas y disfrutar de un sistema mucho más robusto.

**El uso de la lógica difusa es** principalmente por tener la posibilidad de procesar órdenes que implican vaguedad como “más despacio”, ”más deprisa” y **no tener un comportamiento exactamente lineal sino más bien como el de una persona cuando conduce un vehículo.** Gracias al control difuso, el robot podrá reconocer órdenes que forman parte del comportamiento en base a las condiciones ambientales y el robot no se comportará exactamente como un autómata determinista. *Por ejemplo si para llegar al obstáculo quedan 5 metros y se ordena la orden de “más deprisa” la aceleración será mayor que si el obstáculo se encuentra a 2 metros, y para la orden de ir “más despacio” será al contrario, a mayor distancia de obstáculo menor será la desaceleración.*

Por último decir que toda la información para la realización del proyecto, librerías y dispositivos podrán ser utilizadas para posibles prácticas de asignaturas o posibles proyectos futuros.

# CAPÍTULO 3: ESTADO DEL CONOCIMIENTO

# 3 ESTADO DEL CONOCIMIENTO

## 3.1 Historia.

Los robots llevan casi 50 años presentes en los procesos industriales *[24]*. Los primeros robots en los años 50 y principios de los 60 fueron gracias al desarrollo de nuevas tecnologías, como lo son los transistores y los circuitos integrados.

En la actualidad existen multitud de robots con diferentes características que han sido diseñados para fines de diversa índole. Existen robots de uso doméstico, para ayudas médicas, para labores peligrosas y robots para industrias. También hay robots llamados Androides, que son los robots que se asemejan a las características humanas, pero estos están muy lejos de simular perfectamente a una persona.

La palabra robot se usó por primera vez en 1921, en una obra de teatro llamada *Rossum´s Universal Robots (RUR),* escrita por el checo *karel Capek* (1890-1938). En la obra un fabricante ficticio fabricaba robots para reemplazar a los trabajadores humanos. Estos robots obedecían las órdenes dadas sin preguntar. Los robots se volvieron contra sus amos y acabaron con la raza humana, menos un hombre para que pudiera producir más robots.

Existen muchas formas de definir el término robot, una puede ser del Robot Institute of America (RIA) de 1979: *"es un manipulador reprogramable y multifuncional diseñado para mover material, partes, herramientas o bien dispositivos especializados para desempeñar una variedad de labores a través de movimientos diversos programados."*

La Organización Internacional para la Estandarización (ISO) define robot como: *"un manipulador multifuncional reprogramable, capaz de mover materiales, piezas, herramientas o dispositivos especiales, a través de movimientos variables programados."*

La definición de robot según la RAE (Real Academia Española) es: *"Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales"*

Existen otras definiciones dadas por otras asociaciones, como por ejemplo Japan Industrial Robot Association (JIRA) que dice: *"son dispositivos capaces de moverse de modo*

*flexible, análogo al que poseen los órganos, con o sin funciones intelectuales, lo que permite la realización de operaciones en respuesta a órdenes recibidas por humanos.*

El atraso del desarrollo sobre robótica ha sido por el miedo de las personas a que un robot pueda apoderarse del trabajo de ellas, por lo que este pensamiento ha atrasado el desarrollo en esta área. Isaac Asimov postuló tres reglas básicas para robot, que se conocen *como las leyes de la robótica* que son *[24]*:

1) Un robot no debe dañar a un ser humano.
2) Debe obedecer las órdenes que le da el ser humano, al menos si se contradice la primera ley.
3) Un robot debe proteger su propia existencia, al menos que se contradiga las dos leyes anteriores.

En el año 1999 Fuller introdujo una cuarta ley que dice:

4) Un robot puede hacer el trabajo de un ser humano, pero a esa persona no se le puede dejar sin empleo. Esta cuarta ley fue creada por el miedo de las personas a perder sus trabajos.

## 3.2 Los primeros robots

Los primeros robots nacen en los años 50 principios de los 60, son robots industriales conocidos como *Unimates* diseñados por George Devol y Joe Engelberger, siendo el último el "Padre de la Robótica" ya que fue el primero en comercializar estas máquinas. *Unimates* se instaló en una cadena de montaje de General Motors y es un robot que realizaba el trabajo de transportar las piezas fundidas en molde hasta la cadena de montaje y soldar estas partes sobre el chasis del vehículo.

En 1974, la empresa Cincinnati Milacron hizo el robot *The Tomorrow Tool.*

## 3.3 Tipos de Robot

Los robots actuales son máquinas diseñados para realizar labores productivas específicas *[3, 22, 28]*. La gran mayoría de los robots en la actualidad se emplean en la manipulación

industrial, es decir “brazos” y “manos” que son controlados por ordenadores. Hay diferentes tipos y clasificaciones de los robots que se detallarán en las sucesivas secciones.

### 3.3.1 Según el uso del robot:

Según el uso que se le vaya a dar a los robots, se puede clasificar de las diferentes maneras según su utilización *[29]*:

- ***Robots Industriales:*** se emplean en la industria y el objetivo principal de éstos es el de servir a un propósito universal y de mano de obra no calificada o semicalificada, como puede ser pintar, soldar, realizar mecanizados, etc. Este tipo de robot ha sido el más desarrollado (**Ilustración 1**).

**Ilustración 1:** Brazo robot.

- ***Robots para usos especiales***: se utilizan para desenvolverse en zonas inexploradas y a largas distancias de su centro de control, también se emplea en ambientes distintos del entorno normal de una fábrica, como puede ser un robot de serie montado sobre una nave espacial para la recuperación de un satélite defectuoso.

- ***Médicos:*** se utilizan como ayuda a la intervención médica sobre los humanos y como complemento para personas con capacidades disminuidas.

- ***Domésticos:*** son robots que realizan las tareas del hogar, como son aspiradoras (**Ilustración 2**), lavarropas, robots de concina, etc., que modifican su comportamiento en forma autónoma según el ambiente en el que trabajan.

**Ilustración 2:** Robot aspirador.

- ***Sociales:*** Son robots para la interacción

con los humanos que han sido utilizados en supermercados, e incluso en la industria de películas. Estos deben de ser diseñados para realizar una comunicación completa, con gestos, tonos, silencios, etc.

- ***Agrícolas:*** al igual que la robótica está muy extendida en la industria, en la agricultura se está empezando a extender, ya que existe mucha variedad de robots para realizar las tareas agrícolas, como pueden ser las cosechadoras autónomas (**Ilustración 3**), las sembradoras controladas por mapas satélites.

**Ilustración 3:** Deméter (sembradora automática).

### 3.3.2 Según el medio en el que desarrolla la actividad:

Los robots se pueden utilizar en diversos lugares, como puede ser por tierra, agua, aire o incluso para más de un lugar:

- ***Acuáticos:*** Sus movimientos son tridimensionales, en un ambiente hostil desde el punto de vista mecánico y electrónico (**Ilustración 4**).

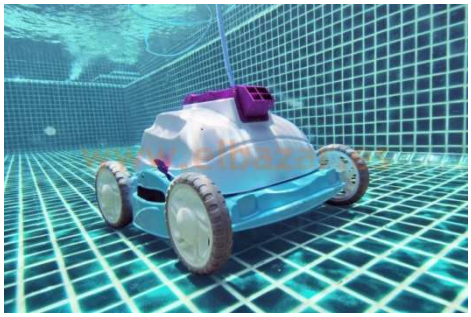


**Ilustración 4:** Robot limpiafondos.

- ***Terrestres:*** son los que más se ven y los mas económicos. Dentro de este grupo se pueden clasificar según el sistema de locomoción, que se verá más adelante.

- ***Aéreos:*** son robots con movimientos tridimensionales como los acuáticos, pero necesitan una precisión mucho mayor a la hora de manejarlos en tiempo real.

- ***Híbridos:*** son robots con ambas combinaciones anteriores. Hay que tener muy en cuenta a la hora de diseñar la mecánica y electrónica del robot, para que pueda funcionar en más de un medio.

### 3.3.3 Según la ubicación de la inteligencia del robot:

- ***Autónomos:*** la inteligencia ésta ubicada en el propio robot, las funciones de lo que debe hacer el robot va incorporado en él, de manera que su funcionamiento no dependa de algo exterior.

- ***Control automatizado***: se pueden llamar también robots semiautónomos, la mayor parte de la inteligencia esta en un sistema central. Los sensores del robot envían la información a ese sistema central y es este el encargado de comunicar las acciones que debe de hacer el robot.

- ***Híbridos***: son robots autónomos, que en un cierto caso pueden ser controlados por personas o por un sistema central. Un ejemplo son los robots de misiones especiales, que son autónomos pero ante un caso pueden ser controlados por otro sitio.

### 3.3.4 Según el sistema de locomoción:

Como se ha visto antes, los robots terrestres tienen varias maneras de moverse, que pueden clasificarse:

#### *3.3.4.1 Robots Orugas:*

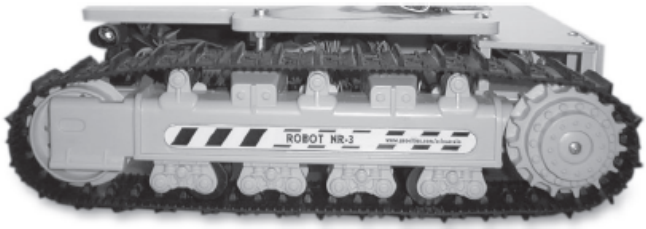



**Ilustración 5:** Robot Oruga.

Los robots orugas **(ilustración 5)**, llevan ruedas, que van unidas la rueda delantera y la trasera. La ventaja de este sistema es que tiene una mayor adherencia al terreno y puede adaptarse mejor por superficies irregulares. Suelen ser de goma pero también las hay metálicas. Una de las desventajas es la imposibilidad de realizar trayectorias curvas. La manera de girar, es dándole a las orugas una dirección opuesta a la otra, lo que hace lograr el giro, pero el avance solo lo hace en línea recta.

#### *3.3.4.2 Robots con Patas:*

Estos robots **(ilustración 6)**, tienen la ventaja de adaptarse a terrenos complejos y muy irregulares. La construcción de estos robots es compleja, ya que se tiene que conseguir que el robot tenga equilibrio y la coordinación tiene que ser muy precisa. Estos robots presentan 3 grados de libertad en cada pierna (cadera, rodilla y tobillo). Cuanto mayor sea el número de patas mayor será el equilibrio.

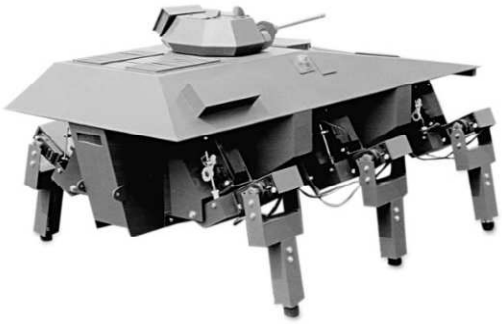


**Ilustración 6:** Robot con patas.

#### *3.3.4.3 Robots Ápodos:*

**Ilustración 7:** Ápodos.

Los robots de este tipo **(ilustración 7)**, son los que no tienen ruedas, ni orugas o patas. El sistema a conseguir es como los gusanos o serpientes. Estos robots están compuestos por módulos y articulaciones pequeñas. Pueden moverse sobre superficies muy diversas y tomar configuraciones distintas para entrar en lugares complicados. El movimiento de estos robots se consigue con un efecto de onda a partir de la cola. En esta arquitectura lo complejo es la coordinación de los módulos y adaptación de sensores para distintos terrenos.

#### *3.3.4.4 Sistemas con ruedas:*

Existes diferentes configuraciones *[29]*:

- **Configuración diferencial (ilustración 8):** es la configuración de ruedas más sencilla y económica. Se utilizan dos ruedas, una a cada lado. Lleva incorporada una tercera rueda libre, para darle un punto de apoyo. Al tener un motor

**Ilustración 8:** Configuración diferencial.

cada rueda, es complicado que el robot siga en línea recta, sin hacer uso de sensores internos como encoders o sensores externos.

La dificultad de este sistema es la de ir hacia un punto concreto, ya que los giros se tienen que hacer poco a poco y despacio.

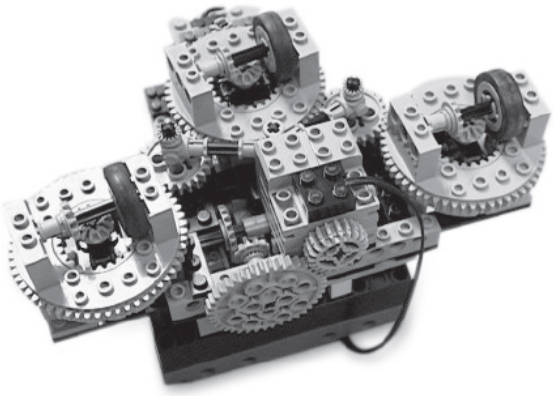


**Ilustración 9:** Configuración síncrona.

- **Configuración síncrona (ilustración 9):** el sistema tiene tres ruedas, alimentado por un mismo motor. Con esto se consigue que el robot vaya en línea recta de forma precisa. Para cambiar de dirección tiene incorporado un segundo motor que actúa rotando las tres ruedas, y después seguirá en línea recta. Para que el chasis vaya siempre apuntando de frente, el sistema deberá llevar un mecanismo para que también gire.

- **Configuración Triciclo (ilustración 10):** Tenemos un sistema con tres ruedas, dos de ellas con tracción movidas por un mismo motor, la otra rueda no tiene tracción, pero si tiene giro sobre su eje, dándole dirección al robot. En línea recta se comporta bien, al ir las dos ruedas gobernadas por el mismo motor, pero en movimientos curvos es complejo, ya que el radio de las curvas es muy grande. No puede girar sobre sí mismo ni hacer curvas cerradas. Para robot que se muevan en interiores pequeños puede ser un gran inconveniente.

**Ilustración 10:** Configuración Triciclo.

- **Sistema Ackerman (ilustración 11):** Este sistema es el que tienen los automóviles convencionales. Las ruedas traseras tienen tracción y las delanteras son las que dan la dirección. Tiene más estabilidad que el sistema triciclo. El problema es que hay que llegar a que las dos ruedas giren en sincronismo. En línea recta no tiene inconvenientes, pero para realizar curvas es complicado. Las ruedas traseras aunque vayan gobernadas por el mismo motor, la tracción es diferencial, es decir una rueda puede girar más que otra si fuera necesario.

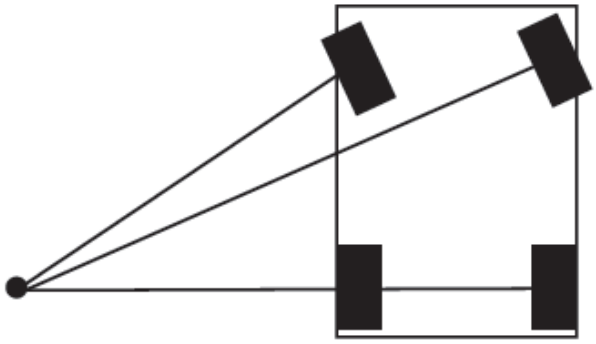


**Ilustración 11:** Configuración Ackerman.

- **Otra configuración:** Una configuración posible podría ser un robot con cuatro ruedas gobernadas por motores diferentes. Para que el robot consiga ir en línea recta los cuatro motores deberían girar en sincronismo. Para los giros hacia un lado u otro las ruedas de un lado deberán girar en sincronismo en dirección contraria a las del lado apuesto. Y otra configuración puede ser las dos ruedas delanteras cada una gobernada con un motor diferentes, y dos ruedas traseras fijas.

## 3.4 Inteligencia del robot:

Cuando Asimov creó sus primeros robots de ficción, al no existir computadores tuvo que inventar los cerebros positrónicos, que son dispositivos con inteligencia artificial que le permitirá al robot almacenar información, realizar operaciones lógicas y tomar sus propias decisiones. Este personaje se adelantó en el tiempo e hizo una descripción de lo que son hoy en día de los controladores que tenemos.

Los primeros robots fueron construidos con simples autómatas, que eran una sucesión de combinaciones de engranajes, poleas y otros artilugios electromecánicos. Luego surgieron los robots con electrónica analógica y compuertas digitales, que en algunos casos hasta se podrían programar con algebra de boole. Ahora en la actualidad nuestros robots poseen microcontroladores y computadores.

### 3.4.1 Microcontrolador:

Un microcontrolador *[13]* es un circuito integrado programable capaz de ejecutar las órdenes grabadas en su memoria. Un microcontrolador incluye en su interior las tres unidades funcionales principales: unidad central de procesamiento (CPU), memoria y periféricos de entrada y salida (I/O). Para que el programa pueda ser grabado en la memoria del microcontrolador, debe ser codificado en sistema numérico hexadecimal, que es el sistema que hace trabajar al microcontrolador cuando es alimentado con el voltaje adecuado y asociado a dispositivos analógicos y discretos para su funcionamiento.

Los puertos de entrada/salida en el microcontrolador, se agrupan en puertos de 8 bits de longitud, lo que permite leer datos del exterior. El objetivo de los microprocesadores es la de trabajar con dispositivos simples como relés, LED, motores, fotocélulas, etc.

El primer microcontrolador TMS 1000 fue creado en 1971 y comercializado en 1974, hecho por los ingenieros de Texas instruments (Gary Boone y Michael Cocharan). Combina memoria ROM, memoria RAM y microprocesador y reloj en un chip.

Los microprocesadores con memoria EPROM reprogramable son más caros que la variantes PROM ya que este solo se podía programar una vez. Para borrar la EPROM se necesita exponer a la luz ultravioleta la tapa de cuarzo transparente. Los microprocesadores con memoria EEPROM permiten que el borrado sea eléctrico y rápido sin la necesidad de un paquete costoso como en la EPROM.

En la realización de este proyecto utilizaremos placas de Arduino, que incorpora que dependiendo de la placa que sea utilizara un tipo de microprocesador u otro. Más adelante se verán las diferentes placas y microprocesadores de Arduino.

## 3.5 Sensores:

Los sensores miden algunas características del ambiente, o sea, permiten sentir el mundo que les rodea, como puede ser la temperatura, distancia, posición, ruido, etc. En algunos casos se necesitarán que los sensores sean muy rápidos y que la precisión no sea muy importante para transmitir información y actuar rápidamente. En otros casos será más importante la precisión que la rapidez, todo depende para que se pondrán sensores y

dependerá de los objetivos y tarea que el robot debe cumplir. Otra cosa a tener en cuenta es el tipo de controlador que se utiliza, ya que en algunos casos se debe de intercalar otro circuito entre el sensor y el controlador, para adaptar las señales del sensor a las de la entrada del controlador, ya que algunas veces no están en la misma magnitud.

Se pueden clasificar en varios grupos, como puede ser **analógico y digital.** Los analógicos dan valores dentro de un rango continuo (**Ilustración 12**), un ejemplo puede ser una fotorresistencia que mide la intensidad de la luz. Los sensores digitales entregan una señal discreta dentro de un conjunto posible de valores (**Ilustración 13**), un ejemplo de un sensor digital puede ser una fotocélula, que entra 0 si no detecta y 1 si detecta.

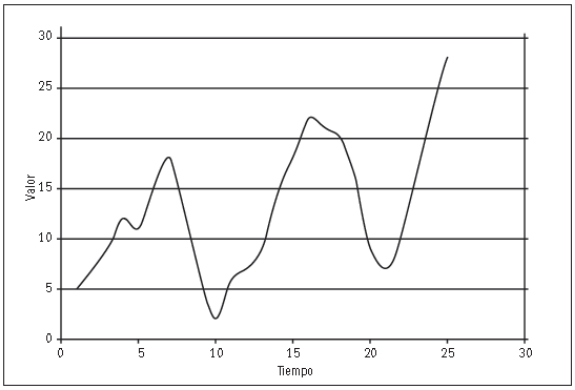



**Ilustración 13:** Grafico analógico.

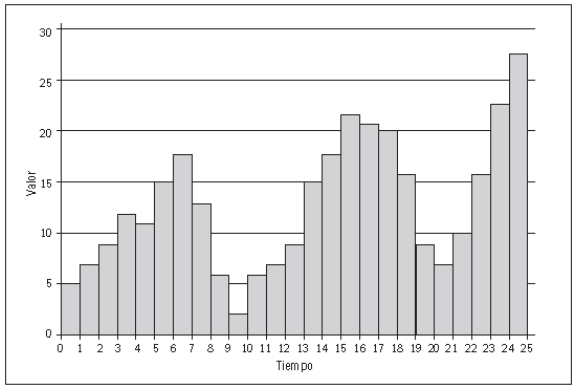



**Ilustración 12:** Grafico Digital.

Otro grupo de clasificación de sensores pueden ser **internos** y **externos**. Los internos son los que proporcionan información del propio robot, como puede ser velocidad, rotación, posición, etc. Los sensores externos son los que proporcionan datos del ambiente, como puede ser temperatura, humedad, etc.

También se pueden clasificar los sensores como **pasivos y activos**. Los sensores activos son los que necesitan enviar una señal al ambiente para luego recibir la información en el rebote, un ejemplo es el sensor ultrasónico.

Describir todos los **tipos de sensores** que hay seria elaborar una lista muy amplia, por lo cual se nombran algunos que son los más utilizados *[29]*:

- **Sensores de interrupción:** detectan si pasa corriente o no. Se utilizan como sensores de choque o para contar el número de veces que sucede un hecho, como puede ser contar el número de veces que el robot choca contra un objeto.

- **Sensores de posición:** este sensor permite saber la posición de los objetos del robot, como puede ser a través de un potenciómetro, que al variar su valor de resistencia se puede determinar en qué posición se encuentra.

- **Sensores efecto hall:** estos sensores se utilizan para la detección de metales. Se basan en la variación de la conductividad al acercarse a un cuerpo metálico.

- **Sensores de luz o brillo:** son sensores que detectan la luz. Algunos sensores pueden detectar colores y si son brillantes o no. Una aplicación de estos sensores seria detectar si hay luz en el exterior de una vivienda para subir o bajar las persianas.

- **Sensores infrarrojos:** estos sensores envían una señal luminosa infrarroja, y según sea el sensor se puede utilizar para medir distancias o detectar intrusiones.

- **Utilización de captación de video como sensor:** últimamente se tiende mucho a incorporar cámaras a los robots, ya que es una de las opciones que más información capta. Este proyecto lleva incorporado una cámara como ya se ha dicho en capítulos anteriores.

## 3.6 Motores:

Un motor eléctrico es una máquina que convierte la energía eléctrica en energía magnética y después en mecánica. Los motores disponen de un eje, para acoplarle engranajes, ruedas, o cualquier mecanismo para trasmitir movimientos creados por el motor. Para darle desplazamiento a los robots necesitamos motores.

La elección de los motores es una tarea complicada, según las características y limitaciones del robot, si tenemos en cuenta todas las características que definen al motor, como puede ser tamaño, peso, velocidad, rendimiento, par, tensión, etc. Son muchos parámetros a tener en cuenta y es una labor complicada. A continuación se verán los diferentes tipos de motores que se pueden utilizar en robots.

### 3.6.1 Motores de corriente continua (CC):

El funcionamiento de los motores de corriente continua **(ilustración 14)**, se basa en la acción de campos magnéticos opuestos que hacen girar el rotor (eje interno) en dirección opuesta al estador (imán externo o bobina). Para invertir el giro de estos motores, basta con invertir la polaridad de la corriente eléctrica. La mayoría de las veces llevan incorporado un juego de engranajes que hacen reducir o aumentar la velocidad, o dar más par o menos, ya que su número de revoluciones es muy elevado.

**Ilustración 14**: Motores de corriente continua.

Estos tipos de motores se pueden encontrar en juguetes y son muy económicos.

### 3.6.2 Motores paso a paso (PAP):

Los motores paso a paso (**Ilustración 15**) controlan en todo momento la posición de su eje, para conseguir movimientos muy precisos y mantiene la posición. Si por ejemplo un motor tiene 12 pasos/rev, entonces cada movimiento tendrá exactamente 30 grados.

**Ilustración 15:** Motores paso a paso.

Tiene el rotor constituido por un imán permanente, y el estador construido por bobinas. Al alimentar una de esas bobinas el polo apuesto del rotor busca y se orienta a esa bobina, y este permanece en esa posición hasta que se excita otra bobina, lo que hace avanzar o retroceder al rotor.

Los motores PAP pueden ser de dos tipos **bipolares o unipolares.** Los motores PAP bipolares llevan dos bobinas independientes. Para controlarlo se necesita invertir la polaridad de cada una de las bobinas en la secuencia correcta. Los motores PAP unipolares llevan 5 ó 6 cables, dependiendo si el común está unido en forma interno o no. Para controlar este tipo de motores se puede hacer de tres maneras con sus correspondientes secuencias de encendido de bobinas. El común se conecta a +Vcc o masa según el circuito de control usado, y después sólo hay que alimentar la bobina correcta para que el motor avance o retroceda según avance o retroceda en la secuencia. Estos motores se encuentran en disqueteras, impresoras.

### 3.6.3 Servomotores:

El servo (**ilustración 16**) lleva en su interior un pequeño motor con un reductor de velocidad y un multiplicador de fuerza. Es un dispositivo pequeño pero muy potente. El recorrido del eje de salida es de 180º en la mayoría de ellos, aunque los hay que giran 360º que actuaria como un motor común. El inconveniente es que son lentos. Los servos se utilizan mucho para mover brazos o controlar la dirección.

**Ilustración 16:** Servomotor.

El control de posición se puede controlar mediante la colocación de un potenciómetro en el eje de la salida, que al variar la resistencia se puede saber la posición del eje. Este controla un PWM (modulador de anchura de pulsos) interno para compararlo con la entrada PWM externa del servo, mediante un sistema diferencial, y así, modificar la posición del eje de

salida hasta que los valores se igualen y el servo se detenga en la posición indicada. En esta posición el servo se puede decir que no consume corriente, salvo un poco en el circuito interno. Si en este estado se modifica la posición del servo por motivos externos, el control diferencial hace que pase la corriente necesaria para corregir la posición.

Para controlar un servo, hay que aplicar un pulso de duración y una frecuencia específica. Los servos tienen tres cables, que son para la alimentación y otro para aplicar el tren de pulsos de control que hará que el circuito de control diferencial interno ponga el servo en la posición indicada por la anchura del pulso.

## 3.7 Arduino:

Arduino *[21]* es una plataforma de electrónica abierta para la creación y construcción de prototipos basada en software y hardware libre, flexible y fácil de usar. La creación de esta plataforma fue para aficionados, artistas en crear entornos u objetos interactivos.

Arduino recibe información del entorno a través de sus pines de entrada, mediante una gama amplia de sensores disponibles, pudiendo controlar el entorno que nos rodea, como puede ser motores, servos, luces, etc. Para trabajar con Arduino se necesita tener conocimientos de electrónica y programación.

El microcontrolador situado en la placa se controla con el lenguaje de programación Arduino (basado en *Wiring*[1]) y el entorno de desarrollo Arduino (basado en *Processing*[2]). Los proyectos hechos con Arduino se pueden ejecutar sin necesidad de conectarlos a un PC.

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más utilizados son Atmega168, Atmega328, Atmega1280, Atmega8 por su bajo coste y sencillez.

Las placas se pueden comprar hechas o para montarlas.

[1] Wiring es un marco de programación de código abierto para los microcontroladores.
[2] Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización.

### 3.7.1 Placas Arduino:

Existen varias placas de Arduino, en función de las entradas y salidas que se desea tener. Las distintas placas tienen procesadores con características diferentes según para lo que se quiera utilizar. La elección de la placa se hará en función de las características que deseamos que nuestro robot tenga. A continuación se verán los diferentes tipos de placas.

**a) Arduino Uno (Ilustración 17):**

Está basado en ATmega328 Tiene 14 pernos digitales de entrada-salida (de las cuales 6 se pueden utilizar como salidas PWM), 6 entradas analógicas. Contiene una conexión USB y un conector de alimentación y conector ICSP Se puede conectar al PC mediante un cable USB. Esta placa se diferencia de las otras porque no utiliza el chip controlador FTDI USB-to-serial, en cambio tiene el ATMEGA16U2 programado como convertidor USB a serie.

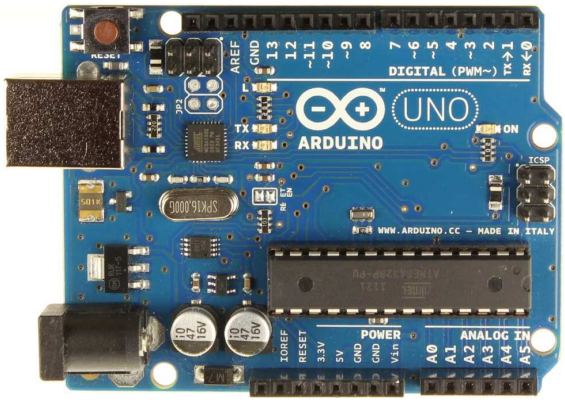



**Ilustración 17:** Placa Arduino Uno.

Las características de la placa son:

- Microcontrolador ATmega328, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la Tensión de entrada limites de 6-20 V.
- Corriente de la C.C. por el pin de entrada-salida 40 mA.
- Corriente de la C.C. para el pin 3.3 V 50 mA.
- Memoria flash de 32 Kb
- SRAM 2 Kb y EEPROM 1 Kb.

### b) Arduino Due (Ilustración 18):

Está basado en Atmel SAM3X8E CPU Tiene 54 pernos digitales de entrada-salida (de las cuales 12 se pueden utilizar como salidas PWM), 12 entradas analógicas. 4 UARTs (puertos serie de hardware) 2 salidas DAC (convertidores de analógico a digital) Contiene 2 conectores USB, un conector de alimentación, un conector ICSP, un conector JTAG En uno de los USB se puede conectar otros dispositivos USB a la tarjeta (ratones, teclados, etc..)

Las características de la placa son:

- Microcontrolador AT91SAM3X8E, la placa funciona con 3.3 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas entradas-salidas 130 mA.
- La Corriente máxima de los pines de Entrada/Salida de 3.3 V 800 mA.
- La Corriente máxima de los pines de Entrada/Salida de 5 V 800 mA.
- Max. 5 V en un pin de entrada.
- Memoria flash de 32 Kb
- SRAM 96 Kb y velocidad de reloj 84 MHz

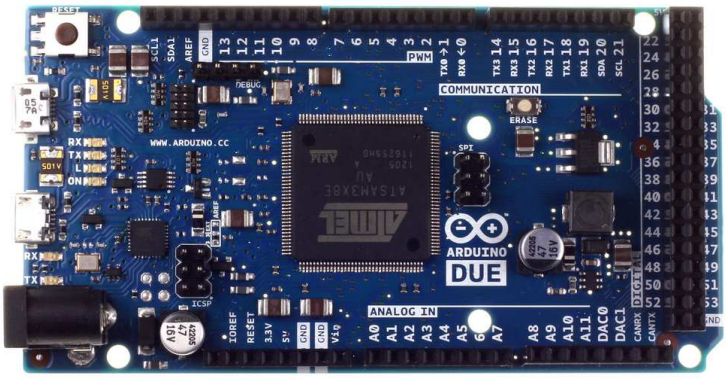



**Ilustración 18:** Placa Arduino Due.

### c) Arduino ADK (ilustración 19):

Está basado en ATmega 2560. Lleva una USB para conectar los teléfonos que están basados en Android. Tiene 54 pines digitales de entrada-salida (de las cuales 14 se pueden

utilizar como salidas PWM), 16 entradas analógicas. 4 UARTs (puertos serie de hardware) Un oscilador de cristal de 16 MHz Contiene una conexión USB, un conector de alimentación, un conector ICSP Lleva un circuito USB que permite comunicarse con los dispositivos USB y suministrarle alimentación.

Las características de la placa son:

- Microcontrolador ATmega2560, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas entradas-salidas 40 mA.
- La Corriente máxima de los pines de Entrada/Salida de 3.3 V 50 mA.
- Memoria flash de 256 Kb
- SRAM 8 Kb, EERPROM 4 Kb y velocidad de reloj 16 MHz.

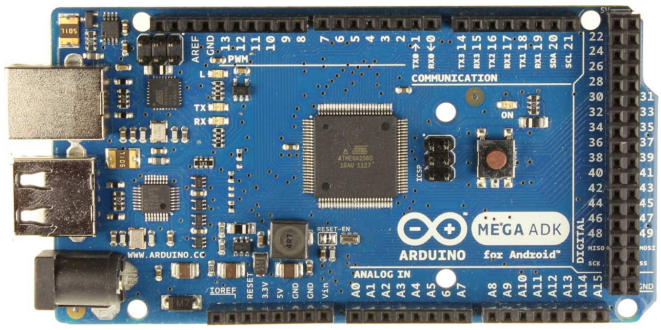



**Ilustración 19:** Placa Arduino ADK.

### d) Arduino Mega2560 (Ilustración 20):

Está basado en ATmega 2560 Tiene 54 pines digitales de entrada-salida (de las cuales 14 se pueden utilizar como salidas PWM), 16 entradas analógicas. 4 UARTs (puertos serie de hardware) Un oscilador de cristal de 16 MHz Contiene una conexión USB, un conector de alimentación, un conector ICSP

Las características de la placa son:

- Microcontrolador ATmega2560, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas entradas-salidas 40 mA.

- La Corriente máxima de los pines de Entrada/Salida de 3.3 V  50 mA.
- Memoria flash de 256 Kb
- SRAM 8 Kb, EERPROM 4 Kb y velocidad de reloj 16 MHz

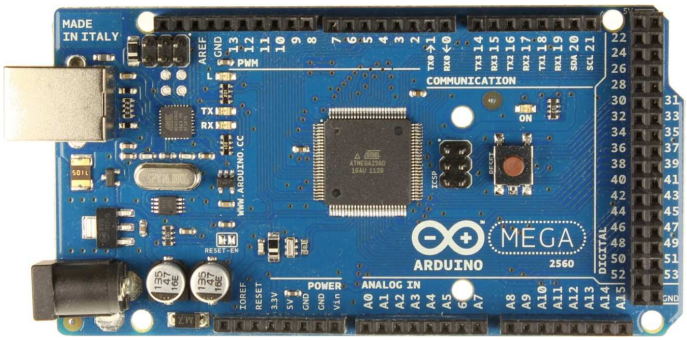



**Ilustración 20:** Placa Arduino Mega.

### e) Arduino Nano (Ilustración 21):

Está basado en ATmega 328. No tiene toma de alimentación y se programa a través de un cable mini-USB. Tiene 14 pines digitales de entrada-salida (de las cuales 6 se pueden utilizar como salidas PWM), 8 entradas analógicas.

Las características de la placa son:

- Microcontrolador ATmega328, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas  entradas-salidas 40 mA.
- Memoria flash de 32 Kb
- SRAM 2 Kb, EERPROM 512 bytes y velocidad de reloj 16 MHz


**Ilustración 21:** Placa Arduino Nano.

### f) Arduino Leonardo (Ilustración 22):

Está basado en ATmega 32u4. Tiene 20 pines digitales de entrada-salida (de las cuales 7 se pueden utilizar como salidas PWM), 12 entradas analógicas. Un oscilador de cristal de 16 MHz Contiene una conexión USB, un conector de alimentación, un conector ICSP. Esta placa es diferente a todas las demás, porque incorpora en su interior la parte de comunicación USB, y no necesita un microprocesador secundario, por lo tanto cuando se conecta al ordenador se reconoce como un ratón o un teclado.

Las características de la placa son:

- Microcontrolador ATmega32u4, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas entradas-salidas 40 mA.
- La Corriente máxima de los pines de Entrada/Salida de 3.3 V 50 mA.
- Memoria flash de 32 Kb
- SRAM 2.5 Kb, EERPROM 1 Kb y velocidad de reloj 16 MHz

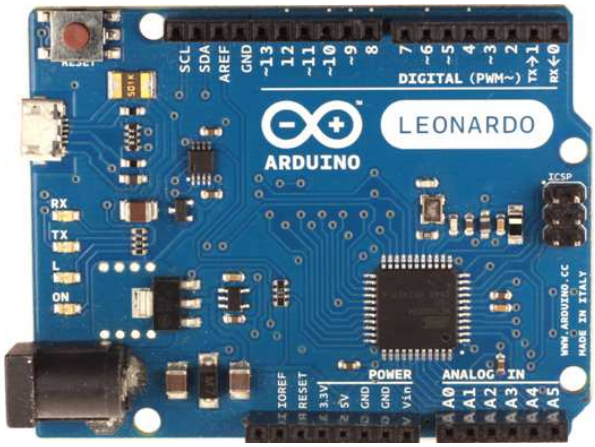



**Ilustración 22:** Placa Arduino Leonardo.

### g) Arduino YÚN (Ilustración 23):

Es la combinación de un Arduino Leonardo, con un chip Wifi que utiliza Linino (un MIPS GNU/Linux basada en OpenWRT). Incorpora Linux en la PCB de la placa Arduino Leonardo y están conectados los dos para ejecutar comandos en Linux, y se utiliza como una interfaz Ethernet y Wifi. Está basado en ATmega 32u4 Tiene 20 pines digitales de

entrada-salida (de las cuales 7 se pueden utilizar como salidas PWM), 12 entradas analógicas. Un oscilador de cristal de 16 MHz Contiene una conexión USB, un conector de alimentación, un conector ICSP Esta placa es diferente a todas las demás, porque incorpora en su interior la parte de comunicación USB, y no necesita un microprocesador secundario, por lo tanto cuando se conecta al ordenador se reconoce como un ratón o un teclado.

Las características de la placa son:

- Microcontrolador ATmega32u4, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas  entradas-salidas 40 mA.
- La Corriente máxima de los pines de Entrada/Salida de 3.3 V  50 mA.
- Memoria flash de 32 Kb
- SRAM 2.5 Kb, EERPROM 1 Kb y velocidad de reloj 16 MH


**Ilustración 23:** Placa Arduino YUN.

### h) Arduino Micro (Ilustración 24):

Esta placa es igual que placa Arduino Leonardo, pero en formato más pequeña. Está basado en ATmega 32u4  Tiene 20 pines digitales de entrada-salida (de las cuales 7 se pueden utilizar como salidas PWM), 12 entradas analógicas. Un oscilador de cristal de 16 MHz Contiene una conexión micro USB, un conector ICSP Esta placa es diferente a todas las demás, porque incorpora en su interior la parte de comunicación USB, y no necesita un microprocesador secundario, por lo tanto cuando se conecta al ordenador se reconoce como un ratón o un teclado.

Las características de la placa son:

- Microcontrolador ATmega32u4, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas  entradas-salidas 40 mA.
- La Corriente máxima de los pines de Entrada/Salida de 3.3 V  50 mA.
- Memoria flash de 32 Kb
- SRAM 2.5 Kb, EERPROM 1 Kb y velocidad de reloj 16 MHz

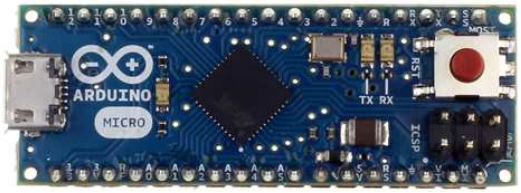



**Ilustración 24:** Placa Arduino Micro.

**i) Ethernet W5100 escudo (Ilustración 25):**

Con esta placa se puede conectar Arduino a internet.  Se basa en el chip W5100 WIZnet Ethernet, que proporciona una red (ip) de pila capaz de TCP y UDP, que soporta hasta cuatro conexiones de socket simultáneas.  Se puede añadir una ranura de tarjeta micro-SD, para almacenar archivos para servir a través de la red. Se adapta a todas las versiones de Arduino tarjeta principal de 2009, la ONU, mega 1280, Mega 2560. Tiene 14 entradas/salidas digitales (de las cuales 4 se pueden utilizar como PWM) y 6 entradas analógicas. Tiene reservado desde 10 hasta 13 pines por SPI, 4 utilizados por la placa SD, y 2 desde W5100

Las características de la placa son:

- Microcontrolador ATmega328, la placa funciona con 5 V.
- La tensión recomendada de entrada 7-12 V, siendo la tensión de entrada limites de 6-20 V.
- Corriente máxima para todas  entradas-salidas 40 mA.
- La Corriente máxima de los pines de Entrada/Salida de 3.3 V  50 mA.
- Memoria flash de 32 Kb
- SRAM 2 Kb, EERPROM 1 Kb y velocidad de reloj 16 MHz

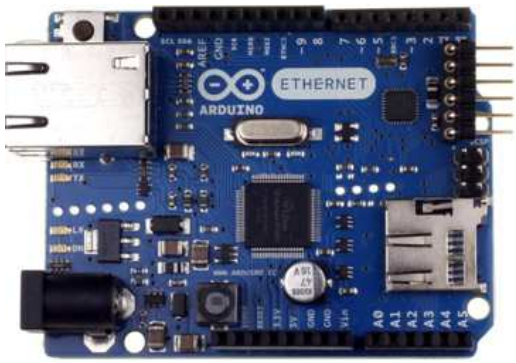



**Ilustración 25:** Placa Arduino Ethernet.

### 3.7.2 Shield Arduino:

Los shield son placas que se conectan directamente a la placa de Arduino, para expandir sus funcionalidades; como por ejemplo conexión de motores, generar una red WiFi, o muchos más shields que se verán a continuación y pueden ser de utilidad para el proyecto o expansión del mismo.

**a) Arduino GSM (Ilustración 26):**

Con este Shields se puede conectar Arduino a internet desde cualquier lugar, recibe y envía SMS, puede recibir y realizar llamadas de voz y de datos. Es compatible con Arduino Uno, mega y ADK, y con una pequeña modificación se puede utilizar para Leonardo. Se puede utilizar cualquier SIM de cualquier operadora.

Especificaciones Técnicas:

- Tensión de trabajo 5 V, suministrados desde la placa Arduino
- Conexión a través de GSM y GPRS
- Requiere una placa Arduino, y la comunicación con este es mediante serial Software.


**Ilustración 26:** Shield GSM.

### b) Arduino Wifi (Ilustración 27):

Con este Shield se puede conectar a internet sin cables, a una red Wifi. Incorpora una ranura micro SD y una conexión Micro-USB para la actualización de firmware Wifi. Contiene un conector R-SMA para ponerle una antena exterior.

Especificaciones Técnicas:

- Tensión de trabajo 5 V, suministrados desde la placa Arduino
- Conexión a través de redes 802.11b/g
- Tipos de cifrados WEP y WPA2
- Conexión con Arduino a través del puerto SPI a través de ICSP, por lo que es compatible con cualquier Arduino que tenga este puerto.
- Requiere una placa Arduino, y la comunicación con este es mediante serial Software.

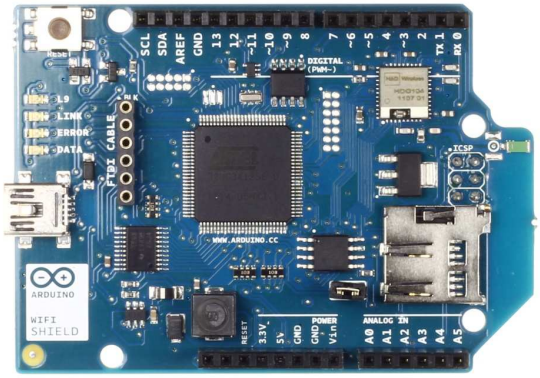



**Ilustración 27:** Arduino Wifi.

### c) Arduino motor:

La placa motor es un controlador de puente completo, diseñado para manejar cargas inductivas, en este caso para los motores. Se puede controlar con facilidad 2 motores DC la velocidad y dirección de forma independiente de cada uno. También se puede medir el consumo de corriente de los motores. Contiene 2 canales separados (A y B) que usa 4 pines de la placa Arduino para variar la velocidad, dirección de rotación, freno rápido, haciendo un total de 8 pines de utilización, pudiendo utilizar cada canal por separado para controlar 2 motores por separado.

Especificaciones Técnicas:

- Tensión de trabajo 5 V, suministrados desde la placa Arduino
- Corriente máxima por canal de 2 A, con un total de 4 amperios.

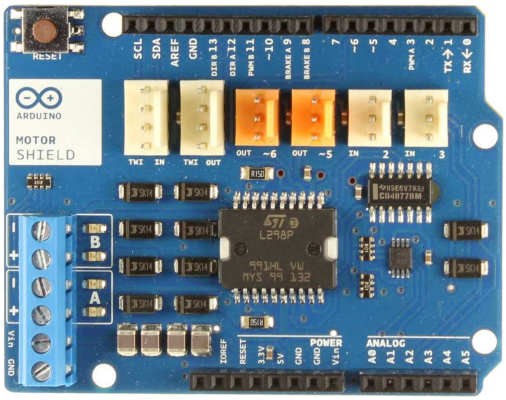



**Ilustración 28:** Shield Motor.

**d) Shield y módulos XBee:**

Son módulos para comunicación inalámbrica. Se pueden utilizar para conectar entre sí o crear redes de ordenadores, transmitir datos de un Arduino a PC, o para comunicar un Arduino a otro Arduino.

Para nuestro proyecto haría falta:

-1 Arduino XBee Shield **(Ilustración 29)**: Conecta el XBee serie 2 transmisor/receptor a la placa Arduino.


**Ilustración 29:** Arduino XBee Shield.

-1 XBee Explorer USB **(Ilustración 30)**: Conecta el XBee serie 2 transmisor/receptor al PC mediante conexión USB.

**Ilustración 30:** XBee Explorer USB.

-1 XBee Explorer Regulated **(Ilustración 31)**: traduce las señales de 5V a 3.3V, para poder conectar un sistema de 5V a cualquier módulo Xbee.

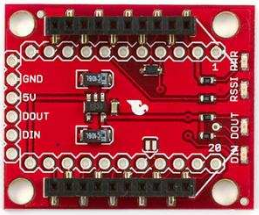



**Ilustración 31:** XBee Explorer Regulate.

-2 XBee serie 2 **(Ilustración 32)**: es el modulo transmisor/receptor, encargado de recibir o enviar datos.


**Ilustración 32:** Módulo XBee.

### e) Módulos nRF24LU1 y nRF201 (Ilustración 33)

Sirven para conectar el PC y Arduino sin cables y se envíen datos el uno con el otro. Trabaja en la Radio de 2.4GHz con una distancia de transmisión de datos de hasta 100m.

**Ilustración 33:** Módulos nRf24LU1 y nRF201.

### f) Módulos APC220 PC Kit-138 (Ilustración 34)

Se utiliza para la transmisión de datos entre PC y Arduino, instalando en el Arduino el modulo APC220 para que se comuniquen entre ellos. Tienen una distancia de transmisión de hasta 1200m.

**Ilustración 34:** Módulo APC220.

### g) Módulos Largo alcance RF Transmisor y receptor (Ilustración 35)

Trabajan en la frecuencia 315MHz. Para la utilización de estos módulos harían falta dos Arduinos para que se comunicaran entre sí, uno conectado al PC y otro conectado al robot.

El receptor corresponde a la ilustración 35 el módulo de abajo, y el receptor el de la parte de arriba.


**Ilustración 35:** Módulos RF largo alcance.

### j) Módulo puerto serie HC-06 Bluetooth (Ilustraciones 36 y 37).

Incorporando este módulo al Arduino, se consigue la conexión entre dispositivos Bluetooth, sin necesidad de cables, como puede ser comunicación con PC. El alcance óptimo aconsejado son máximo 10 metros.

**Ilustración 37:** Módulo HC-06 Bluetooth frontal.


**Ilustración 36:** Módulo HC-06 Bluetooth Trasero.

### k) Controlador puente H L298N (Ilustración 38)

Esta placa se utiliza para el control de velocidad, giro de motores y frenado rápido DC. Está diseñado para la construcción de coches inteligentes para Arduino.

Especificaciones técnicas:

- Tensión nominal: 3-15V
- Corriente nominal: 30 A
- Corriente de pico: 70 A
- Resistencia de los contactos: 0.005 ohm

- Frecuencia de resonancia: 1 k a 200 kHz

**Ilustración 38:** Módulo H L298N.

### 3.7.3 Sensores Arduino:

Los sensores de Arduino, ya vienen preparados para conectarlos directamente a nuestra placa Arduino, sin necesidad de soldar, aunque a veces a lo mejor es conveniente soldar para organizar mejor los elementos y que ocupen menos espacios. Existen multitud de sensores ya hechos para Arduino, como pueden ser de temperatura, humedad, detección de gases, luminosidad, etc. Para este proyecto se utilizará el sensor de distancia.

**a) Sensor de medición de distancias ultrasónico (HC-SR04, Ilustración 39):**

Este sensor realiza mediciones entre los rango 2cm hasta 450cm con un ángulo máximo de 15 grados. La medición la hace enviando una señal sonora, que al rebotar en algún objeto rebota, y al calcular el tiempo de demora mide la distancia a la que se encuentra.

Especificaciones técnicas:

- Voltaje de funcionamiento: 5V
- Corriente de trabajo: 15mA
- Frecuencia de trabajo: 40kHz
- Señal de salida: frecuencia de la señal eléctrica, 5V de alto nivel y de nivel bajo 0V
- El ángulo eficaz <15º
- Resolución: 0.3cm
- Medición de ángulo: 30º


**Ilustración 39:** Sensor Volumétrico HC-SR04.

### 3.7.4 Control de motores con Arduino:

Los controladores de motores permiten hacer un control de motores más preciso y a su vez proteger el microprocesador de corrientes elevadas que obtenemos al conectar motores. A continuación se describe el control para cada uno de los tipos de motores mencionado en puntos anteriores:

#### *3.7.4.1 Control Motor CC:*

Para poder controlar la velocidad y la dirección de los motores de corriente continua, se utiliza un circuito denominado puente H. el nombre viene dado por la forma en la que se colocaban los transistores al realizar el puente. Al avanzar la electrónica se ha producido la misma funcionalidad en espacios más reducidos.

Para el control de motores DC, se necesita un controlador de motores, ya que la placa Arduino no puede gestionar más de 20-40mA. Dicho controlador será controlado por la placa Arduino. Hay que elegir cuidadosamente la potencia de dicho controlador, según la potencia total de los motores. Un buen controlador de motores sería el doble puente H (L298N, **Ilustración 38**) con una potencia de 25W para las dos salidas que dispone.

#### *3.7.4.2 Control Motor PAP:*

Para el control de motores paso a paso con Arduino, se necesita un controlador (EasyDriver, **Ilustración 40**) para poder controlar de esta manera el motor. EasyDriver es compatible con cualquier dispositivo que pueda proporcionar pulsos de 5V. Con este controlador se maneja de manera muy sencilla el motor, ya que con sólo dos pines podemos controlar la dirección como el paso.

Las características de EasyDriver están basadas en un chip A3967 configurado para 8 pasos. Es compatible para motores bipolares de 4, 6 y 8 cables. La tensión de alimentación puede ser de 7V a 30V, cuanto mayor tensión mayor  par. La corriente es ajustable de 150mA/fase a 750mA/fase.

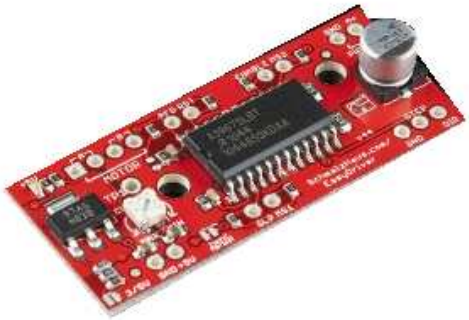


**Ilustración 40:** Placa control motores paso a paso.

#### *3.7.4.3 Control Servomotor:*

Para controlar un servomotor, se controlara con salidas PWM, que son salidas de señales cuadradas con una frecuencia normalmente de 50 Hz, donde se presentan dos estados (1 ó 0), si la onda cuadrada está más tiempo en 1, el servomotor va a atender hacia 180º, las salidas PWM de Arduino al ser de 8 bytes, solo permite girar 255º, pero realizando modificaciones se puede lograr los 360º. (**Ilustración 41**).

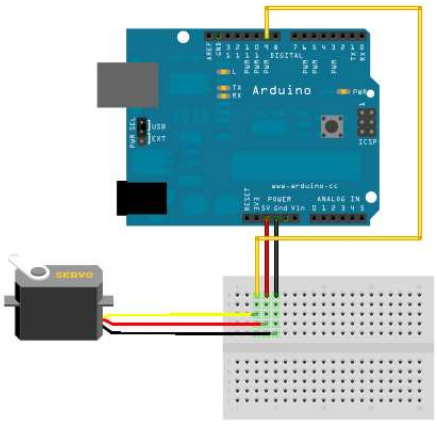



**Ilustración 41:** Esquema conexionado Servomotor.

## 3.8 Comunicación Inalámbrica:

Cuando la unión de los medios que se quieren comunicar no se hace mediante cables es una comunicación inalámbrica *[9]*. Las ventajas más destacadas de este sistema de comunicación es que no hace falta cableado, y su rapidez de la instalación.

Las técnicas que se utilizan son: *por Infrarrojos (IR)* o *por radiofrecuencia (RF).*

**Infrarrojos:** Los puntos de comunicación deben estar visibles, y no debe haber ningún obstáculo, la comunicación es entre pequeñas distancias y el campo de aplicación es limitado.

**RadioFrecuencia:** Permite comunicaciones de corto y medio alcance, pudiendo haber obstáculos y paredes. El campo de aplicación es muy grande. Para este proyecto se llevará a cabo este tipo de comunicación, por lo tanto será el que se describa detalladamente.

### 3.8.1 Tipos de Comunicaciones inalámbricas por RF:

La comunicación con transmisores de datos se están aplicando cada vez más, ya que son circuitos integrados que permiten hacer un diseño sin tener demasiados conocimientos de RF, ni disponer de costosa instrumentación para RF, ya que estos dispositivos requieren pocos componentes externos y ningún tipo de ajuste en RF *[4,14]*.

Primeramente se usaron módulos de RF con componentes discretos unidireccionales, precisamente para no tener que depender de un diseño en RF sin tener experiencia. Posteriormente aparecieron los transmisores completamente integrados con las funciones de emisor y receptor en diferentes bandas de frecuencia que se fueron estandarizando, permitiendo utilizar en diferentes campos, ya sea aplicación industrial, comercial, médico, etc.…

Una división de las comunicaciones inalámbricas puede ser las que no cumplen ningún protocolo estándar (llamadas propietarias) y las que cumplen un protocolo estándar, y por otro lado por las frecuencias de trabajo (las actualmente llamadas <1GHz, y las de

2.4GHz). Las <1GHz van desde 300 a 900 MHz y las de 2.4GHz que están normalizadas en todo el mundo, que definen velocidad de transmisión o ancho de banda *[16]*.

Hay que tener en cuenta que el rango de trabajo en RF depende de:

- Frecuencia
- Potencia de salida
- Sensibilidad de recepción
- Características de la antena
- Entorno de trabajo

Por ejemplo el alcance depende de la frecuencia de trabajo, a mayor frecuencia menor alcance, dependiendo también de lo dicho anteriormente.

### 3.8.2 Bandas ISM (Industrial, Scientific and Medical)

Las bandas ISM para sistemas de comunicaciones digitales inalámbricas que se emplean en la radiofrecuencia, son las que no necesitan licencia (siempre que esté dentro de unos márgenes de potencia) y son gratuitas en cuanto a la necesidad de usar protocolos normalizados.

Las frecuencias de trabajo estandarizadas por debajo de 1GHz son: 315 MHz en USA, 433 (potencia máxima +30dBm), 433 MHz (+10 dBm) y 868 MHz (+14 dBm) en Europa en AM o FM.

Como se puede ver en la **Ilustración 42,** para cada aplicación suele estar asignado a un rango de frecuencia.

| Rango de Frecuencia (MHz) | Aplicaciones | Potencia de Salida | Espacio entre canales | Ciclo de Servicio 0,1% | Ciclo de Servicio 1% | Ciclo de Servicio 10% | Ciclo de Servicio hasta 100% |
|---|---|---|---|---|---|---|---|
| 433.05 - 434.79 | Propósito general | 10 mW | - | | | | |
| 868.00 - 868.60 | Propósito general | 25 mW | - | | X | | |
| 868.60 - 868.70 | Dispositivos de alarma | 10 mW | 25 kHz | X | | | |
| 868.70 – 869.20 | Propósito general | 25 mW | - | X | | | |
| 869.20 – 869.25 | Dispositivos de alarma social | 10 mW | 25 kHz | X | | | |
| 869.25 – 869.30 | Dispositivos de alarma | 10 mW | 25 kHz | X | | | |
| 869.30 - 869.40 | Protocolo EACM | Sin definir | 25 kHz | | | | |
| 869.40 - 869.65 | Propósito general | 500 mW | 25 kHz | | | X | |
| 869.65 - 869.70 | Dispositivos de alarma | 25 mW | 25 kHz | | | X | |
| 869.70 - 870.00 | Propósito general | 5 mW | - | | | | X |

**Ilustración 42:** tabla de rango de frecuencias para diferentes aplicaciones *[16]*.

### 3.8.3 Tipos de Modulación Digital

Las formas básicas de modulación digital son ASK, OOK, FSK, PSK, *[16]*.

#### *3.8.3.1 Modulación por variación de Amplitud (ASK-OOK)*

La modulación por variación de amplitud ASK (Amplitude Shift Keying, **ilustración 43**) tiene por ventajas el sencillo diseño (menor coste) y el bajo consumo.

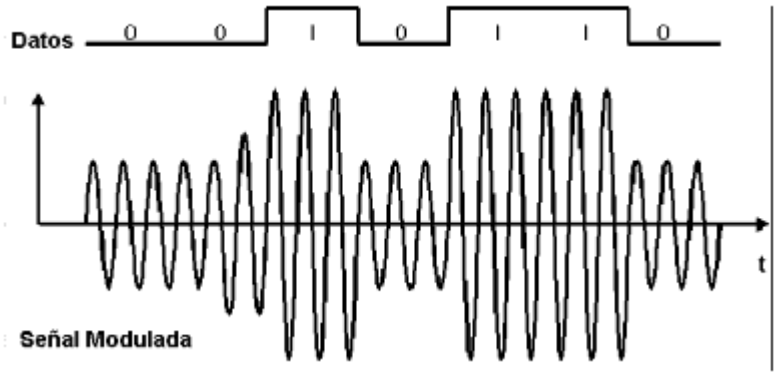



**Ilustración 43:** Señal modulada por variación de amplitud *[16]*.

Si se utiliza el método de modulación **OOK (On/Off Keying, Ilustración 44),** donde un 0 digital no hay potencia de salida y un 1 digital se entrega toda la señal portadora, consiguiendo una reducción mucho mayor del consumo. La desventaja son las interferencias de ruidos, pudiendo provocar errores en los datos recibidos.

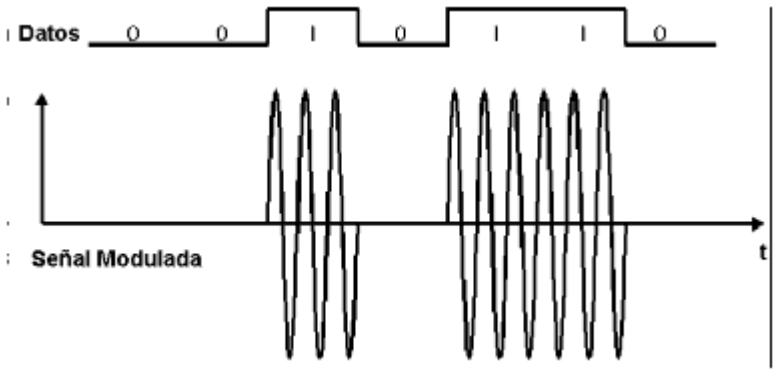



**Ilustración 44:** Señal Modulada On/OFF Keying *[16].*

### *3.8.3.2 Modulación por desplazamiento de Frecuencia (FSK)*

La modulación por desplazamiento de frecuencia (FSK-Frecuency Shift Keying, **ilustración 45**), con un 0 digital se transmite una onda portadora a una frecuencia y con un 1 digital se transmite otra portadora a otra frecuencia distinta, pero con la misma amplitud. Una de la gran ventaja de este sistema es la robustez ante interferencia de ruidos y la desventaja es la complejidad del sistema y el consumo que permanece siempre activo durante la transmisión.

Este sistema es el que se utiliza en los módems de baja velocidad. Se emplea separando el ancho de banda total en dos bandas, los módems pueden transmitir y recibir datos por el mismo canal, poniéndose en modo “llamada” o “respuesta” gracias a un conmutador que hay en cada módem.

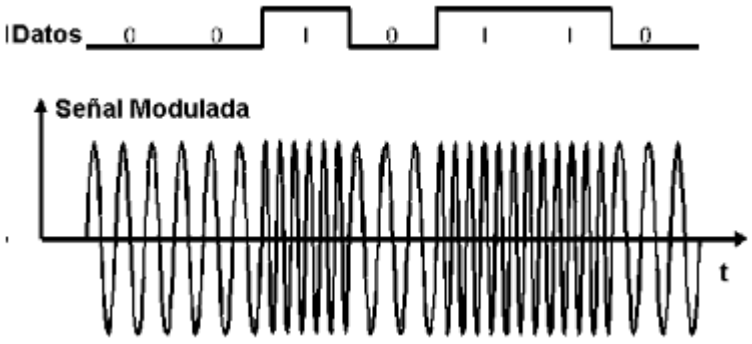



**Ilustración 45:** Señal modulada por desplazamiento de frecuencia *[16].*

#### *3.8.3.3 Modulación por Desplazamiento de Frecuencia Guasiana (GFSK)*

Este tipo de modulación es un tipo donde un 1 lógico es representado mediante una desviación positiva (incremento) de la frecuencia de la onda portadora, y un 0 mediante una desviación negativa (decremento) de la misma.

Esta modulación es una versión mejorada de FSK. En GFSK la información es pasada por un filtro guasiano antes de modular la señal, lo cual permite mayores velocidades de transferencia sobre un mismo canal.

#### *3.8.3.4 Modulación por Desplazamiento de Fase (PSK)*

En este sistema de modulación **(ilustración 46)** los valores binarios se codifican como cambios de fase de la señal portadora

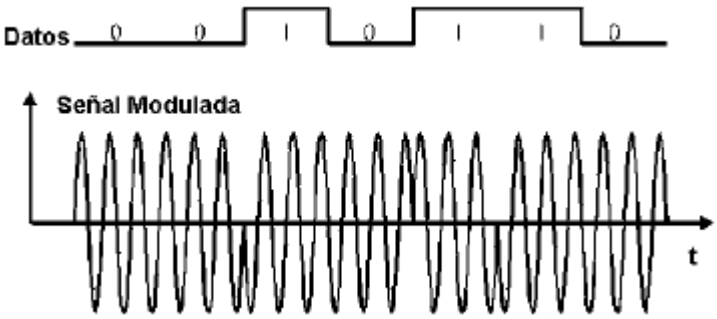



**Ilustración 46:** Señal modulada por desplazamiento de fase *[16]*.

Dentro del este sistema se distinguen dos tipos de modulación de fase:

- **Modulación PSK.** Que consiste en que cada estado de modulación está dado por la fase que lleva la señal respecto de la original.
- **Modulación Diferencial de Fase DPSK.** Cada estado de modulación es codificada por un salto respecto a la fase que tenía la señal anterior. Empleando este sistema, se garantizan las transiciones o cambios fase en cada bit, lo que facilita la sincronización del reloj en recepción.

#### 3.8.3.5 Modulación por desplazamiento de Fase en Cuadratura (QPSK)

Este sistema consiste en que el tren de datos a transmitir se divida en pares de bits consecutivos llamados Dibits, codificando cada bit como un cambio de fase con respecto al elemento de señal anterior.

#### 3.8.3.6 Modulación por desplazamiento de Fase en Múltiple (MPSK)

En este sistema, el tren de datos se divide en grupos de tres bits, llamados tribits, codificando cada salto de fase con relación a la fase del tribit que le precede.

La necesidad de velocidades más elevadas de transmisión de datos a hecho necesario implementar otro tipo de moduladores más avanzados como es la Modulación en Cuadratura. Este tipo de modulación presenta 3 posibilidades:

#### 3.8.3.7 Modulación de Amplitud en Cuadratura (QAM)

En este tipo de modulación, las ondas portadoras están moduladas en amplitud y el flujo de datos, divididos en grupos de 4 bits, y a su vez en subgrupos de 2 bits, codificando cada dibits en 4 estados de amplitud en cada una de las portadoras.

Es una combinación de PSK y ASK, es decir, se van a combinar las variaciones de amplitud en referencia al momento de fase en que ocurren, con lo cual vamos a poder incluir más bits en los mismos ciclos.

#### 3.8.3.8 Modulación de Fase en Cuadratura (QPM)

En este tipo de modulación las portadoras tienen 2 valores de amplitud. El flujo de datos se divide igual que en el caso anterior en grupos de 4 bits y a su vez subgrupos de 2 bits, modulando cada dibit en 4 estados de fase diferencial en cada una de las portadoras.

#### 3.8.3.9 Modulación de Fase y Amplitud en Cuadratura (QAPM)

Este tipo de modulación también llamado AMPSK o QAMPSK, debido a que es una combinación de los dos sistemas de amplitud y fase. Consiste en agrupar la señal en grupos

de 4 bits considerando 2 dibits, el primer dibits modula la portadora I en amplitud y fase mientras que le otro realiza lo mismo con la portadora Q.

#### *3.8.3.10 Modulación por División de Frecuencia Ortogonal (OFDM)*

En este tipo de modulación también conocido como 'Modulación por multitono discreto' (DMT), trabaja dividiendo el espectro disponible en múltiples subportadoras.

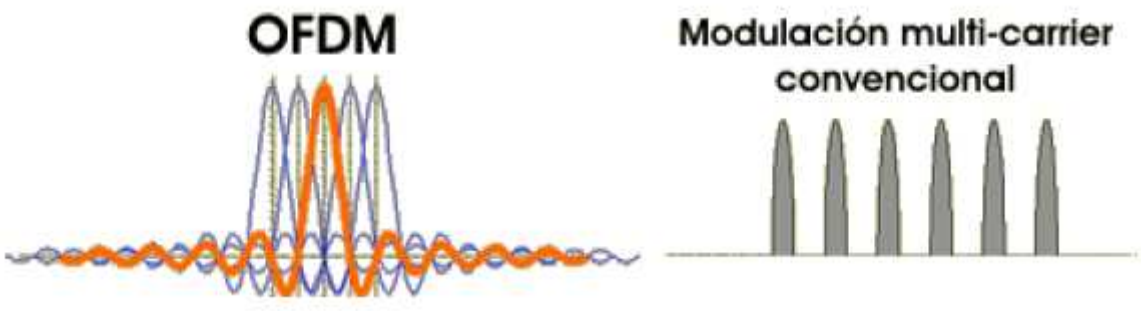



**Ilustración 47:** Modulación por división de frecuencia ortogonal *[16].*

### 3.8.4 Tipos de tecnologías empleadas en radiofrecuencia

En radiofrecuencia se emplean dos tipos de tecnologías, la **banda estrecha** y la **banda ancha** (también es conocido como espectro expandido) que utiliza todo el ancho de banda disponible, en lugar de utilizar una portadora para concentrar toda la energía a su alrededor.

#### *3.8.4.1 Espectro expandido por secuencia directa o DSSS (Direct Sequence Spread Spectrum).*

En este método, se genera un patrón de bits repetitivos para uno de los bits que componen la señal. Cuanto mayor sea esta señal, mayor será la resistencia de la señal a las interferencias. El estándar IEEE 802.11 recomienda un tamaño de 11 bits, pero el óptimo es de 100. En captación de señal es necesario realizar el proceso inverso para obtener la información original.

La secuencia de bits que se utiliza para modular los bits se llama como secuencia de *Barker* (también llamado código de dispersión o *PseudoNoise*). Es una secuencia rápida diseñada para que aparezca aproximadamente la misma cantidad de 1 que de 0. Un ejemplo es:

+1-1+1+1-1+1+1+1-1-1-1-1

Solo los receptores a los que el emisor haya enviado previamente la secuencia podrán recomponer la señal original. Además, al sustituir cada bit de datos a trasmitir, por una secuencia de 11 bits equivalente, aunque parte de la señal de transmisión se vea afectada por interferencias, el receptor aún puede reconstruir fácilmente la información a partir de la señal recibida.

#### *3.8.4.2 Espectro expandido por salto en frecuencia o FHSS (Frecuency Hopping Spread).*

Esta tecnología consiste en enviar una parte de la información en una determinada frecuencia durante un determinado tiempo. Pasado este tiempo se cambia la frecuencia de emisión y se sigue transmitiendo a otra frecuencia. Por lo tanto cada tramo de información se va transmitiendo en una frecuencia diferente durante un intervalo pequeño de tiempo.

### 3.8.5 Indicador de la potencia de señal recibida (RSSI).

RSSI (Receive Signal Strength Indication) es una medida de la potencia presente en los receptores. RSSI es la tecnología métrica de receptor de radio genérica, esto es invisible al usuario del dispositivo que contiene el receptor, pero los usuarios de redes inalámbricas lo conocen como protocolo IEEE202.11. La salida RSSI es a menudo un nivel análogo de corriente continua.

### 3.8.6 Indicador de la potencia de señal recibida (LQI).

LQI (Link Quality Indication) es una medida de la cantidad de la señal recibida. Estima la facilidad de demodular una señal recibida por acumulación de la magnitud del error entre constelaciones ideales y la señal recibida sobre los 64 símbolos inmediatos después de la palabra de sincronización. LQI es la mejor medida relativa de la calidad de un enlace (un valor alto indica un mejor enlace), ya que el valor es dependiente del formato de la modulación.

## 3.9 Reconocimiento de voz:

### 3.9.1 Introducción al reconocimiento de voz:

El habla es una de las partes más importantes de la expresión humana, es algo que nos diferencia del resto de seres vivos en el planeta, ya que sin el habla el pensamiento del hombre no sería posible.

Dada la importancia del habla y lo imprescindible que es, el presente proyecto pretende crear una interacción entre unas de las expresiones esenciales del hombre con el robot, por medio del computador, creando así un Sistema de Reconocimiento de Voz.

El procesamiento digital de señales de voz *[10]* tiene una gran variedad de aplicaciones. Existen funciones para el tratamiento digital de señales, que puede ser implementadas para lograr obtener lo que nos interese según la aplicación.

El Sistema de Reconocimiento de Voz *[26]* es una aplicación del procesamiento digital de señales de voz que se encarga de obtener una señal de voz que permita reconocer que palabra se está hablando. Consta de una interfaz gráfica que permite la interacción del ser humano por medio de un micrófono con la computadora, lo que procesa los datos adquiridos por el microfono automáticamente. Una vez procesados los datos ya se puede utilizar a la libre imaginación, en nuestro caso para el control del robot móvil.

### 3.9.2 Componentes del sistema de reconocimiento de voz:

Un sistema de reconocimiento de voz realizado en Matlab suele estar compuesto por los siguientes componentes *[1, 17, 19]*:

#### *3.9.2.1 Micrófono:*

*Un micrófono es un transductor electroacústico, que su función es transformar o traducir la presión acústica ejercida sobre la capsula por las ondas sonoras en energía eléctrica. La calidad de cada micrófono viene dada por sus características [28].*

#### *3.9.2.2 MATLAB:*

Matlab *[27]* es el nombre abreviado de "MATrix LABorator". Es un lenguaje de alto nivel y de ambiente interactivo que permite realizar tareas intensivas y con una mayor velocidad que los lenguajes de programación comúnmente usados.

MATLAB está especializado para cálculos numéricos con vectores y matrices, como casos particulares, puede trabajar también con otras estructuras de información.

El lenguaje está construido por código llamado M-code que puede ser fácilmente ejecutado en la ventana de comandos. Con lo cual se pueden crear funciones, etc. Pero la razón principal para la elección de este lenguaje de programación son las herramientas que proporciona para el procesamiento de señales, y el conjunto de funciones para el procesamiento digital.

Además con MATLAB también se puede crear entornos gráficos utilizando la GUIDE, que provee herramientas para crear GUIs, 'Grapihical User Interface' con lo cual se puede crear la forma del entorno gráfico así como asociar funciones a los elementos de GUI. MATLAB también incluye funciones para manipular archivos.

#### *3.9.2.3 Señal de voz:*

La voz humana se produce por medio del aparato fonatorio. Este está constituido por los pulmones como fuente de energía, en forma de flujo de aire, la laringe que contiene las cuerdas vocales, la faringe, las cavidades oral y nasal y una serie de elementos articulatorios: los labios, los dientes, el alvéolo, el paladar, el velo del paladar y la lengua.

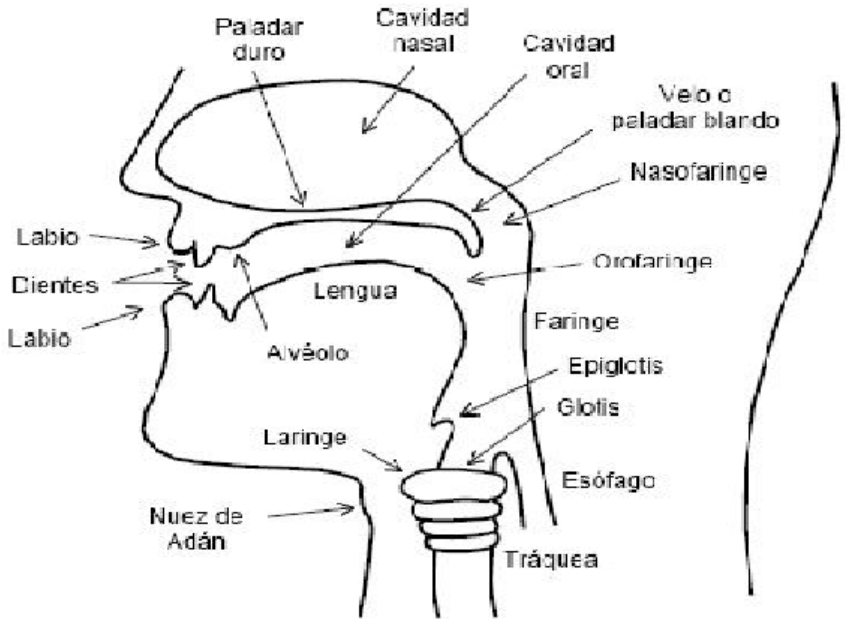



Ilustración 48: Aparato fonatorio humano *[17].*

En el habla los formantes son elementos que sirven para distinguir los componentes del habla humana, principalmente, las vocales y sonidos sonantes. El formante con la frecuencia más baja es llamada como F1, el segundo F2, etc. Normalmente sólo los dos primeros son necesarios para caracterizar una vocal y generalmente, los formantes posteriores constituyen propiedades acústicas como el timbre.

Los dos primeros formantes se determinan principalmente por la posición de la lengua. Siendo que F1 tiene una frecuencia más elevada cuando más baja esta la lengua y para el F2 tiene mayor frecuencia cuanto más hacia delante esta posicionada la lengua.

Si la frecuencia fundamental es más grande que la frecuencia de los formantes, entonces el carácter del sonido se pierde y se vuelven difíciles de reconocer.

En la siguiente tabla se puede ver algunos anchos de banda entre los cuales se localizan las vocales:

| Formantes Vocálicos | |
|---|---|
| Vocal | Región principal formantica |
| /u/ | 200 a 400 Hz |
| /o/ | 400 a 600 Hz |
| /a/ | 800 a 1200 Hz |
| /e/ | 400 a 600 y 2200 a 2600 Hz |
| /i/ | 200 a 400 y 3000 a 3500 Hz |

Ilustración 49: Ancho de banda de las vocales *[17].*

### 3.9.3 Procesamiento de la señal de voz:

Lo primero que debe realizar un reconocedor es procesar la señal de voz *[10, 17]* de entrada del sistema, con la finalidad de extraer la información acústica relevante para la tarea que se desea realizar. Las características que se deben extraer de la señal de voz, son el resultado de un largo proceso de estudio e investigación sobre diferentes procedimientos de parametrización de la voz, planteándose en la mayoría de las situaciones una parametrización espectral que incluye consideraciones preceptúales a partir del funcionamiento del oído.

La señal de voz de entrada puede venir distorsionada por efectos perturbadores, los cuales se desea que sean eliminados, para lograr que la señal no llegue perturbada se ha generado tres líneas generales de trabajo *[10]*:

- **Detección robusta de voz:** Para esto hay innumerables procedimientos de diferenciación entre voz o ruido (silencio) para diferentes tipos de ruido.
- **Reducción de ruido:** hay procedimientos que trabajan directamente sobre la señal de voz y procedimientos que buscan disminuir el efecto del ruido sobre la parametrización de la voz.
- **Cancelación de ecos:** construyendo técnicas de filtrado que se adapten y que permita al usuario comenzar a hablar mientras, desde el PC, se le pueda comunicar un mensaje que puede producir un eco en la voz que entra al reconocedor.

### 3.9.4 Pre-Procesamiento de la señal de voz:

Es necesario un Pre-Procesamiento de la señal de voz. Esto se hace mediante técnicas que permiten obtener la información acústica directamente a partir de la señal de voz hablada.

#### *3.9.4.1 Filtro para la señal de voz:*

Primeramente se realiza un filtrado de la señal para reducir o eliminar los efectos del ruido que acompaña la señal de voz guardada. Al diseñar el filtro se deberá de tener en cuenta el hecho que la elección de las frecuencias de corte y paso de este, podrían no reconocer voces espectralmente parecidas.

#### *3.9.4.2 Segmentación de la señal de voz:*

La segmentación determina los puntos de inicio y fin de la señal para eliminar el ruido, es decir, se debe detectar y diferenciar las partes de señal que llevan información de voz con las que no llevan voz. Este procedimiento ahorra gasto de memoria y tiempo de cálculo en las tramas que no contienen información de la voz para así no obtener resultados erróneos en el análisis de las señales de voz.

De la correcta división de la señal depende en gran medida el perfecto proceso de reconocimiento. De hecho, los errores en la segmentación de la señal constituyen una principal fuente de error en los sistemas de reconocimiento de voz, ya que algunos sonidos que puedan entrar en el sistema, correspondientes a ruido, podrían en ocasiones confundirse con voz. Detectar el inicio y fin de palabra también se realiza con el fin de evitar cálculos innecesarios, así solo se procesa las partes de la señal que corresponden a voz.

#### *3.9.4.3 Ventaneo de la señal de voz:*

En esta sección se encarga que la señal filtrada y sin silencios se hagan N muestras, donde N es un valor que se escoge teniendo en cuenta que para realizar el análisis de Fourier en tiempo corto, la señal de voz estacionaria a “trozos” es condición necesaria para dicho análisis. El intervalo de tiempo que se considera que la señal es estacionaria depende de la velocidad de cambios del tracto vocal y de las cuerdas vocales. Normalmente este intervalo de tiempo se establece en un valor de 20 y 40 ms. Estas N muestras se pasarán por una ventana que en general es una ventana de Hamming. La ventana se define como: $\boldsymbol{wnT} =$

$\mathbf{0.54 - 0.46cos2nN}$ $\mathbf{0 \leq n \leq N}$ donde *"N"* es el tamaño de la ventana o sea las muestras y *"n"tiempo que dura una trama.*

*Una ventana es una función matemática con frecuencia en el análisis y el procesamiento de la señal para evitar las discontinuidades al principio y al final de los bloques analizados. En procesamiento de señales, una ventana se utiliza cuando interesa una señal de longitud voluntariamente limitada [28].*

Como la ventana de Hamming atenúa mucho la señal de los limites de dicha ventana se hace un solapamiento cada 10 o 20 ms, con esto se consigue que no se pierda información útil.

En el reconocimiento de habla, la señal de voz pre-procesada se lleva a un nuevo procesamiento para producir una secuencia de vectores que representa la voz, estas secuencias se denominan parámetros, que deben representar la información contenida en la envolvente del espectro. Cuanto menor sea el número de parámetros mayor rapidez del sistema, ya que si el número es mayor puede colapsarse la base de datos, y a más parámetros menos fiable son los resultados.

Existen varios métodos para la extracción de características, y se concentran en diferentes espectros representativos. Para estos casos se analizarán los dos de mayor importancia para el análisis de la voz:

- Análisis de predicción lineal (LPC)
- Análisis cepstral.

### 3.9.5 Parametrización de la señal de voz:

Como ya se ha comentado anteriormente, la correcta parametrización es muy importante realizarla bien, ya que permitirá una mayor robustez del sistema y una mayor fiabilidad. Con una buena parametrización el sistema será más rápido. A continuación se verán los métodos para la extracción de características.

#### *3.9.5.1 Coeficientes de predicción lineal (LCP):*

Esta técnica es una de las más utilizadas y más potentes de análisis de voz, y uno de los métodos más útiles para codificar voz con buena calidad y conseguir un reconocimiento más fiable. LCP representa la envolvente espectral de una señal digital de voz en una forma comprimida, utilizando la información de un modelo lineal, lo que hace una aproximación muy precisa a los parámetros de la voz.

Su funcionamiento es hacer un modelo de filtro de tipo todo polo, para la fuente de sonido, que permite describir la función transferencia de un tubo, que sin pérdidas está formado por diferentes secciones. El modelo se llama así porque pretende extrapolar el valor de la siguiente muestra de voz s(n) como la suma ponderada de muestras pasadas s(n-1), s(n-2),…, s(n-K) y $\alpha_k$ son coeficientes de predicción lineal.

$$s(n) = -\sum_{k=1}^{p} \alpha_k s(n-k)$$

#### *3.9.5.2 Cepstrum:*

Los sonidos de la voz se pueden representar por un espectrograma, que indica las componentes frecuenciales por la que esta compuesta la señal de voz. De esta manera el espectro nos proporciona información acerca de los parámetros del modelo de producción de voz, tanto de la excitación como del filtro que representa el tracto vocal.

##### 3.9.5.2.1 Modelo de obtención de los coeficientes cesptrales:

MATLAB tiene una función para la obtención de los coeficientes cepstrales utilizando FFT. La función es la ***rceps,*** que proporciona el cepstrum real de la función ingresada, por medio del algoritmo mostrado a continuación:

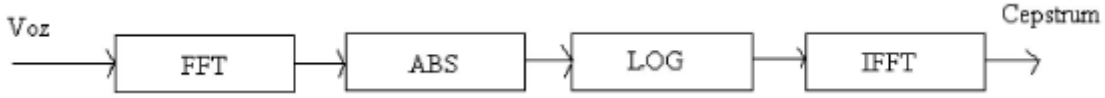



Ilustración 50: Modelo Coeficientes Cesptrales.

La principal razón por la cual se utilizan los coeficientes cepstrales es que tienen la ventaja que no importa las distorsiones que puedan ser introducidas por el micrófono, ya que se puede derivar una serie de parámetros que son invariantes y no afectan dichas distorsiones.

#### 3.9.5.2.2 Coeficientes cepstrales de frecuencia Mel: (MFCC).

Los coeficientes llamados mel-cepstrum o MFCC, son de gran utilidad en la extracción de parámetros de la señal de voz. Los filtros que se utilizan están espaciados linealmente para frecuencias menores de 1000 Hz y logarítmicamente para frecuencias mayores de 1000 Hz, con el objetivo de captar características fonéticamente importantes del habla. A esta escala se le denomina "Escala de MEL" y su fórmula matemática es la siguiente:

$$Mel(f) = 2595 \cdot log\left(1 + \frac{f}{700}\right)$$

Donde "$f$" es la frecuencia.

Los pasos necesarios para el cálculo de los MFCC son:

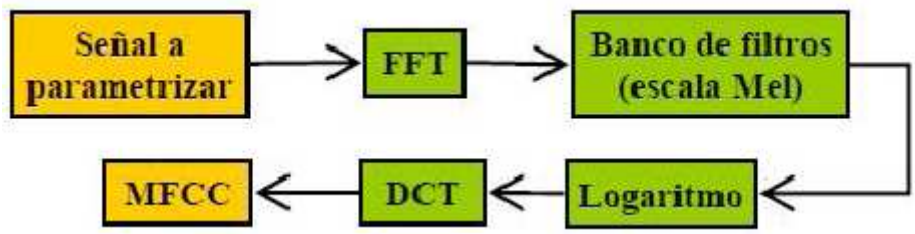



**Ilustración 51:** Proceso cálculo de MFCC

Primeramente calcula la transformada de Fourier de tiempo corto para cada una de las tramas conseguidas de la etapa de pre-procesamiento mediante la ecuación:

$$X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n}$$

Donde: $\omega = \frac{2\pi}{N} \cdot k$ es la función ventana. $x[n]$ es la señal a ser transformada. $X(m,\omega)$ es esencialmente la Transformada de Fourier de $x[n]w[n-m]$ que representa la fase y magnitud de la señal sobre tiempo y frecuencia. El índice de tiempo *m* normalmente se

considera un tiempo *"lento"* y usualmente no se expresa con tan alta resolución como el tiempo *n*. $w[n]$ es la ventana. En el caso de *m* es discreta y en caso de $\omega$ es continua.

El banco de filtros linealmente espaciado en la escala de Mel pueden ser triangulares o tener otras formas, tales como Hamming, Hanning o Kaiser, pero el triangular es el más utilizado.

### 3.9.6 Reconocimiento del habla. Decisión:

Las técnicas de parametrización explicadas anteriormente tienen como objetivo generar una serie de coeficientes que representan las características de la señal de voz, para su posterior utilización en la fase de reconocimiento del habla. La matriz obtenida en el proceso de parametrización depende directamente de la longitud de la señal de voz, cuanto mayor sea la señal mayor tamaño tendrá la matriz. Para usar estas matrices en el reconocimeinto del habla el tamaño de éstas tiene que ser el mismo, por lo tanto se tiene que estandarizar la matriz que contiene los coeficientes cepstrales.

El primer paso que se realiza en el reconocimiento del habla es la estandarización de la matriz de coeficientes cepstrales y se denomina **Cuantización Vectorial.** El paso siguiente corresponde al cálculo de la diferencia entre la señal de voz del hablante y las señales que se encuentran en la base de datos anteriormente guardadas (patrones); dicha diferencia se obtiene mediante el cálculo de la Distancia Euclidiana en varias dimensiones. A continuación se explican los procedimientos.

#### *3.9.6.1 Cuantificación Vectorial de la señal de voz:*

En cualquier tipo de procesamiento es importante la optimización de los algoritmos en cuanto a velocidad y almacenamiento, por lo tanto con la cuantificación de vectores se consigue clasificar un conjunto de vectores de los cuales luego se buscarán los mejores representantes para reducir el tamaño de la información a manejar. El objetivo de un cuantificador es obtener un conjunto de vectores representativos llamado codebook, que presente el menor error por distorsión, por ejemplo para cuantificar los vectores de observación.

- **Componentes de un cuantificador vectorial** *[26]***:**

  Para diseñar un cuantificador vectorial se necesita:

  1. Un gran número de vectores de patrones V1, V2,…, Vn, que conforman el grupo de entrenamiento. El grupo de entrenamiento se utiliza para crear el grupo de vectores del codebook que representa la variabilidad espectral observada en el grupo de entrenamiento.

  2. Una medición de distancia entre los vectores de entrenamiento y el vector a reconocer para agrupar el conjunto arbitrarios a cada entrada del codebook.

  3. Un procedimiento de ordenamiento para ubicar y calcular los centroides. Sobre la base del particionamiento que clasifica el grupo de n vectores en *M* clústeres o sectores primero elegimos el *M* codewords del codebook, para luego proceder a la clasificación.

- **Vectores de Observación :**

  Los vectores de observación son vectores que contienen las características que describe la voz humana. Estos vectores son obtenidos al final de los distintos pasos para el tratamiento de la señal de voz.

- **Clasificación de Vectores:**

  Este módulo agrupa una cantidad de vectores característicos, *N*, en una cantidad *M* (MDN), discreta, de sectores o celdas de clasificación logrando que las características albergadas en cada sector sean similares.

- **Algoritmos de Clasificación:**

  Los *N* vectores originales de tamaño *D* quedan representados por *M* vectores, cada uno es llamado “palabra de código” o codeword, el grupo entero de dichos vectores, forma un “libro de códigos” o codebook, quedan entonces delimitadas *M*

regiones o sectores, llamados regiones de Voronoi. Existen Varios algoritmos de clasificación de vectores como son *Algoritmo K-Means, Algoritmo LGB [26]*.

- **Utilización del cuantificador y del codebook:**

  Cuando se tiene construido el codebook, el procedimiento para cuantificar vectores es básicamente realizar una búsqueda por todo el codebook para encontrar el mejor representante. Un procedimiento de cuantificación para señal de voz es elegir el vector más cercano del codebook al vector de observación y utiliza ese vector denominado codeword, como la representación resultante para etapas posteriores, dando como respuesta, a su salida, el vector que mejor representa ese entrada.

- **Distancia Euclidiana:**

  Se emplea para el reconocimiento del habla para calcular la diferencia existente entre el vector caracteristico obtenido de la palabra dicha por el hablante y el resto de vectores carateristicos almacenados en la base de datos de entrenamiento del sistema. El resultado final es un valor numérico que representa la distancia entre dos matrices de dimensiones igules. El vector característico de los patrones cuya distancia al vector caracteristico generado por la palabra dicha por el hablante sea menor que el nivel de comparación establecido, identifica la palabra con la que tenga mayor semejanza. La fórmula que se utiliza para el cálculo de la distancia euclidiana en dos o más dimensiones es la siguiente:

$$d(v,w) = \sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2 + (v_3 - w_3)^2 + \cdots + (v_n - w_n)^2}$$

$$donde\ v = [v_1, v_2, v_3 \ldots, v_n]\ y\ w = [w_1, w_2, w_3 \ldots, w_n] son\ matrices\ de\ longuitud\ n$$

## 3.10 Lógica Difusa:

Este tipo de lógica es una lógica aplicada a conceptos que pueden tomar un valor indeterminado de veracidad dentro de un conjunto de valores cuyos límites son la verdad absoluta o la falsedad absoluta. Esta lógica permite trabajar con información que es

imprecisa y no está bien definida y nos permite introducir valores intermedios entre la afirmación completa o la negación absoluta *[15, 18, 20]*.

La lógica difusa *[7, 8, 25]* fue investigada por primera vez alrededor de mediados de los años sesenta por el ingeniero *Lotfy A. Zadeh* en la Universidad de Berkeley (California). El describió así este principio: "*Conforme la complejidad de un sistema aumenta, nuestra capacidad para ser precisos y construir instrucciones sobre su comportamiento disminuye hasta el umbral más allá del cual la precisión y el significado son características excluyentes.*"

En ese momento fue cuando *Lotfy A. Zadeh* introdujo el concepto de conjunto difuso (Fuzzy Set). Este concepto quiere decir que el pensamiento humano se basa en etiquetas lingüísticas y no en números.Es decir, que no solo se trabajará con números, si no que se trabajará con términos lingüísticos que aunque son más imprecisos que los números, muchas veces son más fáciles de entender para el razonamiento humano.

Para entender mejor de que se trata la lógica difusa se explicará un ejemplo que hizo *Zadeh "los hombres altos" [8]*. Según la teoría de lógica clásica, al conjunto de hombres altos solo pertenecen los que miden más de una determinada altura, siendo 1.80 metros el límite, un hombre que mida 1.81 metros será algo, pero sin embargo si mide 1.79 será bajo. Esto no es muy lógico para catalogar a un hombre alto y bajo, ya que un hombre bajo de uno alto para el caso expuesto solo se lleva 2 centímetros. Hay casos en los que no es fácil catalogar algo y se introduzca la lógica borrosa. Según esta lógica, no tiene un límite claro que indique que pertenece a un grupo u otro. El evaluar si un hombre es alto o bajo, se hace mediante una función que define la transición entre alto a bajo y para ellos asigna a las distintas alturas un valor entre 0 y 1. Según sea este valor se considera que se pertenece al conjunto o no. Aplicando esto al caso anterior un hombre que mida 1.79 metros se puede decir que pertenece al conjunto de hombres altos con un grado de 0.75 y el hombre que medía 1.81 metros pertenece al conjunto de hombres altos con un grado de 0.8. Si representamos esto en una gráfica se obtendrá que la transición entre alto o bajo con la lógica borrosa es una curva con cambios no abruptos **(Ilustración 52)** mientras que con la lógica clásica, el paso de alto a bajo o viceversa es brusco **(Ilustración 53)**:

En los conjuntos difusos la función de pertenencia puede tomar valores del intervalo entre

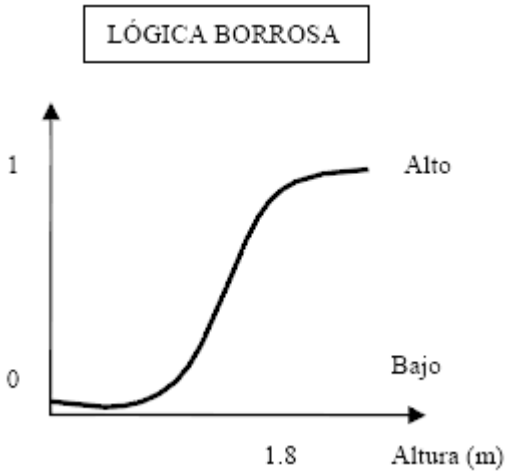



**Ilustración 52:** Gráfica de pertenencia log. Difusa *[8]*.

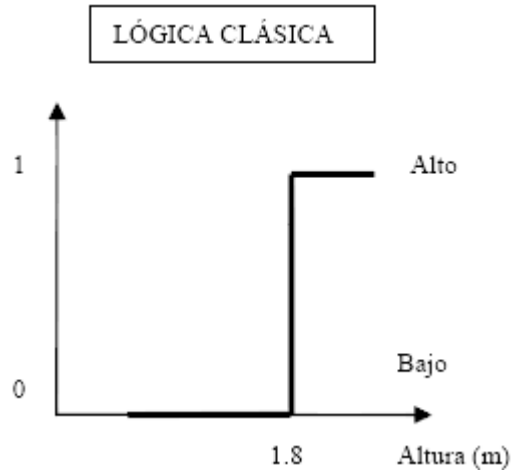



**Ilustración 53:** Gráfica de pertenecía log. Clásica *[8]*.

0 y 1, y la transmisión del valor entre cero y uno es gradual y no cambia de manera instantánea como pasa en los conjuntos clásicos. Un conjunto difuso en un universo en discurso puede ser:

$$A = \{(x, \mu_A(x)) | x \in U\}$$

Donde µA(x) es la función de pertenencia de la variable x, y U es el universo en discurso. Cuando más cerca esté la pertenencia del conjunto A el valor de 1, mayor será la pertenecía de la variable x al conjunto A, esto se puede ver en la **Ilustración 54**.

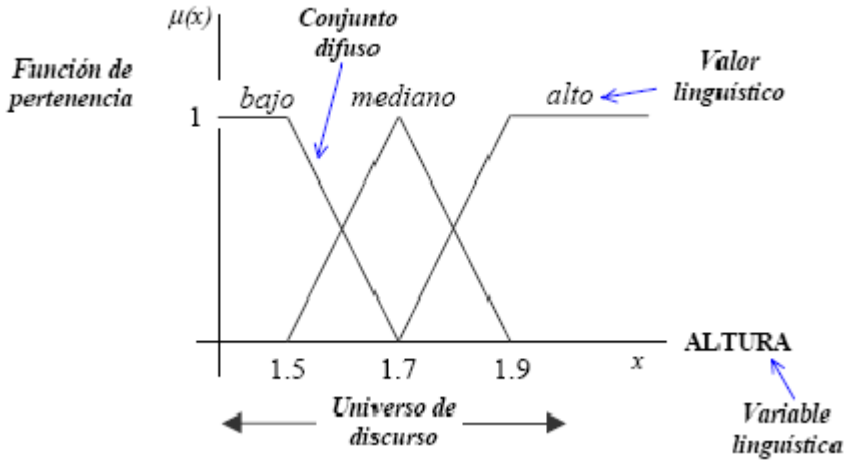
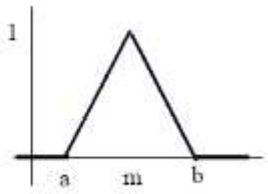



**Ilustración 54:** Gráfica de funciones de pertenencia *[7]*.

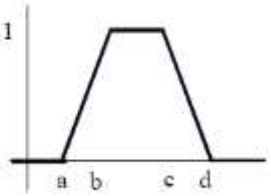
El grado de pertenencia se define mediante la función característica asociada al conjunto difuso, para cada valor que puede tomar la variable x, la función característica μA(x) proporciona el grado de pertenencia de ese valor x al conjunto difuso A.

Se puede utilizar cualquier función para definir un conjunto difuso, pero los más utilizados por su simplicidad matemática son las de tipo triangular y las trapezoidales:

$$\mu(x)=\begin{cases}0 & \text{para } x\le a\\ \dfrac{x-a}{m-a} & \text{para } a<x\le m\\ \dfrac{b-x}{b-m} & \text{para } m<x\le b\\ 0 & \text{para } x>b\end{cases}$$




$$\mu(x)=\begin{cases}0 & \text{para } x\le a\\ \dfrac{x-a}{b-a} & \text{para } a<x\le b\\ 1 & \text{para } b<x\le c\\ \dfrac{d-x}{b-c} & \text{para } c<x\le d\\ 0 & \text{para } x>d\end{cases}$$




Un sistema basado en lógica difusa siempre está compuesto por los siguientes bloques *[8]* **(Ilustración 55)**:

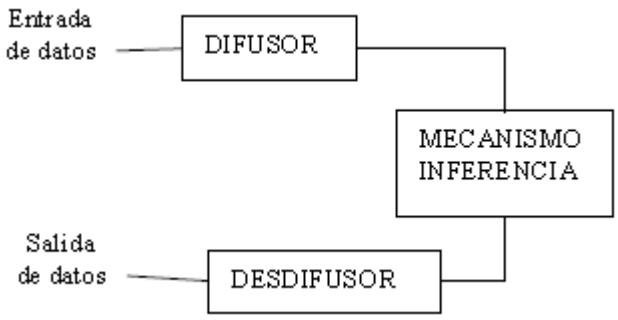



**Ilustración 55:** Esquema composición de un sistema de lógica difusa *[8]*.

- BLOQUE DIFUSOR: este bloque es el encargado de asignar un grado de pertenencia a cada dato de entrada a cada uno de los conjuntos difusos considerados mediante la función característica.

- BLOQUE DE INFERENCIA: relaciona conjuntos difusos de entrada y de salida y representa a las reglas que definen el sistema. Según el grado de pertencia de cada conjunto difuso se activará unas reglas u otras.

- DIFUSOR: se obtiene un resultado concreto a partir de los conjuntos difusos procedentes de la inferencia, mediante la aplicación de métodos matemáticos de desdifusión.

Existe variedad de tipos de reglas, donde se pueden distinguir dos grandes grupos, las **reglas difusas de Mamdani** y las **reglas difusas de Takagi-Sugeno** (TS, para abreviar).

**-Reglas difusas de Mamdani:**
if X is $A_1$ and y is $B_1$ then Z is $C_1$
if X is $A_2$ and y is $B_2$ then Z is $C_2$
Donde tanto A como B son conjuntos difusos, X son los atributos del sistema (entrada) y Z son los atributos controlables del sistema (salida)

**-Reglas modelo Takagi-Sugeno:** este modelo utiliza una función como consecuente:
if X is $A_1$ and y is $B_1$ then Z =f(x,y)
donde Z=f(x,y) es una función exacta en el consecuente y f(x,y) es un polinomio.

Las ventajas de las **reglas Mamdani** es que son intuitivas, tienen una amplia aceptación y están bien adaptadas a la incorporación de conocimiento y experiencia. Las ventajas de las **reglas Takagi-Sugeno** son que es computacionalmente eficiente, trabaja bien con técnicas lineales, con técnicas de optimización y control adaptable, tiene garantizada una superficie de control continua y está bien adaptado al análisis matemático.

Para el diseño de un controlador basado en lógica difusa debe tener un compromiso entre diversos criterios de diseño: velocidad, precisión y flexibilidad.

Para que el sistema pueda conseguir los resultados deseados debe tener una velocidad adecuada para que no haya mucho tiempo de respuesta. Si deseamos una alta precisión en el control se necesitará una gran cantidad de conjuntos para cada variable y un alto número

de reglas, lo que aumentará el tiempo de respuesta al tener un mayor número de operaciones. Por lo tanto habrá que alcanzar un equilibrio entre diversos criterios.

Para el campo de los sistemas basados en lógica borrosa se han introducido diversos sistemas de desarrollo de propósito específico. Uno de los más utilizados es Matlab y el que se utilizará en este proyecto.

Matlab es probablemente el entorno de desarrollo matemático más extendido para las aplicaciones de control y procesamiento de señal, especialmente en el ambiente universitario, donde se puede utilizar para la simulación de control de sistemas. En los últimos años se ha incorporado el Neural Networks Toolbox y el Fuzzy Logic Toolbox, para el trabajo con redes neuronales y lógica difusa, respectivamente, con relativa sencillez y buenas salidas gráficas.

Algunas aplicaciones de los sistemas difusos pueden ser en el **área médica**, empleándose para diagnósticos, acupuntura, análisis de ritmos cardiacos, o de arterioestenosis coronaria. Otro campo principal es el de **control**, a partir del empleo de las expresiones de la lógica difusa para implementar reglas orientadas al control de sistemas. En el campo de **control de sistemas en tiempo real** destaca el control de un helicóptero por órdenes pronunciadas de viva voz (Sugeno), y el control con derrapaje controlado de un modelo de coche de carreras de Altrok. Dentro del **sector del automóvil** existen gran número de patentes sobre sistemas de frenado y cambios de marcha automáticos. En el **área de aparatos de consumo** como pueden ser los nuevos aires acondicionados, donde la temperatura es regulada por lógica difusa. Para el **control de maquinara** el más destacado es el frenado de metro de Sendai (Japón), realizado por Hitachi, también se han implementado la lógica difusa para el control de ascensores.

En el **área de los robots móviles y navegación autónoma** es muy interesante aplicar la lógica difusa, como puede ser para controlar la dirección, seguimiento de un objeto y detención del robot a una distancia prefijada del objeto (robot autónomo Pilar), o bien para controlar la velocidad.

### 3.10.1 Toolbox fuzzy de Matlab:

Matlab *[27]* tiene disponible multitud de utilidades, toolboxes de control especificas. Entre ellas se encuentra la toolbox fuzzy, que permite definir un controlador difuso.

Para acceder a él, en el editor se pone *fuzzy:* **(Ilustración 56)**

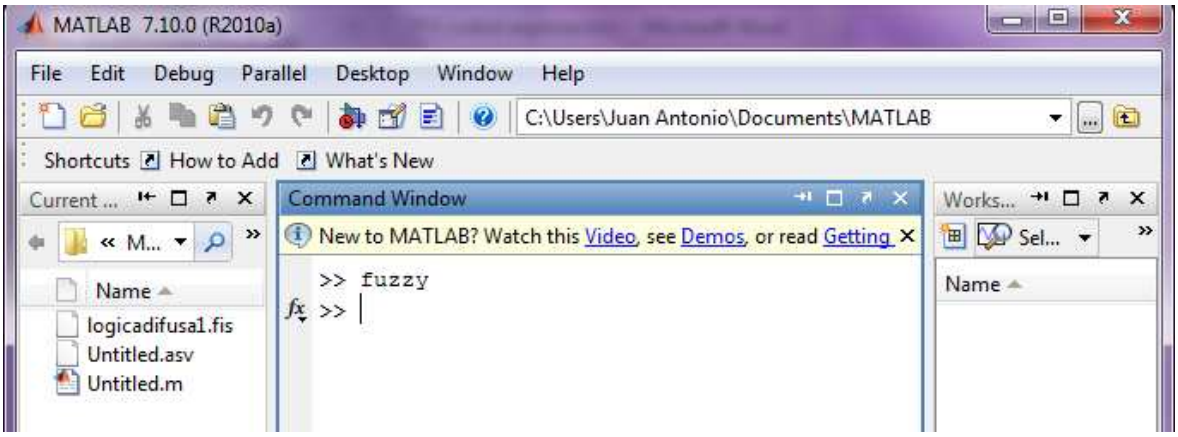



**Ilustración 56:** Workspace de Matlab con el comando fuzzy.

Y a continuación aparecerá la siguiente ventana **(Ilustración 57)**:

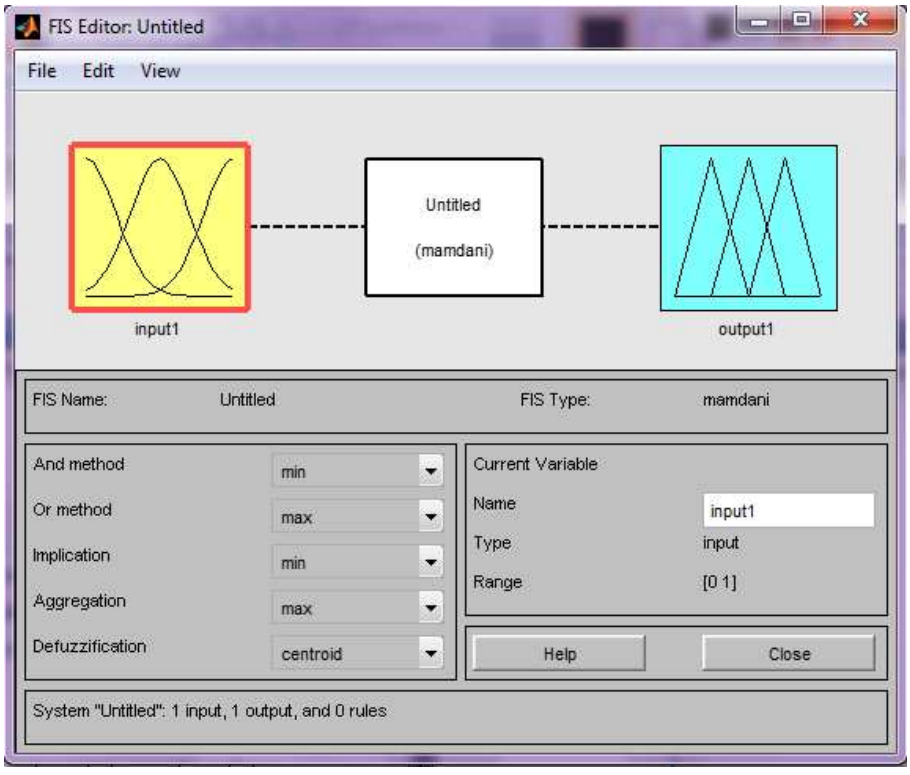



**Ilustración 57:** Toolbox Fuzzy de Matlab.

A través de este editor, se pueden modificar los métodos de los operadores lógicos ‘and’ y ‘or’, los métodos de implicación, de agregación y de defuzzificación. Permite elegir el modelo a usar Sugeno o Mamdani, a través de las distintas opciones de la barra de menús. Se pueden agregar variables de entrada y de salida, así como definir sus conjuntos borrosos, eligiendo rango, forma y parámetros.

Las reglas de control del método, se editan a través de un editor específico, en el que se van seleccionando los distintos valores de las variables de entrada y de salida.

Para poder implementar el controlador es necesario guardarlo, para poder importar el archivo al workspace, para que luego Matlab lo pueda reconocer y pueda ser usado por cualquier aplicación que lo necesite, o por Simulink. Si se desea trabajar con un modelo ya guardado se debe importar desde el menú *fuzzy* primero y luego exportarlo al workspace. En el apartado de robot explorador se explicará más detalladamente el procedimiento para nuestro caso.

# CAPÍTULO 4: ROBOT MÓVIL CON CONTROL DIFUSO POR VOZ.

# 4 ROBOT MÓVIL CON CONTROL DIFUSO POR VOZ

## 4.1 Introducción.

Como se explicó al principio de este documento, el objetivo de este proyecto, no es otro que la construcción de un robot móvil controlado por voz incorporando un motor de razonamiento difuso para el control de la aceleración y el frenado del vehículo.

Para la construcción física del robot, se ha optado por hacerlo con una base comprada y componentes Arduino al ser de bajo coste y tener buenas prestaciones. En el próximo punto se explica con más detalle los componentes y la constitución del robot y seguidamente en diferentes puntos se irá explicando el funcionamiento del robot y todo lo necesario para ello.

## 4.2 Componentes del robot móvil.

Tras un estudio del comportamiento y características que deseamos que el robot tenga y de la forma más económica, se ha llegado a una lista de componentes “definitiva”. A continuación se detallaran cada una de las partes que llevará el robot.

### 4.2.1 Arduino Uno

La placa Arduino que mejor se adapta al robot es la Arduino Uno *[12]*. En el apartado 3.7 se explicó que es y en qué consiste Arduino. A continuación se verá la placa Arduino y sus partes:

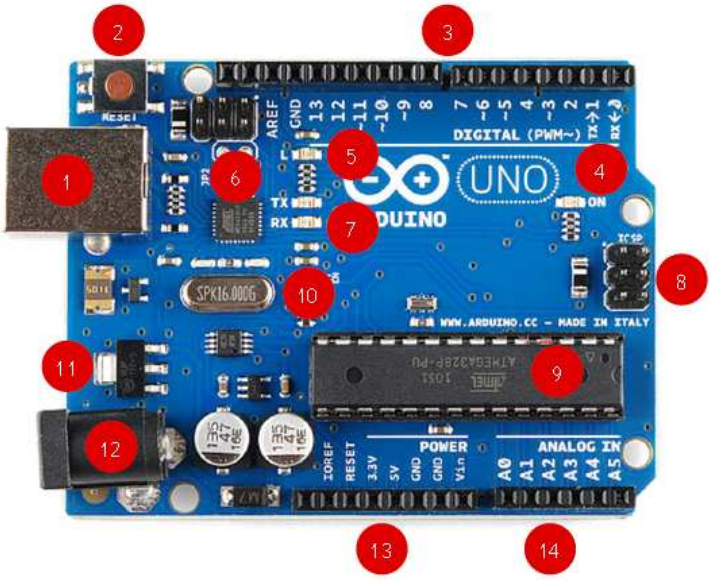



**Ilustración 58:** Placa Arduino Uno Detallada.

1) Conector USB para el cable Tipo AB
2) Pulsador Reset
3) Pines de E/S digitales y PWM
4) LED verde de placa encendida
5) LED naranja conectado al pin13
6) ATmega 16U2 encargado de la comunicación con el PC
7) LED TX (Transmisor) y RX (Receptor) de la comunicación serial
8) Puerto ICSP para programación serial
9) Microcontrolador ATmega 328, cerebro del Arduino
10) Cristal de cuarzo de 16MHz
11) Regulador de voltaje
12) Conector hembra 2.1 mm con centro positivo
13) Pines de voltaje y tierra
14) Entradas analógicas

La elección de esta placa se ha basado por su bajo coste y con la velocidad del microcontrolador que lleva integrado es suficiente para este proyecto. Las entradas y salidas disponibles en esta placa son más que suficientes, ya que no se utilizarán las 5 salidas PWM, solo se utilizará 2 para controlar los motores. Una entrada y una salida digital para el sensor ultrasónico y 4 salidas digitales para el control de la placa de motores.

Los Shields y sensores que llevará incorporado el robot son compatibles con la placa Arduino Uno.

### 4.2.2 Sensor ultrasónico HC-SR04

Se utilizará este sensor para realizar medidas de distancia a las que el robot se encuentra con los obstáculos, para su posterior control de velocidad según la distancia al obstáculo. Un ejemplo de funcionamiento sería si del robot al obstáculo hay 3 metros, el robot va a una velocidad alta, y cuanto menor vaya siendo esta distancia el robot irá disminuyendo su velocidad para que el frenado no sea brusco.

El HC-SR04 es un sensor de 4 pines, dos de alimentación Vcc y GND y dos para capturar la distancia Trig y Echo. El sensor se alimenta a 5V corriente continua, por lo que se puede alimentar directamente desde Arduino.

Para medir la distancia hay que generar un pulso en el pin Trig de un ancho o tiempo de 10 microsegundos como mínimo. Al mismo tiempo hay que monitorizar la señal que llega al pin Echo. La distancia calculada por el sensor se corresponde a:

Distancia = (Ancho de pulso·Velocidad Sonido)/2

Solo se tendrá que suministrar un impulso 10us corto a la entrada de disparo (Trig) para iniciar el alcance, y luego el módulo enviará una ráfaga de 8 ciclos de ultrasonidos a 40 kHz y elevar su eco. El eco es un objeto de distancia que es de ancho de pulso y la gama en proporción. Se puede calcular el rango a través del intervalo de tiempo entre que termina la señal de disparo y recibir la señal de eco: $Cm = \frac{uS\ (tiempo)}{58}$

### 4.2.3 Motor CC Reductor, chasis y ruedas.

Se ha elegido un paquete de chasis ya construido **(ilustración 59)**, lo único que hay que hacer es montarlo. Al chasis se le harán modificaciones en función de los componentes que se le vaya a montar.

El motor que trae el paquete funciona con valores de tensión entre 3V y 12V corriente continua. La corriente de carga es de 70mA con 300mA de máxima.

La primera configuración del chasis elegido es diferencial pero se ha tenido que realizar una segunda modificación sustituyendo la rueda libre trasera por dos ruedas fijas debido a un problema que se justificará en el apartado 4.4. Se han elegido estas configuraciones al ser las más económicas ya que solo se necesita dos motores.

**Ilustración 59:** Kit base robot.

### 4.2.4 Controlador motor L298

Se ha elegido esta placa controladora para los motores **(ilustración 60)**, porque la placa Arduino no puede gestionar directamente motores de corriente continua. Este controlador será gestionado por Arduino. Los dos motores a plena carga consumen 2x300mA = 600 mA. aproximadamente.

Un buen candidato tanto por sus dimensiones como sus prestaciones, es el *controlador DIY motor drive junta módulo inteligente trolley* basado en el conocido driver L298. Con sus dos salidas, tiene potencia suficiente (30W), para alimentar los dos motores. Cada motor irá conectado a una salida.

Este controlador tiene seis pines de salida (OUT1, OUT2, OUT3, OUT4 y tres salidas de 5V), donde OUT1 y OUT2 se conecta un motor y OUT3, OUT4 el otro motor, se deberá hacer la correcta conexión para que cuando se le ordene a la placa que los dos motores giren hacia delante, lo hagan los dos en esa dirección, si no es así se deberá invertir la conexión del motor que gire en sentido contrario. Dos salidas de 5V son para suministrar 5 voltios a las entradas analógicas del controlador, con esto se consigue una velocidad de giro del motor siempre constante y la otra salida de 5V es para alimentaciones auxiliares a 5 voltios. También tiene ocho pines de entrada (Vcc, GND, IN1, IN2, IN3, IN4, ENA,

ENB), en las entradas Vcc y GND son los pines donde se alimenta el controlador, con unos límites entre 3-15V, en este proyecto se alimentará con 12 Voltios. Las entradas IN1, IN2, IN3, IN4 son las que controlan la dirección de giro de los motores, donde IN1, IN2 son para un motor y IN3, IN4 son para otro motor. Las entradas ENA y ENB son enteradas analógicas que sirven para controlar la velocidad de giro en función de la tensión con las que son alimentadas.

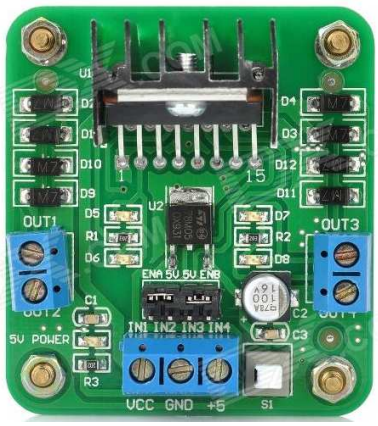



**Ilustración 60:** Controlador L298-H visto desde arriba.

### 4.2.5 Módulo inalámbrico APC220

Se utilizará el módulo APC220 para conectar el ordenador vía serial con el Arduino, con el objetivo de controlar el robot remotamente. Se han elegido estos módulos por su larga distancia de conectividad, ya que puede llegar a una distancia de transmisión de 1200m sin obstáculos. Se ha descartado los módulos XBee por tener un precio más elevado, y con el APC220 se pueden conseguir nuestros objetivos. Los módulos Bluetooth y módulos RF se han descartado por su corto alcance de transmisión.

Lo que hay que saber sobre la familia APPCON es su funcionamiento y sus características que son:

- **Frecuencia:** indica el rango de frecuencia en los cuales trabajan los módulos. El rango de frecuencia es indicado por el tipo de módulo elegido como –xx. Las bandas de frecuencia utilizadas por estos módulos son las siguientes:
  - 418Mhz a 455Mhz (Ej.: APC220-43)
  - 470Mhz a 510Mhz (Ej.: APC220-47)

Dentro de estas bandas de frecuencia, los módulos pueden ser programados en más de 100 canales.

- **Sensibilidad del receptor:** indica qué cantidad de señal (dBm) debe estar presente en un receptor inalámbrico para trabajar correctamente a una determinada velocidad de transmisión (bps). Cuanto menor sea este valor más sensible será y podrá recibir señales de potencias menores. La sensibilidad de los módulos APPCON varían desde -110dbm@9600bps a -117dBm@9600bps.
- **Potencia de salida:** es la potencia en mW que el módulo es capaz de entregar a una carga fantasma de 50 ohms.
- **Distancia:** es el alcance que se obtiene entre dos módulos del mismo modelo a una velocidad de transmisión determinada y con características de visión directa y adaptación de antena. Varían según los modelos desde 300 a 3000 m.
- **Max. Aire rate:** indica la máxima velocidad de transmisión de datos que puede alcanzar el módulo a través del aire entre un módulo y otro. Se expresa en bps.
- **Max. Baude rate:** indica la máxima velocidad de comunicación a través de la interface con el microcontrolador o dispositivo de comunicación de datos y el módulo
- **Interfaz:** es la forma de comunicación que puede tener el módulo con un dispositivo externo. Normalmente los módulos son incluidos dentro de sistemas que contienen microcontroladores o microprocesadores. Por lo tanto su entrada y salida de datos debe ser adaptable a circuitos de este tipo. Las diferentes interfaces que ofrece esta familia son UART, RS232 y RS485.

**Cuadro comparativo de características**

| Part Number | Tipo | Frecuencia (Mhz) | Sensibilidad de Recepción (dbm@9600) | Output Power (mW) | Distancia (mts@2400bps) | Interface | Dimensión | Max Air rate (bps) | Max Series rate (bps) |
|---|---|---|---|---|---|---|---|---|---|
| APC200A-43 | Transceptor | 431-470 | -113 | 20 | 800-1000 | RS232, RS485, UART | 39×19×7.0 | 9600 | 19200 |
| APC200A-91 | Transceptor | 862-956 | -113 | 20 | 600 - 800 | RS232, RS485, UART | 39×19×7.1 | 9600 | 19200 |
| APC220 | Transceptor | 418-455 o 470 - 510 | -114 | 20 | 800 - 1000 | UART | 37.5×18.3×7.0 | 19200 | 57600 |
| APC230 | Transceptor | 418-455 o 470 - 510 | -117 | 100 | 1500-1800 | UART | 39.5×18.3×7.0 | 19200 | 57600 |
| APC240 | Transceptor | 418-455 | -110 | 10 | 300-400 | UART | 35×18.3×7.0 | 25000 | 57600 |
| APC250 | Transceptor | 410 - 440 860-875 905-925 | -115 | 10 | 300-400 | UART | 32.1x18.3x7 | 19200 | 57600 |
| APC802 | Transceptor | 418-455 o 470 - 510 | -117 | 500 | 2700-3000 | RS232 (or RS485), UART | 50x31.9x7 | 19200 | 57600 |
| APC220N | Networks module | 418-455 o 470 - 510 | -116 | 20 | 600-800 | UART | 37.5×18.3×7.0 | 14400 | 115200 |
| APC230N | Networks module | 470-510 | -112 | 50 | 1000-1300 | UART | 39.5×18.3×7.0 | 14400 | 115200 |
| APC910M | concentrator | 470-510 | -116 | 50 | 1000-1300 | UART | 57.2×31.2×7.0 | 14400 | 115200 |

**Ilustración 61:** Tabla parámetros de los diferentes tipos de módulos *[6]*.

Como ya hemos dicho anteriormente el módulo elegido es el APC220-43, que es un transceptor half-duplex de alto nivel de integración. Cuenta con un MCU de altísima velocidad y un CI con grande capacidades en sus características de RF.

Cuenta con un avanzado sistema de corrección de errores gracias a la codificación por interpolación, reduciendo de esta manera la tasa de error. Es por este motivo por lo que se recomienda en industrias y zonas de gran interferencia.

El APC220-43 guarda una excelente relación entre el coste y beneficio, y junto con su tamaño lo hacen ideal para sistemas donde hace falta transferencia de datos de forma inalámbrica.

Posee una zona de buffer de datos de 256 bytes para transferencias, pero no solo es un transceptor completamente transparente para el usuario, sino que a esto se le suma la capacidad de discriminar hasta 100 canales diferentes lo cual lo hace altamente versátil.

**Características:**

- Hasta 800 mts de alcance (2400 bps)
- Potencia de salida de 20mW
- Rango de frecuencia entre 418Mhz y 455Mhz
- Más de 100 canales
- Modulación en GFSK
- Interfaz UART/TTL
- Buffer de datos de 256 bytes
- Apto para grandes transferencias de datos
- Software RF Magic V4.2 para seteo de parámetros

**Aplicaciones**

- Lectura automática de medidores
- Sensores inalámbricos
- Automatización industrial
- Control de semáforos
- Terminales inalámbricos de mano
- Control y monitoreo remoto
- Reemplazo de cables
- Detectores de gas y combustible
- Control de Robots

**Datos mecánicos:**

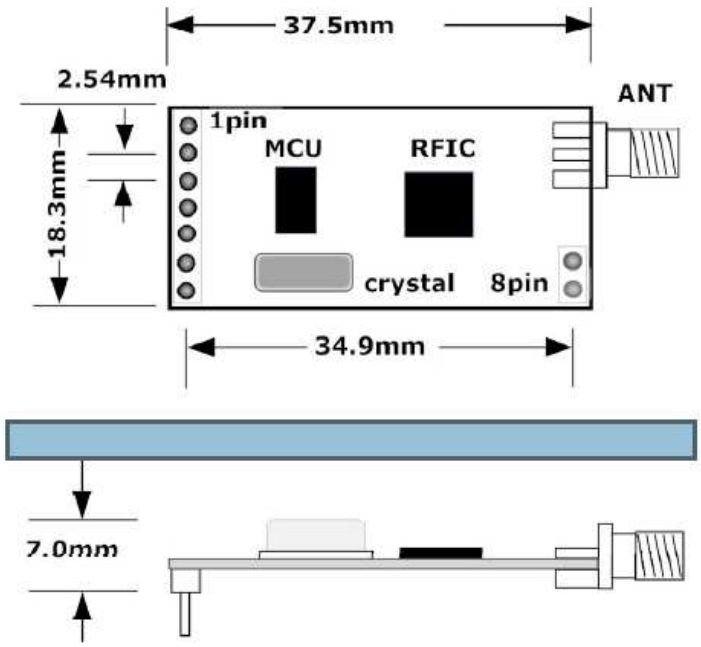



**Ilustración 62:** Dimensiones del módulo APC220 *[5]*.

**La descripción de los pines es la siguiente (ilustración 63):**

| PIN | NOMBRE | FUNCION | DESCRIPCION |
|---|---|---|---|
| 1 | GND | POWER | TIERRA (0v) |
| 2 | VCC | POWER | ALIMENTACION DC 3.5 – 5.5 V |
| 3 | EN | ENTRADA | HABILITADO = 1 LÓGICO<br>SLEEP = 0 LÓGICO |
| 4 | RXD | ENTRADA | ENTRADA UART TTL |
| 5 | TXD | SALIDA | SALIDA UART TTL |
| 6 | AUX | ENTRADA | NO SE UTILIZA |
| 7 | SET | ENTRADA | MODO SETEO = 0 LÓGICO<br>MODO NORMAL = 1 LÓGICO |

**Ilustración 63:** Tabla de descripción de pines del módulo APC220 *[5]*.

**Especificaciones Técnicas (ilustración 64):**

| APC220-43 | |
|---|---|
| Frecuencia de trabajo | 418MHz to 455MHz |
| Modulacion | GFSK |
| Intervalos de frecuencia | 200KHz |
| Potencia de salida | 20mw (10 niveles) |
| Sensibilidad de recepcion | -118dBm@1200bps |
| Baudios en aire | 2400 - 19200bps |
| Baudios serie | 1200 - 57600bps |
| Puertos COM soportados | 8E1/8N1/8O1 |
| Buffer de datos | 256bytes |
| Humedad | 10%~90% |
| Temperatura | -20℃ - 70℃ |
| Tension de alimentacion | 3.3 – 5.5V (ripple ±50mV ) |
| Corriente de transmission | ≦35mA@20mW |
| Corriente de recepcion | ≦25mA |
| Corriente en SLEEP | ≦5uA |
| Distancia de transmision | 800mts |
| Dimensiones | 37.5mm x 18.3mm x 7.0mm |
| | |

**Ilustración 64:** Tabla de especificaciones técnicas del módulo APC220 *[5]*.

**Tabla de parámetros del APC220 (ilustración 65):**

| Parámetro | Opciones | De fábrica |
|---|---|---|
| Velocidad serie | 2400, 4800, 9600, 19200, 38400, 57200 | 9600 |
| Paridad | Par, Impar, Sin paridad | Sin paridad |
| Frecuencia | 418 – 455 Mhz | 434 Mhz |
| Velocidad RF | 2400, 4800, 9600, 19200 | 9600 |
| Potencia RF | 1 a 9 | 9 (20mW) |

**Ilustración 65:** Tabla de parámetros del módulo APC220 *[5]*.

Todos los módulos APPCON tienen una serie de parámetros que el usuario debe configurar para que el mismo responda según las condiciones que se requiera. Esos parámetros son los siguientes:

- **Canal de frecuencia:** dentro de la banda de frecuencia del módulo elegido, el usuario puede elegir en que frecuencia exacta quiere que el módulo trabaje. Esto permite que en un mismo ámbito de trabajo, puedan coexistir diferentes grupos de módulos sin que los mismos se interfieran entre sí.
- **Baute Rate Serie:** con esto le vamos a indicar a nuestro módulo con que tasa de transferencia nos vamos a comunicar con él. Esta tasa de transferencia debe ser igual a la que seleccionemos en nuestro circuito electrónico.
- **Paridad serie:** Aquí seleccionamos el chequeo de paridad que queremos realizar, podemos elegir entre paridad par, paridad impar o sin paridad.
- **Potencia de salida:** la característica indicada como potencia de salida de la tabla de arriba, nos indica la máxima potencia que el módulo puede generar. Pero el usuario tiene hasta 9 niveles de potencia seleccionable.
- **RF Rate:** con este parámetro indicamos a qué velocidad queremos que se transmitan los datos y a qué velocidad se prepara para recibir.

## 4.3 Instalación y configuración de Módulo APC220

Existen dos formas de cambiar la configuración de los parámetros:

1. A través del software de PC y la placa USB que permite su interconexión.
2. On-line, esto es, a través del microcontrolador en la propia aplicación del circuito.

**Mediante el software del PC**

Hay tres programas diferentes para la programación de los módulos dependiendo de qué módulo se quiera programar **(Ilustración 66)**.

| Módulos | Software |
|---|---|
| APC200 | RF-MAGIC V4.2 |
| APC220<br>APC230<br>APC802 | RF-MAGIC V1.2A |
| APC240<br>APC250 | RF-MAGIC V1.4 |

**Ilustración 66:** Tabla de software necesario para la instalación según el módulo que se tenga *[6]*.

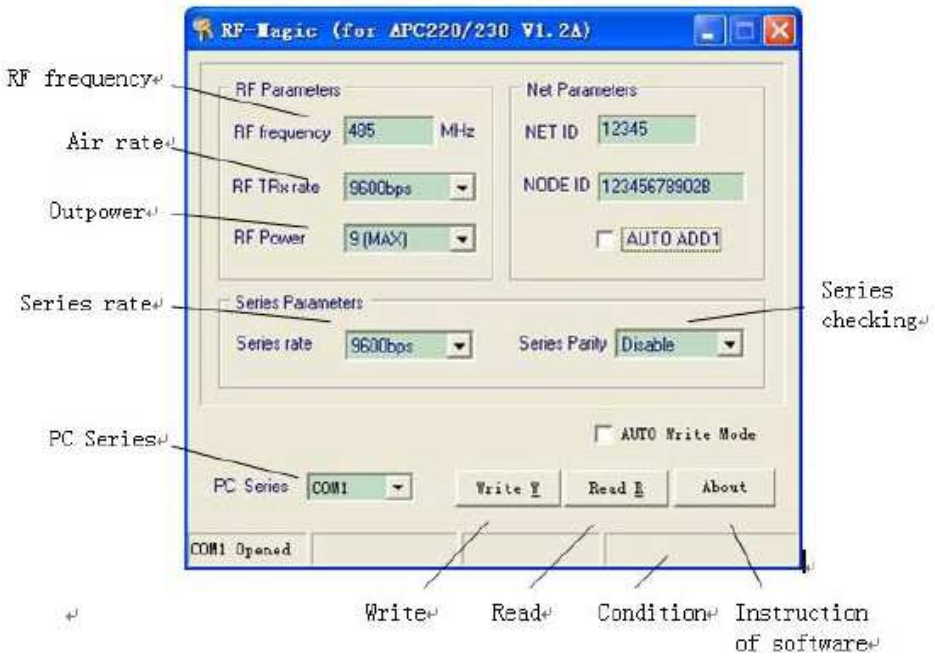



**Ilustración 67:** Programa Magic detallando los campos de los parámetros.

La configuración de los parámetros se realiza a través de la UART del módulo. Los niveles de tensión deben ser TTL por lo tanto se necesita de un adaptador entre el puerto COM (o USB) de la PC y el módulo.

Una vez conectado el conversor USB (o COM)/UART TTL vamos a ver en la parte de abajo del software al lado del "COM Opened" la leyenda "Found Device". A partir de este momento podemos comenzar a leer y escribir los parámetros que consideremos necesarios para nuestra aplicación.

**Configuración mediante ON-LINE: (A TRAVÉS DE UN MCU)**

Esta configuración es válida solo para los módulos APC220, 230, 802. Para los otros módulos se sigue otro procedimiento que no se explicará ya que nuestro módulo elegido es el 220.

Para la programación on-line de los parámetros del módulo, debemos asegurarnos de una correcta conexión eléctrica entre el microcontrolador y el módulo. Se deberá conexionar así **(ilustración 68)**:

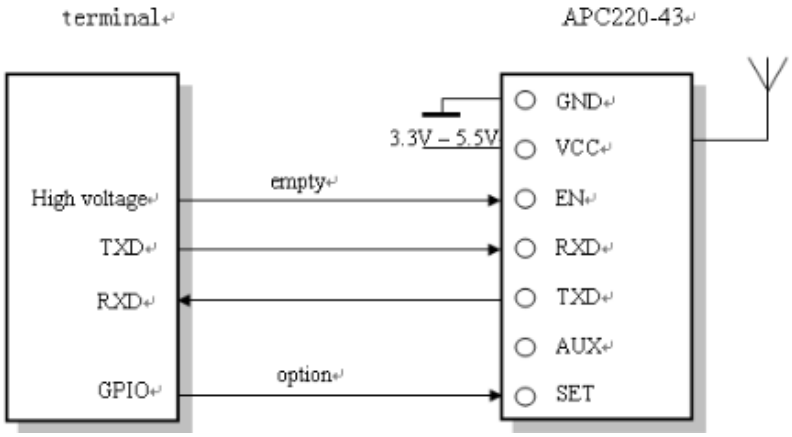



**Ilustración 68:** Esquema de conexión APC220 y microcontrolador para la configuración *[5]*.

Los pines TXD y RXD son los de la UART del microcontrolador los cuales se conectan con el módulo para transferir los datos y para la programación.

Según queramos transmitir o configurar debemos colocar sobre el pin SET el nivel de tensión correspondiente **(ilustración 69)**:

| NIVEL LOGICO DEL PIN SET | FUNCION |
|---|---|
| '1' | Estado de funcionamiento normal (RUNNING) |
| '0' | Estado de configuración (SETTING) |

**Ilustración 69:** Tabla de estados del pin SET *[6]*.

En el diagrama de tiempo de la siguiente imagen se puede ver con mayor detalle cuales son las condiciones que debemos cumplir para que el módulo pueda ser programado on-line **(ilustración 70)**:

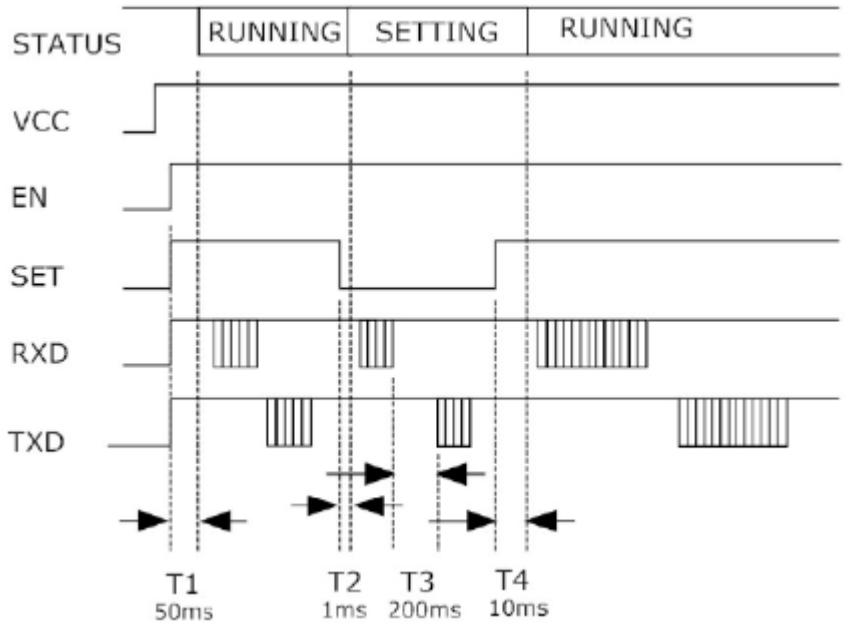



**Ilustración 70:** Diagrama de tiempo de configuración de parámetros *[6]*.

En este diagrama (ilustración 70) se puede observar que para enviar comandos de programación debemos poner a '0' el pin de SET y esperar un tiempo T2 mayor a 1ms para comenzar a enviar los comandos de configuración. Cabe aclarar que en el diagrama el pin descrito como RXD es el del módulo, por lo tanto es el PIN TXD del microcontrolador por el cual se transmiten los comandos. El pin TXD del diagrama es el pin RXD del microcontrolador sobre el cual se reciben las respuestas a los comandos emitidos por el pin TXD.

El seteo de los parámetros debe realizarse a 9600bps sin paridad. La configuración se realiza a través del código ASCII.

**PROTOCOLO PARA CONFIGURACION DE PARÁMETROS**

| Comando | (32) | Para 1 | (32) | Para 2 | (32) | Para 3 | (32) | Para 4 | (32) | Para 5 | \r | \n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Ilustración 71:** Protocolo que hay que seguir para la configuración del módulo *[6]*.

**Comando:** son 2 bytes e indica si vamos a escribir o leer del módulo.

1. ASCII: WR DECIMAL: (87; 82) HEXA: (0x57; 0x52) indica que se van a escribir parámetros en el módulo.
2. ASCII: RD DECIMAL: (82; 68) HEXA: (0x52; 0x44) indica que se van a leer los parámetros del modulo.

**(32):** es un byte. Es el valor decimal fijo que indica ESPACIO y sirve para separar los parámetros. En Hexadecimal es 0x20.

**Para x:** son los distintos valores que le asignamos a cada parámetro. La cantidad de byte y la información que se envía cambia para cada parámetro según la siguiente tabla:

| Tabla de parámetros | | |
|---|---|---|
| Parámetro | bytes | Formato |
| Frequency **(para 1)** | 6 | La unidad es el Khz, por ejemplo 434MHz es 434000 |
| Air rate **(para 2)** | 1 | 1: 2400<br>2: 4800<br>3: 9600<br>4: 19200 |
| Output power **(para 3)** | 1 | 0 a 9, 0 expresa -1dBm, 9 expresa 13dBm(20mW) |
| Series data rate **(para 4)** | 1 | 0, 1, 2, 3, 4, 5, 6<br>expresa respectivamente<br>1200, 2400, 4800, 9600, 19200,38400,57600bps |
| Series checkout **(para 5)** | 1 | 0: sin paridad<br>1: paridad par<br>2: paridad impar |

**Ilustración 72:** Tabla de parámetros que se desea para el módulo *[6]*.

**Es muy importante destacar que los parámetros se escriben en código ASCII, o sea que un 0(cero) no es realmente el número cero, sino el valor ASCII que corresponde al cero, el cual es en decimal 48 y en hexadecimal 0x43.**

Por ejemplo vamos a setear a un APC220-43 con las siguientes características:

Frecuencia = 434Mhz

RF data rate = 9600bps

Output power = 20mW

Serie data rate = 1200bps

Paridad = sin paridad

El armado de la trama a enviar en código ASCII será:

**WR_434000_3_9_0_0**

En código hexadecimal la trama queda de la siguiente manera:

**0x57,0x52,0x20,0x34,0x33,0x34,0x30,0x30,0x30,0x20,0x33,0x20,x0x39,0x20,0x30,0x20, 0x30,0x0D,0x0A**

La respuesta del módulo en ASCII es la siguiente:

**PARA_434000_3_9_0_0**

La respuesta del módulo en Hexadecimal será:

**0x50,0x41,0x52,0x41,0x20,0x33,0x34,0x30,0x30,0x30,0x20,0x33,0x20,x0x39,0x20,0x30, 0x20,0x30,0x0D,0x0A**

**Para este proyecto se ha optado por la configuración mediante PC, por lo tanto a continuación de describirán los pasos que se han llevado a cabo.**

Primero descargar e instalar los controladores de convertidor USB-TTL **(ilustración 73)**.

**Download for Windows XP/Server 2003/Vista/7/8/8.1 (v6.6.1)**

| Platform | Software | Release Notes |
|---|---|---|
| Windows XP/Server 2003 Vista/7/8/8.1 | Download VCP (3.49 MB) | Download VCP Revision History |

**Ilustración 73:** Enlace de descarga programa Magic.

Después conectar el convertidor USB con el módulo de APC insertado. Cuando se conecta el módulo, Windows lo reconoce e instala los controladores necesarios **(ilustración 74 y 75)**.


**Ilustración 74:** Buscando controlador de dispositivo(módulo APC220).

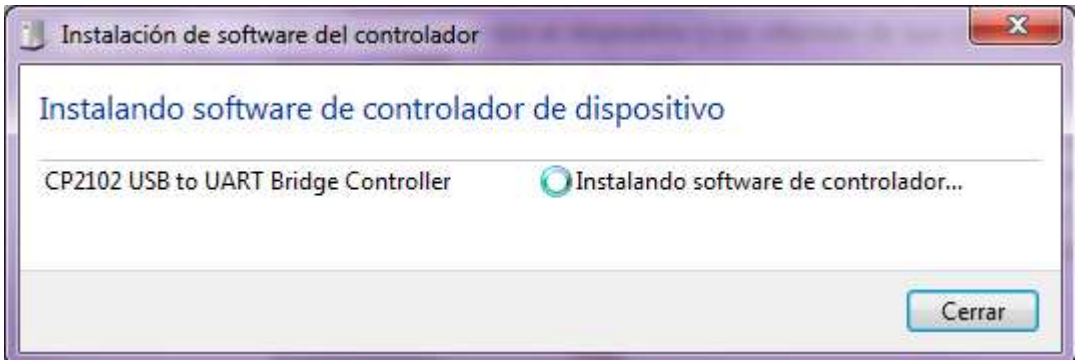



**Ilustración 75:** Instalando controlador de dispositivo (módulo APC220).

Una vez que los controladores están instalados correctamente y el hardware está listo para usar, iniciar el programa de interfaz de RF-ANET **(ilustración 76)**, ejecutándolo como administrador, ya que si no es así no reconoce ningún puerto COM visible. Hay que cambiar el puerto-com del módulo USB-TTL a un número de puerto más bajo, en este caso se ha optado por COM 7.

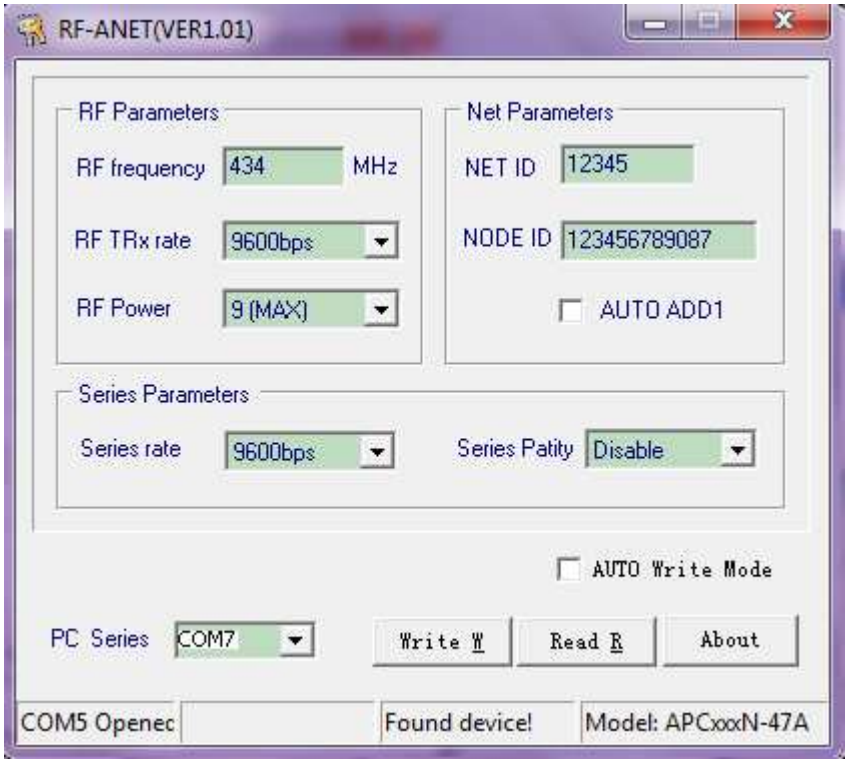



**Ilustración 76:** Programa Magic con parámetros del módulo APC220.

Para cambiar el número de puerto serie, abrir el Administrador de dispositivos **(ilustración 77)** de Windows. Hacer clic en Puertos (COM y LPT) y pinchar en Silicon Labs CP210x USB to UART Bridge COM7 **(ilustración 78)** y configurándolo a 9600Bps, 8 Bits de datos, Paridad de Ninguno, Bits de parada 1, control de flujo Ninguno, Com-Port elegir COM7.

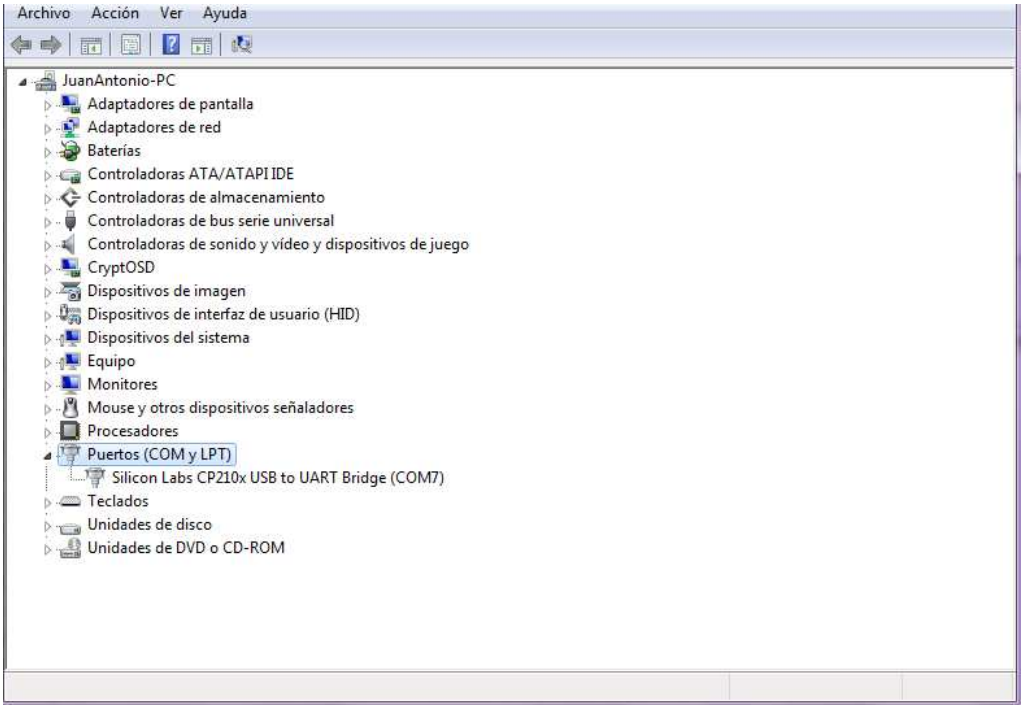



**Ilustración 77:** Administrador de dispositivos.

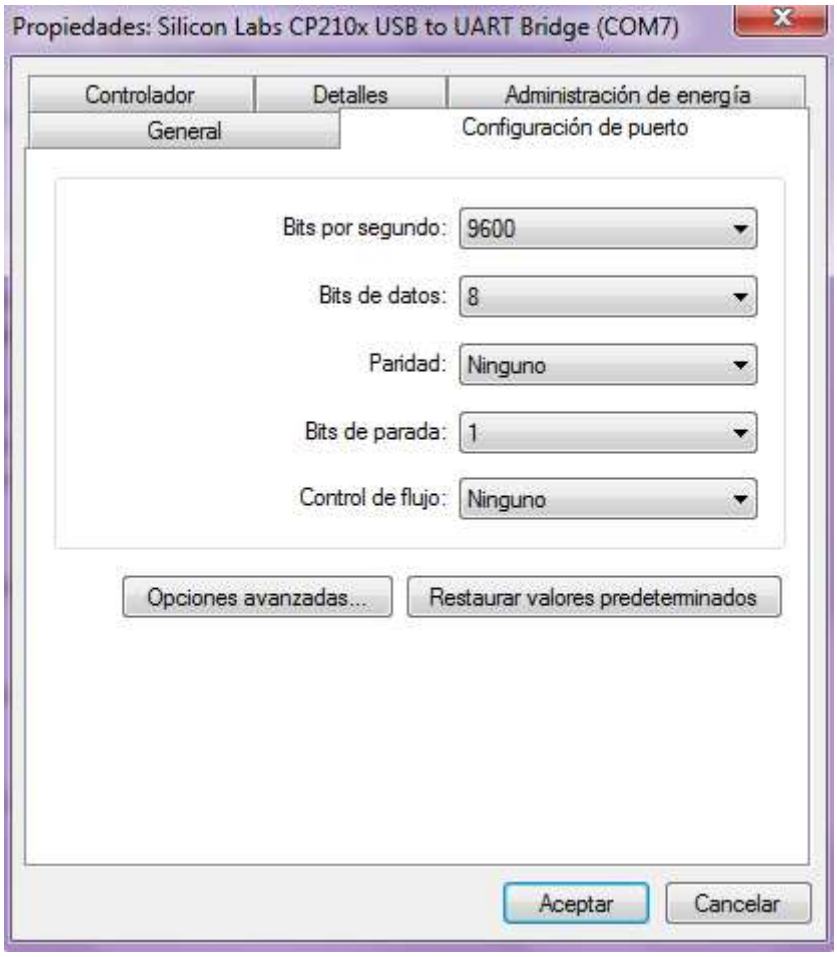



**Ilustración 78:** Administrador de dispositivo, detalles de Modulo APC220.

Iniciado RF-ANET (Magic), pulsar Read R, para que lea los datos del módulo, y luego pulsar Write W para guardar la configuración en el módulo.

Desconectar el módulo e insertar el otro módulo en el adaptador USB-TTL y volver a abrir el programa RF-ANET y darle Read R, y verificar que todos los datos son iguales que el anterior menos el ID de nodo, y después pulsar Write W para guardar los ajustes en el segundo módulo.

Para que la conexión inalámbrica funcione, primero hay que cargar mediante el cable USB el programa al Arduino, y después desconectarlo y cambiar el puerto com ajustado al modulo APC y abrir el puerto serie

## 4.4 Montaje del robot.

Para la construcción del robot se ha optado por comprar una base desmontada **(ilustración 79)**, que se procederá al montaje de dicha base y se instalaran los componentes del robot.

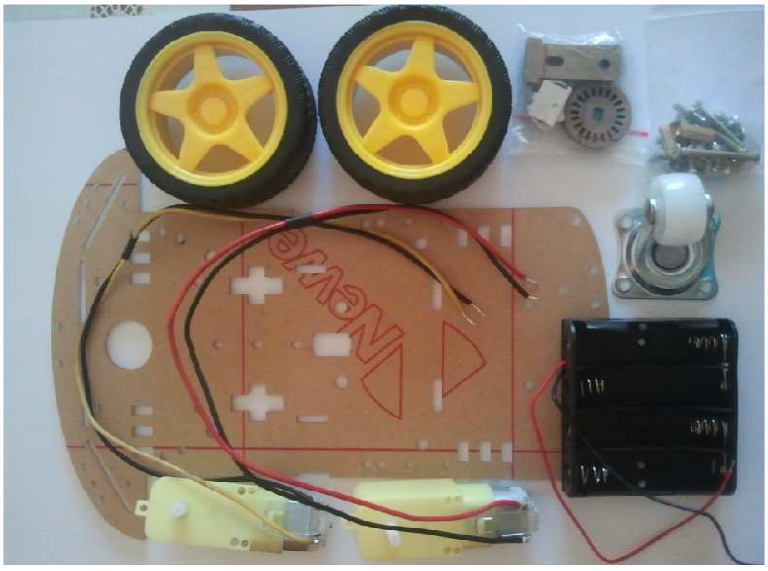


**Ilustración 79:** kit de base robot.

A continuación se irá viendo paso a paso el montaje:

Para soldar se necesita **(ilustración 80)** soldador eléctrico, estaño y líquido especial para soldar para que el estaño se extienda mejor por la superficie a soldar. También usaremos lija para lijar las partes que puedan tener esmalte o suciedad, con esto se conseguirá una mejor soldadura.

**Ilustración 80:** Soldador, estaño, lija y decapante.

Primero se ha soldado dos cables a cada uno de los motores de corriente continua **(ilustración 81)** ya que venían sin cables (negro para negativo y rojo o amarillo para positivo).

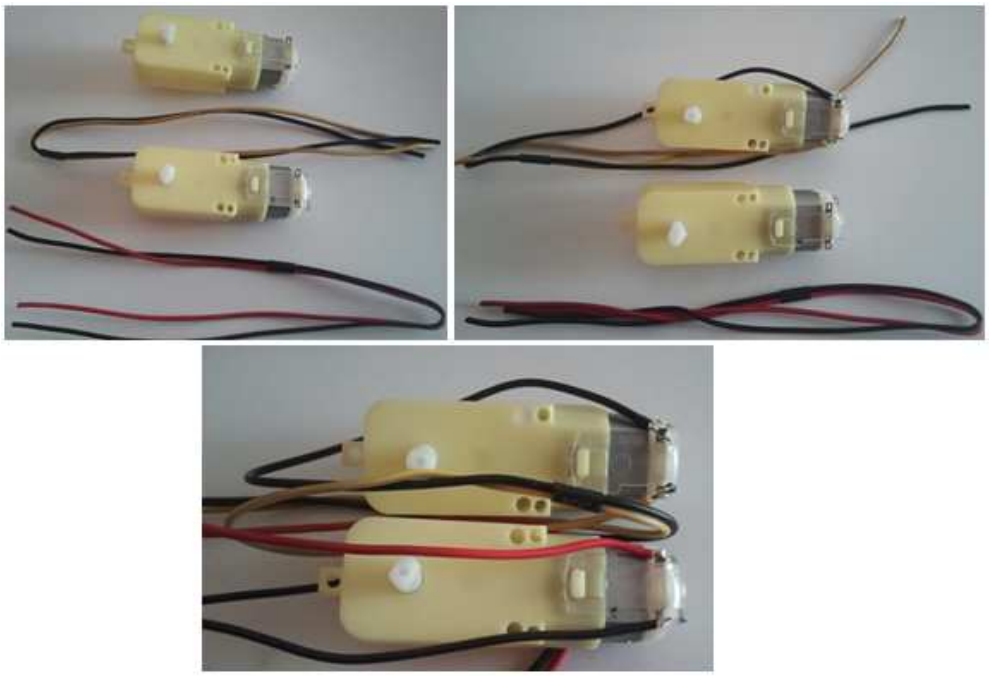


**Ilustración 81:** Soldado de cables a motores CC.

Segundo se procederá al montaje de los motores en la base **(ilustración 82)**, para ellos introduciendo las 4 “T” en las correspondientes ranuras, para despues sujetar los motores con tornillos adecuados. También se monta el interuptor en el hueco que viene definido para él. Se colocan los pequeños circulos en el eje del motor para que el eje esté equilibrado.

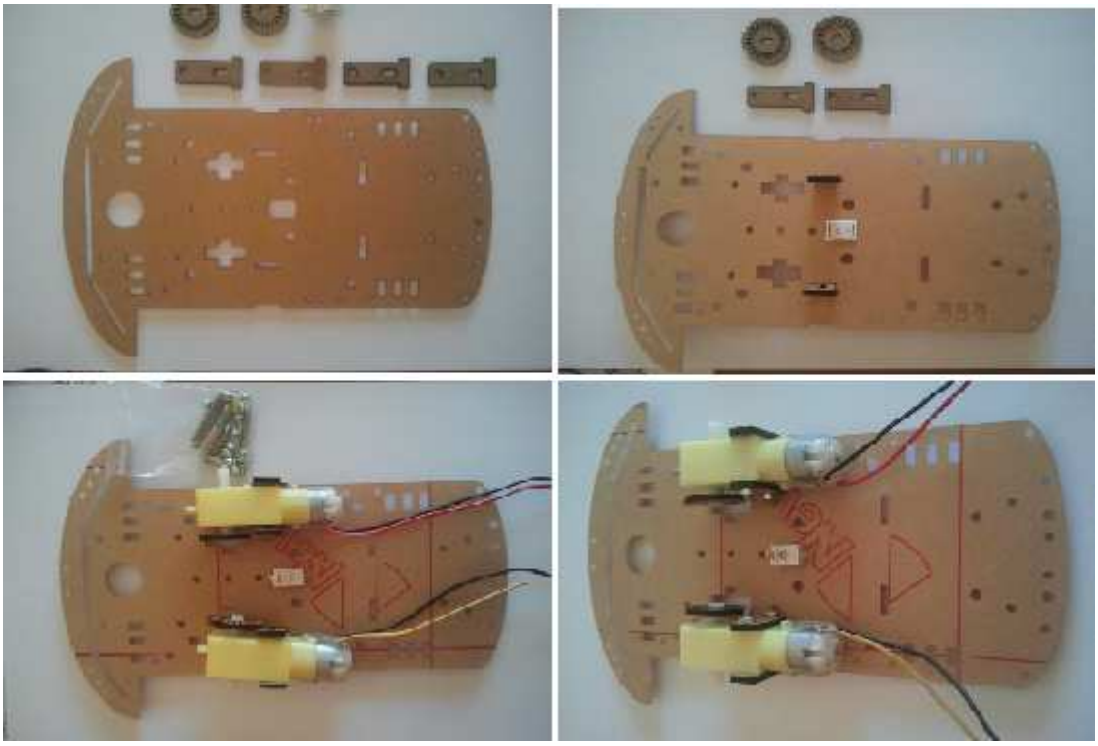


**Ilustración 82:** Montaje de motores en la base.

Una vez que tenemos los motores sujetos, se monta la rueda libre trasera **(ilustración 83)**, haciendo uso de tuercas largas para que se quede a la altura adecuada y la base esté a nivel, una vez que se coloquen las ruedas en los ejes de los motores. A los engranajes de la rueda trasera se le echó un poco de aceite para que gire mejor.

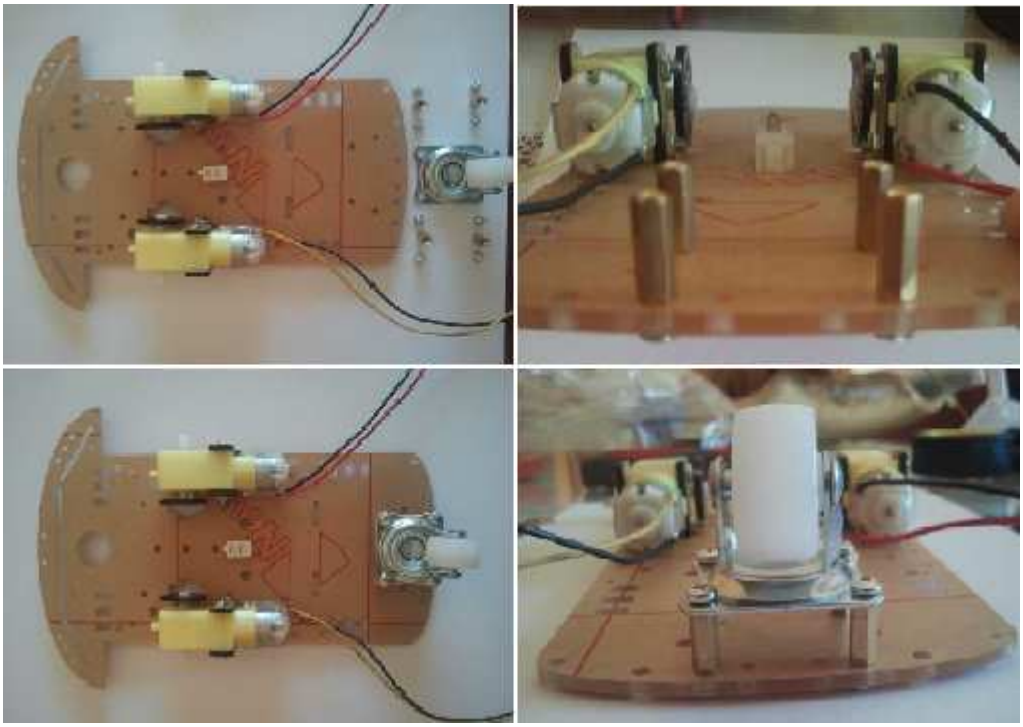


**Ilustración 83:** Montaje de rueda trasera.

Cuando está la rueda trasera montada se procede al montaje de las ruedas delanteras **(ilustración 84)**. Para montarlas basta con ponerlas sobre el eje exterior del motor y ejercer presión, no se necesita pegamento ni cualquier otro adiptivo, ya que hay que ejercer bastante presión para que entren en el eje, y no se saldrán solas.

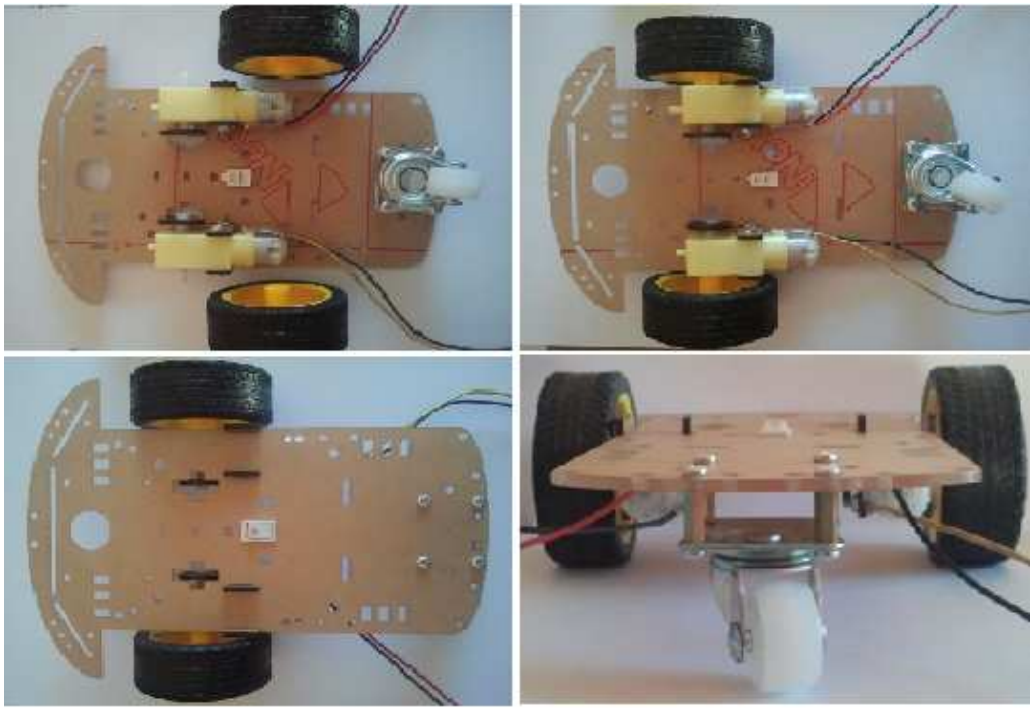


**Ilustración 84:** Montaje de ruedas delanteras.

Una vez llegado a este punto se retira el papel protector de la base y de los componentes **(ilustración 85)**, con la ayuda de un cúter.

**Ilustración 85:** retirada del papel protector de la base.

Una vez que esta la base totalmente montada, se intalarán los componentes electronicos **(ilustración 86)** que llevará el robot. Se ha hecho uso de componentes de aparatos viejos en desuso, en este caso de un televisor y algunos componentes de un ordenador. Principal mente lo que se a reutilizado de otros aparatos han sido conectores y cables.

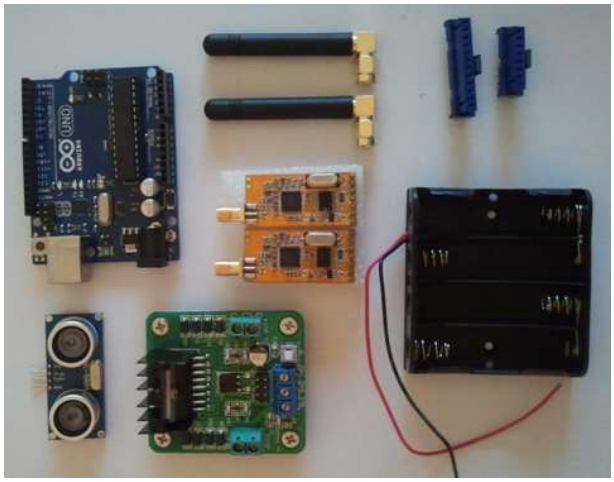


**Ilustración 86:** Componentes electrónicos del robot.

Los componentes de la ilustración 86 son la placa Arduino Uno, los módulos inalámbricos APC220, controlador de motores, sensor volumétrico, conectores (cogidos de un televisor viejo) y soporte para pilas. Este último en principio no se montará, pero se guardará por si hicera falta en algún momento.

Primero lo que se ha hecho es un reorganización de los componentes electrónicos sobre la base **(ilustración 87)**, colocándolo de diferentes maneras y en posiciones distintas, llegando a la siguiente configuración:

**Ilustración 87:** reorganizamiento de los componentes electrónicos.

El sensor volumétrico tiene que ir en la parte delantera, que será el encargado de tomar las medidas a los objetos que se encuentra, por lo tanto tiene que ir en la parte delantera. Los módulos APC220 se ha optado por montarlos en la parte trasera para que la antena vaya libre de obstáculos y tenga una mejor recepción. La placa Arduino Uno se ha puesto en medio para su fácil conexión con el PC, al haber más hueco disponible. El controlador de motores se ha montado en la parte trasera porque la parte delantera deberá ir la cámara de video. El interruptor de encendido se ha cambiado de lado, anteriormente estaba en el centro de la base como se vió en las imágenes anteriores y ahora se ha puesto en un hueco que quedaba entre el motor de la rueda izquierda y el controlador de motores.

Una vez decidido la organización adecuada, se pasa a la modificación de la base **(ilustración 88)**, ya que serán necesarios nuevos agujeros y ranuras para montar los conectores y el interruptor. Se marca con un rotulador rojo lo que hay que modificar y se lleva a cabo la modificación.

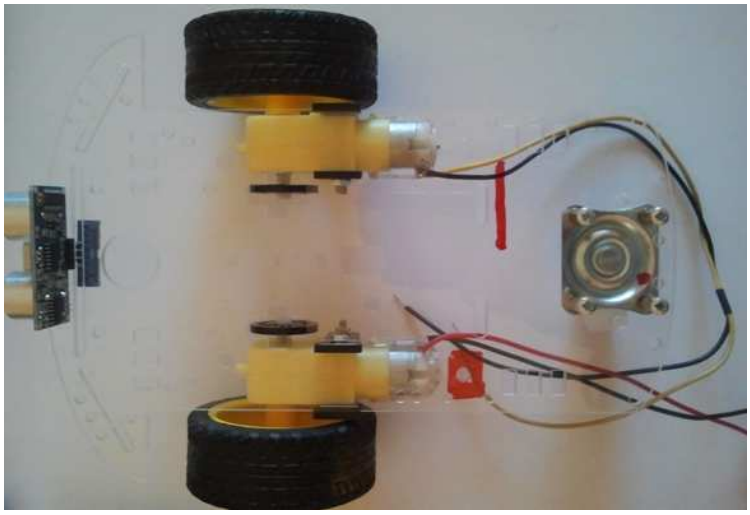


**Ilustración 88:** Modificación de la base para instalar APC220 e interruptor.

Como se puede observar hay varias marcas rojas, eso son las modificaciones que hay que hacer. En la parte donde está colocado el sensor volumétrico también se hizo una pequeña modificación para que entrara el conector, no está marcado en dicha imagen porque se olvido tomar la foto.

Para la modificacion de la base se a utilizado una máquina llamada mini taladro **(ilustración 89)**, que tiene varias "fresas" para limar superficies. La "fresa" utilizada para esta modificación es la que tiene puesta en la imagen.


**Ilustración 89:** Mini taladro.

Una vez realizada la modificación se lleva a cabo el montaje de los componentes que van sobre dichas modificaciones **(ilustración 90)**.

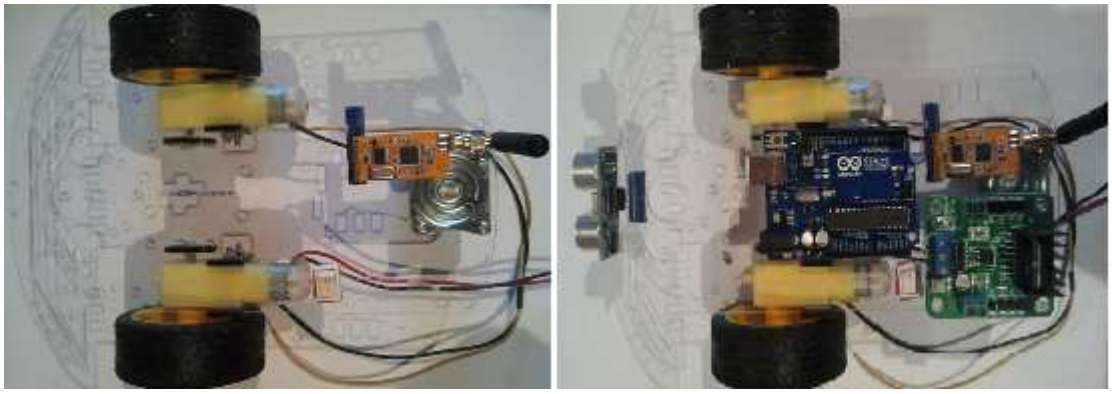


**Ilustración 90:** Montaje de componentes electrónicos.

Una vez llegado a este punto, se empiezan a soldar los cables a los pines de los conectores **(ilustración 91)**. Se mostrarán las conexiones del conector grande ya que el conector de menor tamaño se hace exactamente igual. Los cables que se le van a soldar son de una manguera en desuso, así se podrán aprovechar.

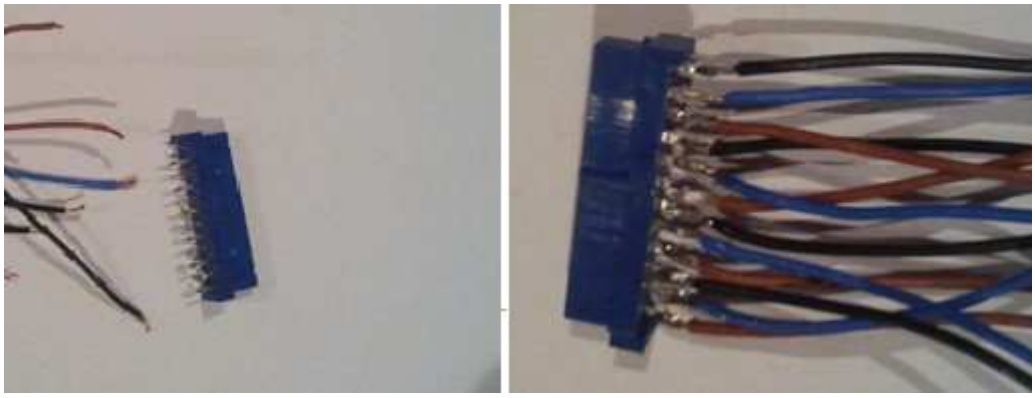


**Ilustración 91:** Soldado de cables a los conectores.

Una vez soldados los cables a los conectores, se montan en la base **(ilustración 92)** y se pegan con algún pegamento adecuado para estos materiales. En este caso se realizó con pegamento de secado rápido.

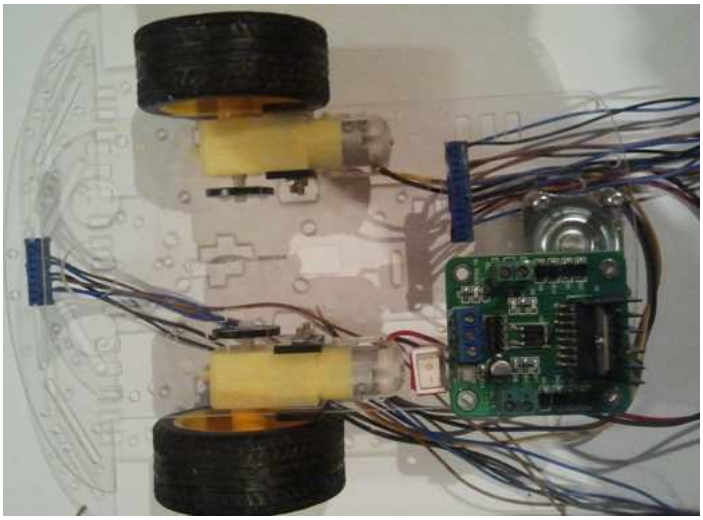


**Ilustración 92:** Instalación de conectores a la base.

Una vez puestos los conectores en su lugar, se montan el resto y los dispositivos y se fijan en la base para que no se muevan **(ilustración 93)**. El módulo APC220 va en uno de los conectores, solo hace falta introducir adecuadamente los pines del módulo en el conector, igualmente se hace con el sensor volumétrico.

**Ilustración 93:** Montaje total de los componentes sobre la base.

Como se puede observar en la ilustración 92 los conectores tienen mas pines de los necesarios, se hace así por si mas adelante pudieran hacer falta para cualquier conexión improvista. Si no hicieran falta se dejarían en desuso.

Ahora lo siguiente es ir conectando eléctricamente los componentes electrónicos. En primer lugar se han hecho los conectores para conectar el controlador de motores a la placa Arduino Uno. Los conectores se han cogido de componentes de informática que no valian y lo modificamos a nuestra necesidad **(ilustración 94)**. Una manguera elegida tiene en sus dos extremos un conector de tres pines, donde se le ha hecho un corte y se han separado,

eliminando uno de ellos que hacia de pantalla del cable, dejando solo dos pines a cada lado, luego se ha cortado por la mitad para aprovechar un trozo para futuras conexiones, ya que solo se necesita un estremo con dos pines para conectar las entradas ENA y ENB. Hay otro conector de cuatro pines que vendrá bien para conectar las entradas IN1, IN2, IN3, IN4 del controlador, y las puntas del extremo del cable se soldarán para hacer una buena conexión y se ajuste bien a la placa Arduino Uno.

**Ilustración 94:** Modificación de conectores para la adaptación a Arduino.

Una vez que tenemos los conectores hechos se pasa a la conexión entre el controlador de motores y la placa Arduino **(ilustración 97)**, pero primeramente se soldarán las puntas del extremo de los cables de los conectores **(ilustración 95)** para que encajen bien en la placa Arduino Uno.

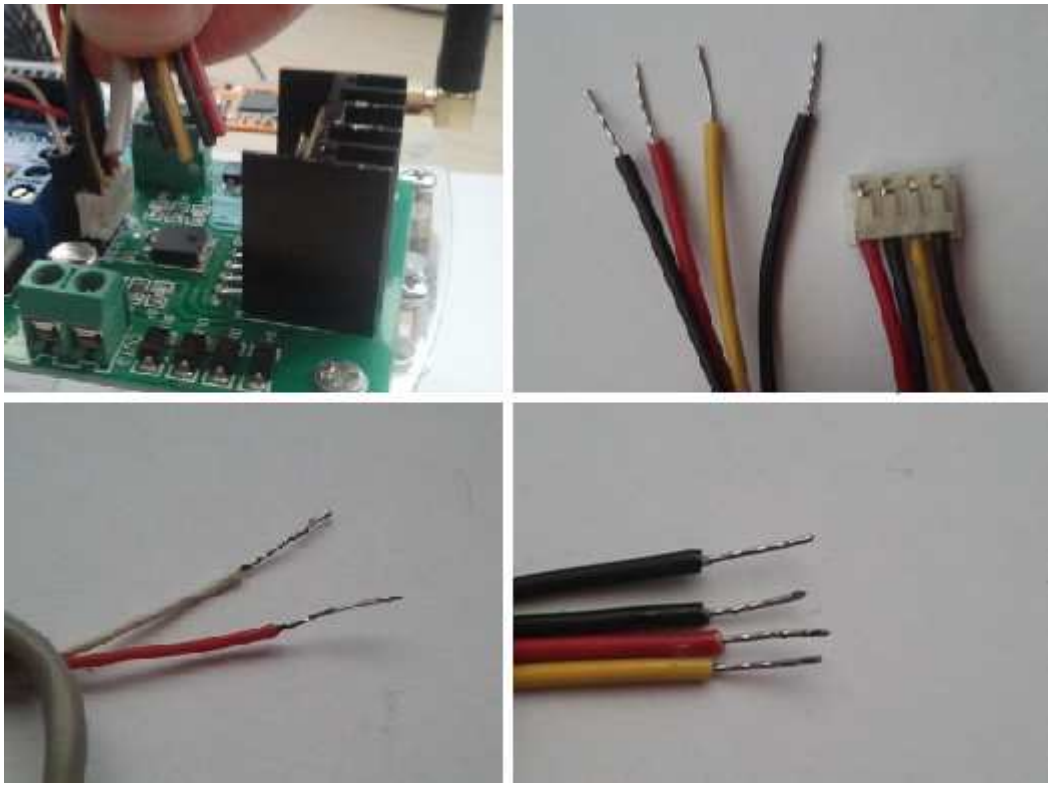


**Ilustración 95:** Estañado de los cables para Arduino.

Una vez soldado los extremos se conectará el controlador de motores y Arduino Uno según el siguiente esquema **(ilustración 96)**:

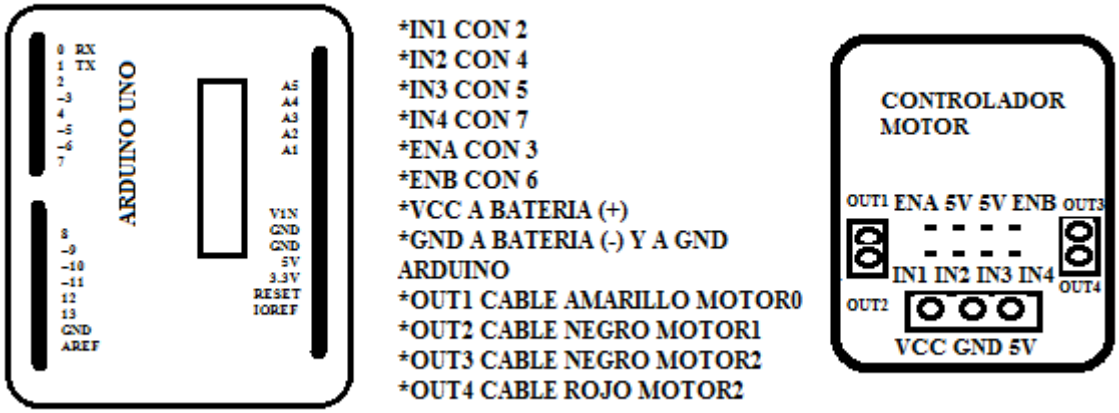



**Ilustración 96:** Esquema de conexión Arduino al resto de los componentes.

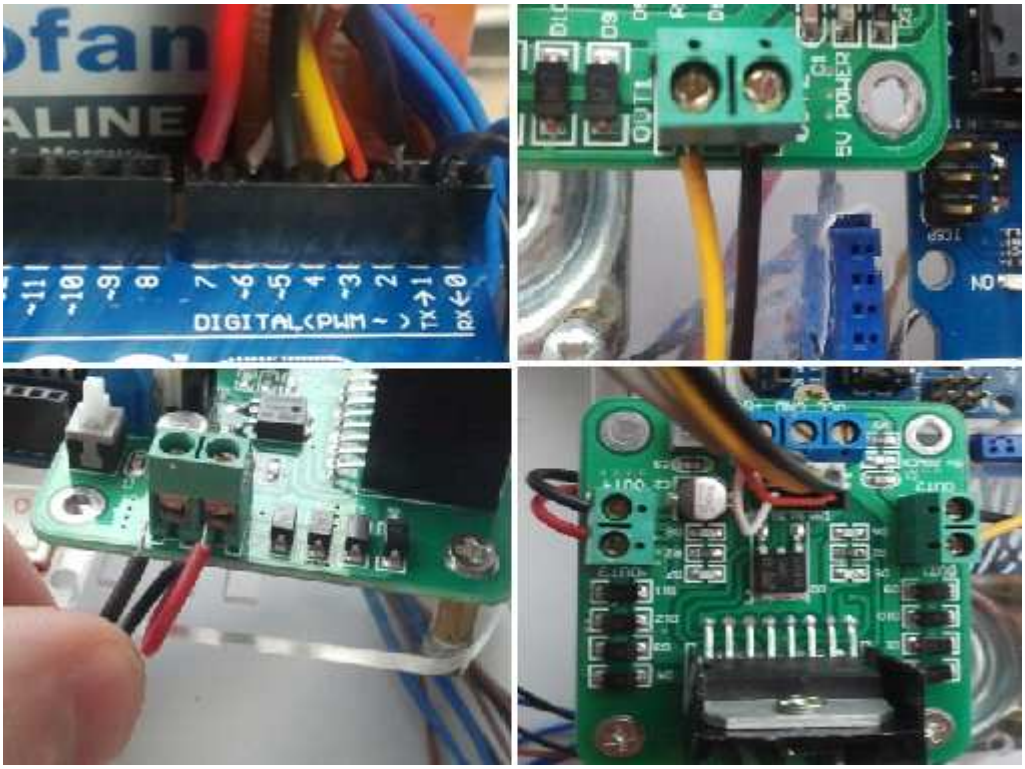



**Ilustración 97:** Conexión controlador motor con la placa Arduino.

Una vez esté hecha la conexión entre Arduino Uno y el controlador motor **(ilustración 97)**, se pasa a conectar el módulo APC220 con Arduino Uno **(ilustración 98 y 101)**. Primeramente se conecta el módulo APC220 en los pines que se desea. En este caso se ha puesto lo más cerca al exterior, después se identifican los cables del conector y se pasa a soldar las puntas del otro extremo del conector, para que encaje bien en la placa arduino.

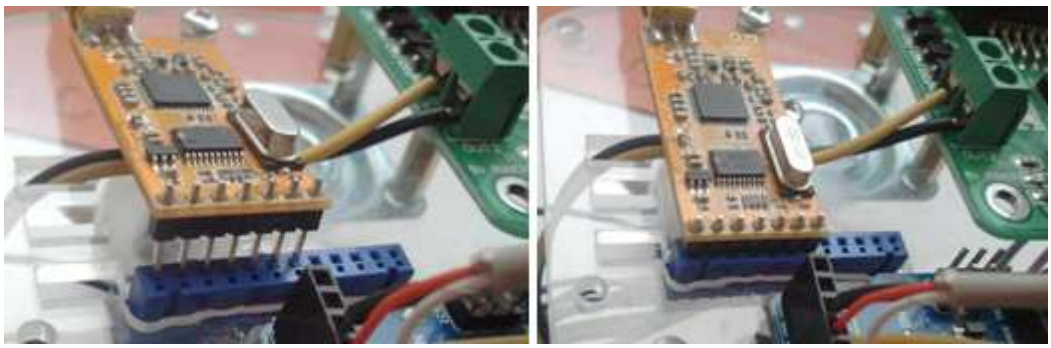


**Ilustración 98:** Conexión Módulo APC220 en el conector.

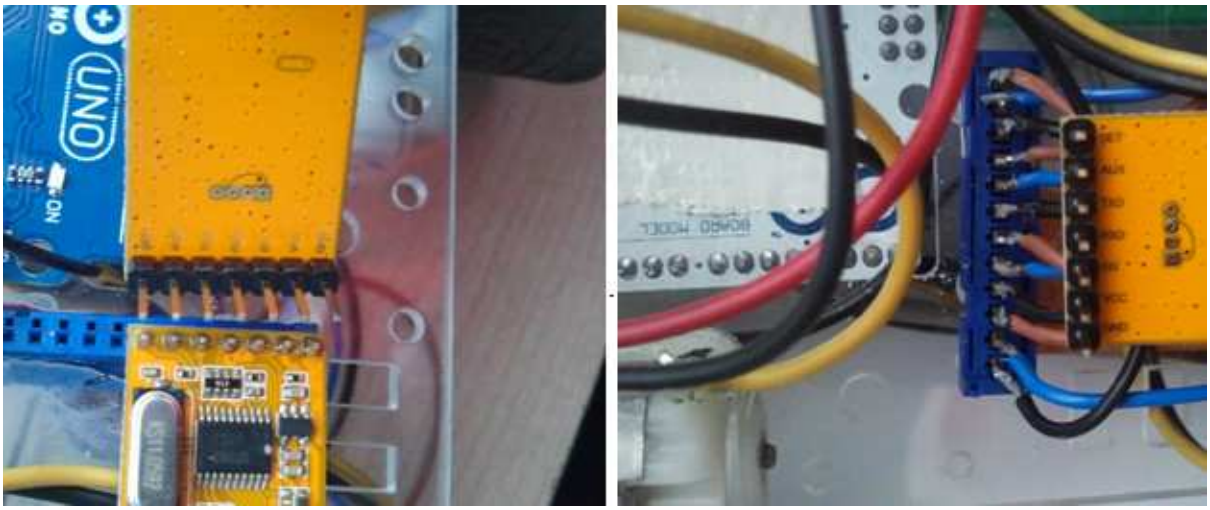


**Ilustración 99:** Identificación de los pines del módulo APC220.

Una vez identificado cada cable del conector **(ilustración 99)**, se conecta a la placa Arduino según el siguiente esquema **(ilustración 100)**:


**Ilustración 100:** Esquema conexión módulo APC220 al Arduino.


**Ilustración 101:** Conexión de los cables del conector del módulo APC220 al Arduino.

No hace falta conectar los pines EN, AUX, SET, del módulo, conectando los cuatro pines dichos anteriormente funciona perfectamente.

Una vez conectada la placa Arduino con el controlador de motores se colocan los cables y se le pone unas bridas **(ilustración 102)** para que estén todos juntos y no puede ver una desconexión por enganche de un cable sobre algún objeto.

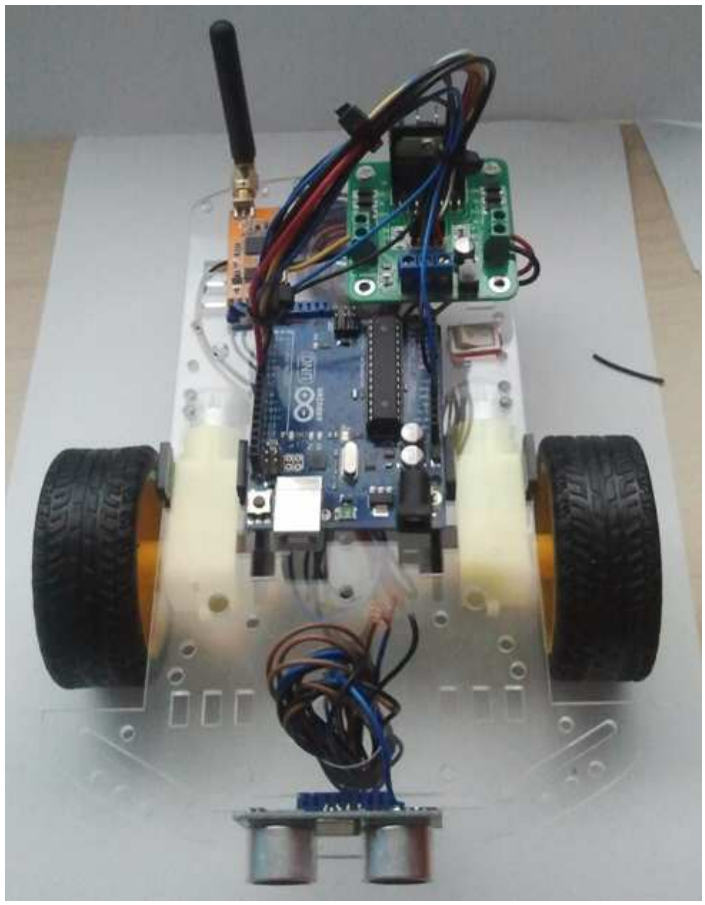


**Ilustración 102:** Robot después del montaje.

Una vez llegado a este punto se puso el robot en funcionamiento para ver como se comportaba con esta configuración de tres ruedas, el resultado no fue muy satisfactorio ya que la rueda trasera tardaba en enderezarse y esto hacia, que cuando se ordenaba que el robot fuera hacia delante la respuesta era en curva. Por lo tanto se llegó a la conclusión de cambiar la rueda trasera giratoria por dos ruedas traseras fijas para conseguir los movimientos en línea recta. Con esta configuración se obtiene un resultado satisfactorio en líneas rectas pero en los giros tiene el inconveniente de que en superficies rugosas los giros son más lentos y forzados. A continuación se describe el proceso del cambio de las ruedas:

Primero se tuvo que desmontar la placa de control de motores y posterior el desmontaje de la rueda trasera móvil, una vez desmontados dichos componentes se hicieron modificaciones en la fase del robot (agujeros y ranuras) para la incorporación de las nuevas ruedas fijas **(ilustración 103)**. En este procedimiento se aprovechó para conexionar el sensor ultrasónico, soldando los cables correspondientes y conexionándolos, más adelante se verá un esquema de conexión.

**Ilustración 103:** Sustitución de rueda móvil por ruedas fijas.

Otra modificación que se hizo fue la colocación del interruptor que aparece en la **ilustración 103** en la parte de la derecha. Su función es la activación y desconexión del módulo APC220, ya que para cargar el programa al robot desde el PC el módulo debe estar desconectado ya que los puertos de comunicación del Arduino y del módulo entra en conflicto y no es posible cargar el programa, por lo tanto se tenía que quitar el módulo y una vez cargado el programa volver a conectarlos en los pines. Con este interruptor lo que se ha conseguido es que cuando se quiera desconectar el módulo no hace falta quitarlo, solo con accionar el interruptor es suficiente, ya que lo que hace es cortar la conexión del

RX y TX entre módulo y Arduino, esto es posible porque el interruptor es doble **(ilustración 104)** con un único mecanismo de accionamiento.

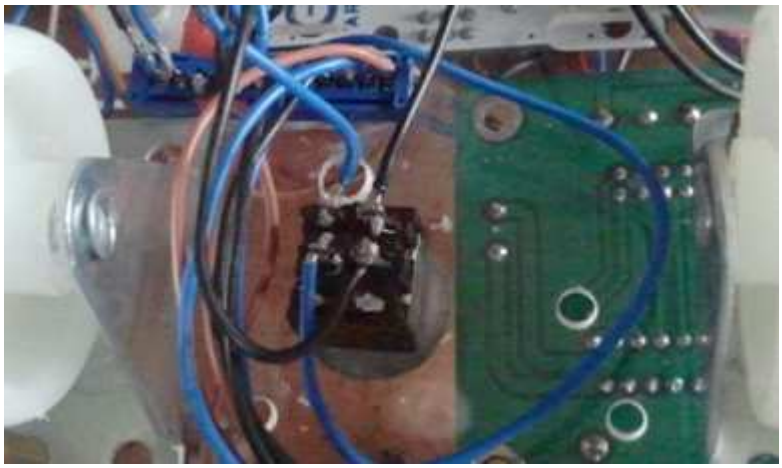


**Ilustración 104:** Interruptor conexión APC220.

Una vez instaladas las ruedas fijas, sensor e interruptor se montaron las baterías de prueba **(ilustración 105)** sobre el robot para su posterior ensayo del comportamiento del robot siendo éste satisfactorio.

**Ilustración 105:** Robot con ruedas fijas y baterías de prueba.

## 4.5 Cálculo de baterías.

Para el cálculo de las baterías se necesita saber el consumo que tienen todos los elementos de nuestro robot, para poder estudiar la capacidad y autonomía que deben tener las baterías del robot. Por lo tanto se describirá a continuación el consumo de cada uno de los elementos:

**Consumo sensor ultrasónico:** corriente de trabajo 15mA.

**Consumo APC220:** la corriente máxima se da en la transmisión de datos, igual a 35mA que para una tensión de 5.5V da una potenca de salida de 20mW. La corriente en reposo es de 5uA.

**Consumo arduino Uno:** el consumo máximo de las entradas/salidas de 50mA. El consumo de placa sin nada conectado 24mA.

**Consumo motores:** los motores consumen como máximo 300mA, por lo tanto entre los dos tenemos 600mA, poniendo una proximación por sobrecarga de 1 A.

**Consumo de controlador motor:** la parte de lógica de control consume alrededor de 36mA.

Sumando todos los consumos a plena carga tenemos que: consumo a plena carga en mA: 15+35+50+24+1000+36=1160mA que se sobredimensionará a **1500mA** para futuras incorporaciones de elementos.

El tiempo de funcionamiento del robot se estimará en **3 horas a plena carga**, obteniendo una bateria de capacidad: 3h*1500mA=**4500mAh.**

Una vez hecho el estudio de consumo y capacidad de la batería se deberá elegir el tipo de bateria que se adapte mejor a la situación de este proyecto.

Opciones de bateria: power one cr123a (cr17345) con capacidad de 1550mAh y 3Vdc. Panasonic con capacidad 1550mAh. Duracel ultra 3V capacidad 1550mA. Sanyo cr123A 3V con capacidad 1400mAh.

Como las pilas que dispongo son las de capacidad 1550mAh, se elegirán estas pilas.

Para conseguir una capacidad de 4500mAh, dividimos la capacidad total entre la capacidad de una pila: 4500/1550=2.90 entonces cogemos 3 pilas puestas en paralelo, haciendo una capacidad total de 3x1550=**4650mAh.** Para dar los 12 voltios deseados para la bateria, hace falta conectar cuatro paquetes en serie (cuatro paquetes de tres pilas conectados en paraledo cada uno) ya que cada paquete son de 3 voltios. Por lo tanto para construir la

bateria se necesitarán 12 pilas, cuatro paquetes conectados en serie, y cada paquete esta formado por tres pilas puestas en paralelo.

### 4.5.1 Construcción de la batería.

Para tener las pilas ordenadas es necesario hacer un compartimento donde todas las pilas estén hubicadas e instaladas todas juntas, constituyendo una batería. Para este compartimento se ha optado por la construcción de un recipiente cuadrado, donde irán todas las pilas metidas dentro. En el exterior del recipiente habrá unas bornas de conexión que será la salida de tensión de la bateria. A continuación se verá el procedimiento llevado:

Primero se necesita un material que pese poco, para no aportar mucho peso al robot, el material elegido es metacrilato flexible de expesor 1mm **(ilustración 106)**, ya que pesa muy poco y a su vez es trasparente danto un mejor aspecto al robot al poder ver las pilas en el interior del recipiente.

**Ilustración 106:** Metacrilato para recipiente batería.

Una vez elegido el material se cortan las piezas a medida **(ilustración 107)** y se pegan con algún adhesivo apropiado para este tipo de materiales, en este caso se ha utilizado una pistola termofusible.

**Ilustración 107:** Corte metacrilato.

Una vez cortadas las piezas se pegan con el pegamento termofusible y se recorta la base. Una vez pegado todo **(ilustración 108)** se recortan los restos de pegamento sobrantes para dejar una superficie lisa.

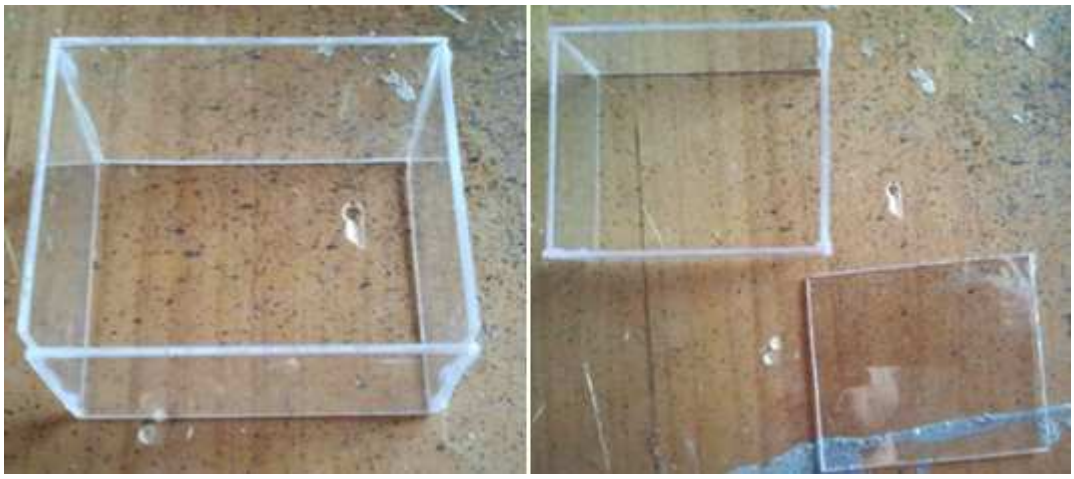


**Ilustración 108:** Pegado de piezas metacrilato.

Para la tapa del recipiente se ha optado por una tapa corredera **(ilustración 109)**, por lo tanto se ha necesitado elaborar unas correderas para la introducción de la tapa.

**Ilustración 109:** Recipiente con tapa corredera.

Una vez hecho el recipiente se ponen los bornes de conexión sobre él, y se conexionan las pilas para la obtención de los 12 voltios deseados. Esta salida se conecta al interruptor principal del robot, y del interruptor se da alimentación al controlador de los motores y a la placa Arduino.

Para la incorporación de la batería en el robot se necesita una tercera modificación, quitando la placa Arduino de la base ya que el recipiente va en la base y colocando la placa Arduino en un lateral de la batería.

## 4.6 Incorporación de cámara en el robot.

Como se ha dicho en apartados anteriores se quiere visualizar el entorno que rodea al robot en el PC, por lo tanto es necesaria la incorporación de una cámara. En este caso se ha optado por utilizar la cámara de los teléfonos móviles, ya que es de gran utilidad la incorporación de teléfonos móviles a proyectos de robótica por la posibilidad de utilización de los sensores del teléfono y otros elementos como puede ser en este caso la cámara. Con esto conseguimos un gran ahorro en la construcción del robot, ya que no tenemos que comprar una cámara aparte (valor aproximado 85 euros).

Para la incorporación del teléfono móvil al robot es necesaria la construcción de un soporte para el móvil. El soporte se hará para que casi todos los teléfonos móviles se puedan poner en él, dándole también un movimiento de inclinación y desplazamiento del teléfono móvil. A continuación se verá el procedimiento para dicha construcción.

Primero se ha elegido el material con el que se hará el soporte. La base será de madera, ya que hay que hacerle unas ranuras **(ilustración 110)** y es el mejor material para realizar modificaciones con facilidad sobre él. Los soportes que agarran el móvil están hechos con metacrilato porque pesa muy poco.

Una vez elegido los materiales se corta la madera y se le hacen las ranuras y agujeros correspondientes para su posterior acoplamiento al robot.

**Ilustración 110:** Ranura en madera para soporte del teléfono

Una vez hecha la ranura para el teléfono móvil **(ilustración 110)** se hace una segunda ranura **(ilustración 111)** para el desplazamiento del soporte para poder ajustar la posición de la cámara en la posición adecuada.

**Ilustración 111:** Segunda ranura del soporte.

Una vez terminadas las ranuras en la madera se pasa a construir la "pinza" **(ilustración 112)** que sujetara el teléfono móvil y le dará la inclinación deseada. Esta pinza está hecha de metacrilato.

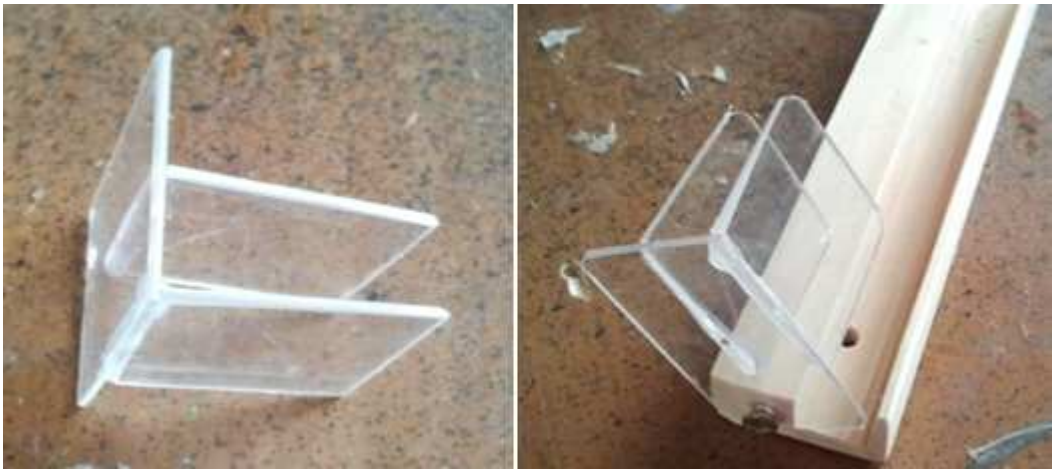


**Ilustración 112:** Pinza para soporte teléfono móvil.

El siguiente paso será pintar la madera y realizar una segunda "pinza" **(ilustración113)** para el otro extremo del teléfono móvil, esta segunda pinza será para adaptar la sujeción correcta del teléfono móvil en función al tamaño de este.

**Ilustración 113:** Soporte teléfono móvil acabado.

Una vez acabado el soporte se instalará en el robot **(ilustración 114)**. Para instalar el soporte será necesario realizar un par de agujeros para agarrar el soporte a la base del robot.

**Ilustración 114:** Robot con soporte teléfono móvil.

### 4.6.1 Uso y configuración de la cámara del teléfono móvil como cámara IP

Para utilizar la cámara de un teléfono móvil como una cámara IP hace falta tener instalada una aplicación en el teléfono móvil y acceso a internet desde el mismo, también hace falta un ordenador conectado a internet. Para algunas de estas aplicaciones es necesario que el ordenador y teléfono estén conectados a la misma red Wifi o a través de la red móvil, pero

existen algunas aplicaciones que no hace falta estar en la misma red Wifi conectados. La aplicación que se ha elegido para este caso se llama **IP Webcam (ilustración 115).**

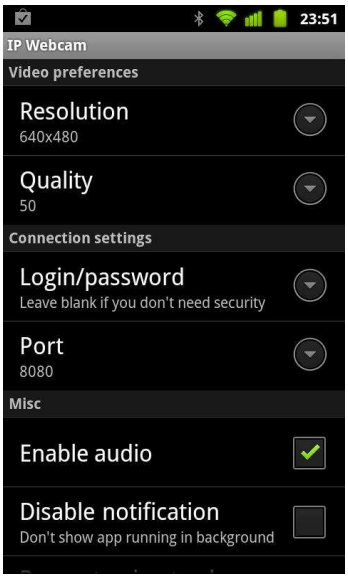



**Ilustración 115:** Aplicación IP Webcam.

Con esta aplicación se puede elegir la resolución, calidad, posicionamiento de cámara (horizontal, vertical), también tiene una opción de hacer uso de la cámara delantera del teléfono móvil y tiene opción de capturar el audio. Una vez configurada la aplicación a nuestra necesidad se activa la aplicación pulsando **Start server.** Cuando puesto en funcionamiento, la aplicación nos da una dirección IP que es la que se tiene que poner en el navegador de internet del ordenador.

Una vez introducida la dirección IP que ha generado la aplicación en el navegador de internet del ordenador aparecerá lo siguiente **(ilustración 116)**:

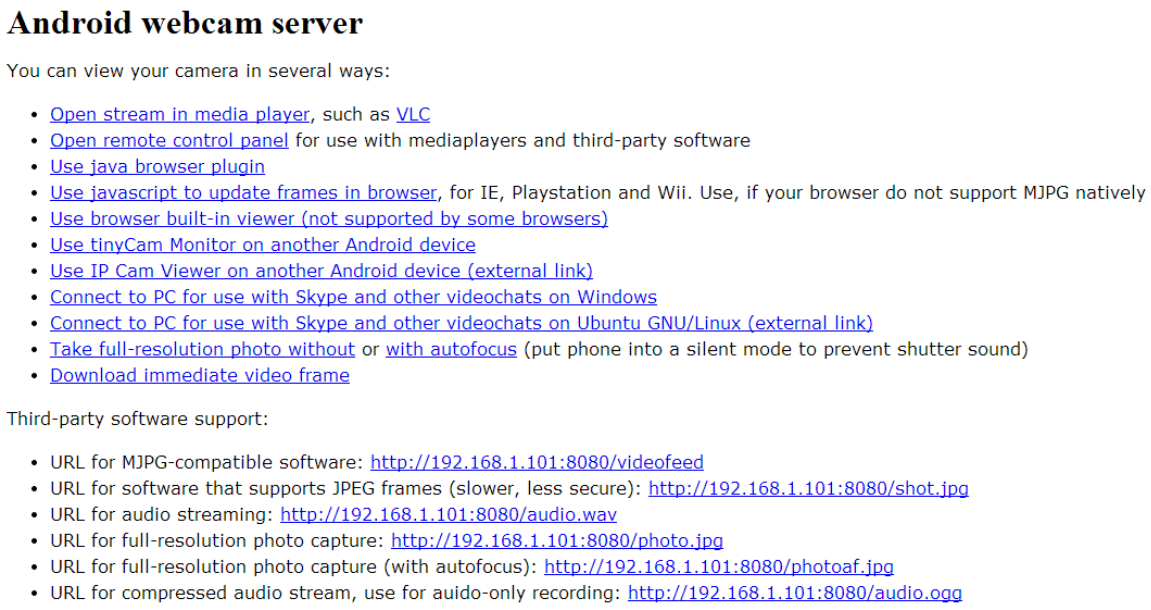



**Ilustración 116:** Opciones cámara IP.

Una vez que tenemos las opciones se elige la más adecuada para el caso a tratar. Para este caso se ha elegido la opción ***Use browser built-in viewer (not supported by some browsers).*** Seleccionando esta opción aparecerá en un recuadro la captura de video que el teléfono móvil está captando una opción por si queremos escuchar el audio.


**Ilustración 117:** Captación de video.

## 4.7 Aspecto y esquema conexiones del robot.

Una vez puesto el soporte del teléfono móvil al robot se queda toda la parte física acabada, teniendo el siguiente aspecto definitivo **(ilustración 118)**.

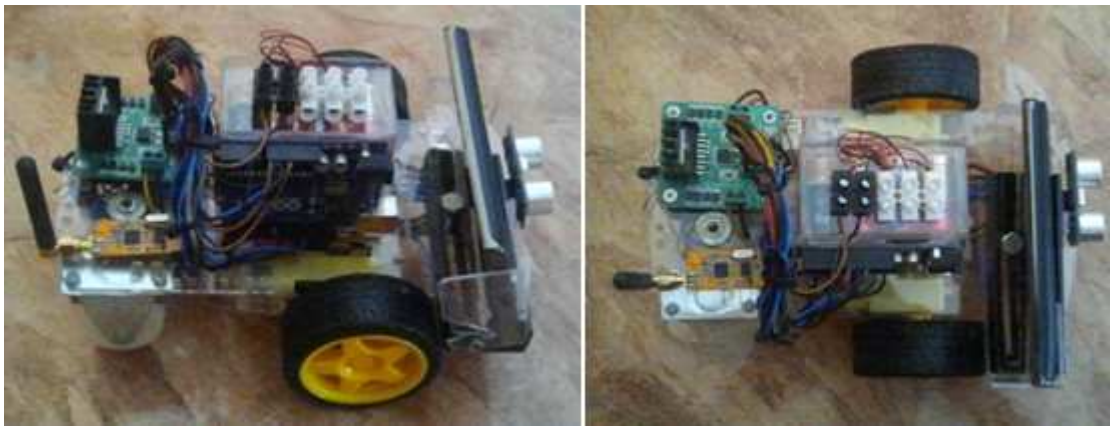


**Ilustración 118:** Robot controlado por voz con lógica difusa.

La conexión eléctrica detallada de todos los elementos del robot se puede ver a continuación **(ilustración 119)**.

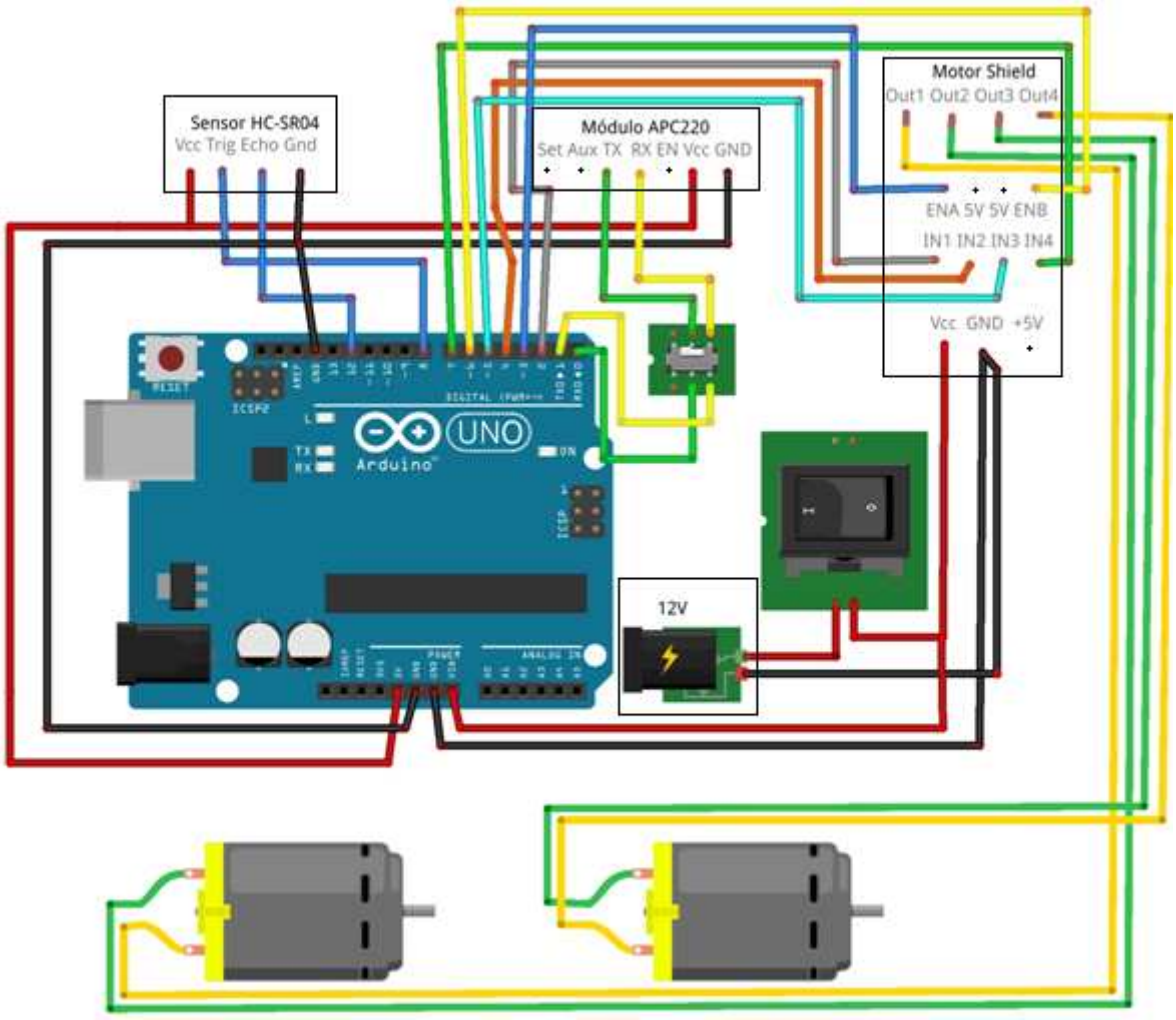



**Ilustración 119:** Esquema conexiones eléctricas.

## 4.8 Funcionamiento robot móvil con control difuso por voz.

El control del robot se hará mediante voz, lo que hace necesario tener un programa de reconocimiento de voz. Los comandos de voz que se desean incorporar al control del robot son:

- **Avanza.**
- **Más deprisa.**
- **Más despacio.**
- **Para.**
- **Gira izquierda**
- **Gira derecha.**
- **Retrocede.**

Para la orden "**avanza"** se empezará siempre con el mismo valor de velocidad, aunque dependerá de la superficie en la que se encuentre el robot, con un valor de 60. Este valor no es la velocidad, si no el valor que da Arduino a su salida PWM para el control de la velocidad de giro de los motores, pero nosotros para mayor comodidad la llamaremos velocidad.

Las órdenes "**más deprisa"** y "**más despacio"** no siempre tienen la misma respuesta en el robot, dependerá de la velocidad del robot en ese momento y de la distancia al obstáculo. "*Más deprisa"* como es lógico acelerará el robot si es posible y "*Más despacio"* disminuirá la velocidad. Todo esto se consigue gracias a la ***LÓGICA DIFUSA.*** Esto será posible dentro de las limitaciones que el robot tiene, porque no podrá superar una cierta velocidad máxima y se tendrá que limitar para que no pase de ese cierto valor por mucho que se ordene "*más deprisa"*.

La orden **"para"** detiene por completo el robot.

Las ordenes **"gira izquierda"** y **"gira derecha"** gira el robot 45º aproximadamente, dependiendo de la rugosidad de la superficie en la que se encuentre el robot. Para girar 90º habría que decir "*gira izquierda" o "gira derecha"* dos veces y para 180º cuatro veces.

**"Retrocede"** hace que el robot retroceda una distancia prefijada (en realidad tiempo prefijado), está programado para 20 cm aproximadamente dependiendo de la superficie por donde se mueva el robot.

A parte del control por voz el robot está dotado de un segundo control mediante joystick, para situaciones en las que la comunicación por voz sea difícil debido a ruidos.

## 4.9 Programa reconocimiento de voz con MATLAB.

Cuando se comenzó con la realización de reconocimiento de voz se planteó en una primera fase crear un programa donde siempre estuviera a la escucha para no tener que activar ningún botón cada vez que se quiera reconocer una palabra. Este programa tenía el inconveniente de que cualquier ruido mientras se estaba en silencio detectaba los pequeños ruidos y lo comparaba con los patrones, dando como resultado un reconocimiento no deseado. Para eliminar este problema se grabaron distintos silencios que se podrían dar al

estar en silencio, llegando a un resultado casi satisfactorio. Tenía el inconveniente de cuando se decía una frase, por ejemplo, *más deprisa*, al tener otra frase con *más despacio*, no distinguía cuál de ellas se quería reconocer, dando como resultado unas veces más deprisa y otras más despacio. En la versión final del reconocedor este problema se elimina con el ventaneo.

En el primer reconocedor se intentó también que pudiera reconocer órdenes de distancias concretas, pero este se hacía muy lento al ir reconociendo paso por paso, ya que primero tenía que reconocer avanza, luego pasaba a un segundo nivel donde era la cantidad y después pasar a un tercer nivel donde tenía que reconocer la magnitud. Esto se hacía muy lento y no fue preciso. Para este reconocedor se utilizaba la transformada rápida de Fourier para obtener las señales de la voz y se compara el valor absoluto de la diferencia de los valores de la transformada de Fourier de los patrones guardados y la transformada de la orden a reconocer. No se explicará más sobre este programa ya que no se obtuvieron buenos resultados. Los códigos para este reconocedor de voz es el siguiente:

```
function varargout = interfazcontrolvoz(varargin)
% INTERFAZCONTROLVOZ M-file for interfazcontrolvoz.fig
%      INTERFAZCONTROLVOZ, by itself, creates a new INTERFAZCONTROLVOZ or raises
the existing
%      singleton*.
%
%      H = INTERFAZCONTROLVOZ returns the handle to a new INTERFAZCONTROLVOZ or
the handle to
%      the existing singleton*.
%
%      INTERFAZCONTROLVOZ('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in INTERFAZCONTROLVOZ.M with the given input
arguments.
%
%      INTERFAZCONTROLVOZ('Property','Value',...) creates a new
INTERFAZCONTROLVOZ or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before interfazcontrolvoz_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to interfazcontrolvoz_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help interfazcontrolvoz

% Last Modified by GUIDE v2.5 19-Nov-2013 11:40:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
```

```
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interfazcontrolvoz_OpeningFcn, ...
                   'gui_OutputFcn',  @interfazcontrolvoz_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before interfazcontrolvoz is made visible.
function interfazcontrolvoz_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interfazcontrolvoz (see VARARGIN)
% Choose default command line output for interfazcontrolvoz
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes interfazcontrolvoz wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = interfazcontrolvoz_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in ordeneson.
function ordeneson_Callback(hObject, eventdata, handles)
global on y fs %Establecer varialbles globales
on=1;
% Parámetros entrenados para reconocer
av = wavread('avanza');    % leo los archivos .wav q tenemos almacenados
fr = wavread('frena');
re = wavread('retro');
iz= wavread ('i');
de= wavread ('d');
si= wavread ('silencio');
si2 = wavread('silencio2');
si3= wavread ('silencio3');
si4= wavread ('silencio4');
si5= wavread ('silencio5');
ru= wavread ('ruido');
ru3= wavread ('ruido3');
ru4= wavread ('ruido4');

%--- Uso de la Función Normalizar
orden1 = normalizar(av);   % normalizo los archivos de audio
orden2 = normalizar(fr);
orden3= normalizar (re);
orden4= normalizar (iz);
```

```
orden5= normalizar (de);
orden6= normalizar (si);
orden7= normalizar (si2);
orden8= normalizar (si3);
orden9= normalizar (si4);
orden10= normalizar (si5);
orden11= normalizar (ru);
orden12= normalizar (ru3);
orden13= normalizar (ru4);

%---- Transformada de las Letras y Números
transorden1 = abs((fft(orden1)));  % encuentro la transformada rápida de Fourier
y su valor absoluto
transorden2 = abs((fft(orden2)));
transorden3 =abs((fft(orden3)));
transorden4=abs((fft(orden4)));
transorden5=abs((fft(orden5)));
transorden6=abs((fft(orden6)));
transorden7=abs((fft(orden7)));
transorden8=abs((fft(orden8)));
transorden9=abs((fft(orden9)));
transorden10=abs((fft(orden10)));
transorden11=abs((fft(orden11)));
transorden12=abs((fft(orden12)));
transorden13=abs((fft(orden13)));

% Se realiza el proceso para grabar audio en tiempo real
%COMANDOS BASICOS
while (on == 1); %activar cuando se pulsa ordenes on
clc %limpia la pantalla
fs=11025; %frecuencia de muestreo
tiempograb=1; %Tiempo de grabación
y=wavrecord(tiempograb*fs,fs,1); %función de grabación
%soundsc(y,fs); %Reproduce grabación
ts=1/fs;
t=0:ts:tiempograb-ts;  % vector de tiempo
b=[1 -0.95];
yf=filter(b,1,y); %Proceso de filtrado
len = length(y); %longitud del vector
avg_e = sum(y.*y)/len; %promedio señal entera
THRES = 0.2;
wavwrite(yf,fs,'voz'); %Graba en archivo .wav el audio q pronunciamos
axes(handles.axes1);
plot(t,yf);grid on; % Grafica en los axes
%NORMALIZACIÓN
N=length(y); %calcula el tamaño del vector
f=(0:N-1)*fs/N;
%----- Proceso de Reconocimiento
voz_usuario=wavread('voz');   % leo el archivo de audio q grabamos en tiempo real
usuario=normalizar(voz_usuario); % normalizo el archivo de audio
transusuario=abs((fft(usuario))); % encuentro la transformada rápida de Fourier y
su valor absoluto
av1=transorden1-transusuario; % realizo la resta de las señales procesadas (audio
base de datos - audio grabado en tiempo real)
fr1=transorden2-transusuario;
re1=transorden3-transusuario;
iz1=transorden4-transusuario;
de1=transorden5-transusuario;
si1=transorden6-transusuario;
si21=transorden7-transusuario;
si31=transorden8-transusuario;
si41=transorden9-transusuario;
si51=transorden10-transusuario;
ru1=transorden11-transusuario;
ru31=transorden12-transusuario;
ru41=transorden13-transusuario;
error(1)=mean(abs(av1));  % mean = Devuelve los valores de media de los elementos
```

```
error(2)=mean(abs(fr1));
error(3)=mean(abs(re1));
error(4)=mean(abs(iz1));
error(5)=mean(abs(de1));
error(6)=mean(abs(si1));
error(7)=mean(abs(si21));
error(8)=mean(abs(si31));
error(9)=mean(abs(si41));
error(10)=mean(abs(si51));
error(11)=mean(abs(ru1));
error(12)=mean(abs(ru31));
error(13)=mean(abs(ru41));
min_error=min(error); %Almacena el menor error

%display(error) %Muestra el vector error
%display(min_error) %Muestra el menor error

if(min_error == error(1)) % realizo la comparación del mínimo error
    x=0;
    x3=0;
    x5=0;
    set (handles.texto, 'String', 'AVANZA'); % Muestra la letra comparada
    %ORDENES PARA EL COMANDO AVANZA
    while ((x==0)&&(on == 1));
     %limpia la pantalla
    fs=11025; %frecuencia de muestreo
    tiempograb=1; %Tiempo de grabación
    y=wavrecord(tiempograb*fs,fs,1); %función de grabación
    %soundsc(y,fs); %Reproduce grabación
    ts=1/fs;
    t=0:ts:tiempograb-ts;  % vector de tiempo
    b=[1 -0.95];
    yf=filter(b,1,y); %Proceso de filtrado
    len = length(y); %longitud del vector
    avg_e = sum(y.*y)/len; %promedio señal entera
    THRES = 0.2;
    wavwrite(yf,fs,'voz'); %Graba en archivo .wav el audio que pronunciamos
    axes(handles.axes2);
    plot(t,yf);grid on;% Grafica en los axes
    %NORMALIZACIÓN
    N=length(y); %calcula el tamaño del vector
    f=(0:N-1)*fs/N;

    p1=wavread('1');
    p2=wavread('2');
    p3=wavread('3');
    p4=wavread('4');
    p5=wavread('5');
    p6=wavread('6');
    p7=wavread('7');
    p8=wavread('8');
    p9=wavread('9');
    poco=wavread('poco');
    mucho=wavread('mucho');

    orden14=normalizar(p1);
    orden15=normalizar(p2);
    orden16=normalizar(p3);
    orden17=normalizar(p4);
    orden18=normalizar(p5);
    orden19=normalizar(p6);
    orden20=normalizar(p7);
    orden21=normalizar(p8);
    orden22=normalizar(p9);
    orden23=normalizar(poco);
    orden24=normalizar(mucho);
    transorden14=abs((fft(orden14)));
```

```
transorden15=abs((fft(orden15)));
transorden16=abs((fft(orden16)));
transorden17=abs((fft(orden17)));
transorden18=abs((fft(orden18)));
transorden19=abs((fft(orden19)));
transorden20=abs((fft(orden20)));
transorden21=abs((fft(orden21)));
transorden22=abs((fft(orden22)));
transorden23=abs((fft(orden23)));
transorden24=abs((fft(orden24)));

%----- Proceso de Reconocimiento
voz_usuario=wavread('voz');% leo el archivo de audio q grabamos en
%tiempo real
usuario=normalizar(voz_usuario); % normalizo el archivo de audio
transusuario=abs((fft(usuario)));% encuentro la transformada rápida
%de Fourier y su valor absoluto
si1=transorden6-transusuario;
si21=transorden7-transusuario;
si31=transorden8-transusuario;
si41=transorden9-transusuario;
si51=transorden10-transusuario;
ru1=transorden11-transusuario;
ru31=transorden12-transusuario;
ru41=transorden13-transusuario;
p11=transorden14-transusuario; % realizo la resta de las señales
%procesadas (audio base de datos - audio grabado en tiempo real)
p21=transorden15-transusuario;
p31=transorden16-transusuario;
p41=transorden17-transusuario;
p51=transorden18-transusuario;
p61=transorden19-transusuario;
p71=transorden20-transusuario;
p81=transorden21-transusuario;
p91=transorden22-transusuario;
poco1=transorden23-transusuario;
mucho1=transorden24-transusuario;

error2(1)=mean(abs(p11));  % mean = Devuelve los valores de media de
%los elementos
error2(2)=mean(abs(p21));
error2(3)=mean(abs(p31));
error2(4)=mean(abs(p41));
error2(5)=mean(abs(p51));
error2(6)=mean(abs(p61));
error2(7)=mean(abs(p71));
error2(8)=mean(abs(p81));
error2(9)=mean(abs(p91));
error2(10)=mean(abs(poco1));
error2(11)=mean(abs(mucho1));
error2(12)=mean(abs(si1));
error2(13)=mean(abs(si21));
error2(14)=mean(abs(si31));
error2(15)=mean(abs(si41));
error2(16)=mean(abs(si51));
error2(17)=mean(abs(ru1));
error2(18)=mean(abs(ru31));
error2(19)=mean(abs(ru41));
min_error2=min(error2);

    if (min_error2 == error2(1))

        set (handles.parametro, 'string', 'Uno')
        x=x+1;

    elseif (min_error2 == error2(2))
```

```
    set (handles.parametro, 'string', 'Dos')
    x=x+1;

elseif (min_error2 == error2(3))

    set (handles.parametro, 'string', 'Tres')
    x=x+1;

elseif(min_error2 == error2(4))

    set (handles.parametro, 'string','Cuatro');
    x=x+1;

elseif(min_error2 == error2(5))

    set (handles.parametro, 'string','Cinco');
    x=x+1;

elseif(min_error2 == error2(6))

    set (handles.parametro, 'string','Seis');
    x=x+1;

elseif(min_error2 == error2(7))

    set (handles.parametro, 'string','Siete');
    x=x+1;

elseif(min_error2 == error2(8))

    set (handles.parametro, 'string','Ocho');
    x=x+1;

elseif(min_error2 == error2(9))

    set (handles.parametro, 'string','Nueve');
    x=x+1;

elseif (min_error2 == error2(10))

    set (handles.parametro, 'string', 'Poco')
    x=x+1;
    x7=0;
    x3=1;
    x5=1;

elseif (min_error2 == error2(11))

    set (handles.parametro, 'string', 'Mucho')
    x=x+1;
    x7=0;
    x3=1;
    x5=1;

elseif(min_error2 == error2(12))

    set (handles.parametro, 'string','si1');

elseif(min_error2 == error2(13))

    set (handles.parametro, 'string','si2');

elseif(min_error2 == error2(14))

    set (handles.parametro, 'string','si3');

elseif(min_error2 == error2(15))
```

```
        set (handles.parametro, 'string','si4');

    elseif(min_error2 == error2(16))

        set (handles.parametro, 'string','si5');

    elseif(min_error2 == error2(17))

        set (handles.parametro, 'string','ru1');

    elseif(min_error2 == error2(18))

        set (handles.parametro, 'string','ru3');

    elseif(min_error2 == error2(19))

        set (handles.parametro, 'string','ru4');

    end

end

    if ((x==1)&&(x5==0));
        x3=0;
        x7=0;
        %DEFINICIÓN DE UNIDAD
        while ((on==1)&&(x3==0));
        fs=11025; %frecuencia de muestreo
        tiempograb=1; %Tiempo de grabación
        y=wavrecord(tiempograb*fs,fs,1); %función de grabación
        %soundsc(y,fs); %Reproduce grabación
        ts=1/fs;
        t=0:ts:tiempograb-ts;  % vector de tiempo
        b=[1 -0.95];
        yf=filter(b,1,y); %Proceso de filtrado
        len = length(y); %longitud del vector
        avg_e = sum(y.*y)/len; %promedio señal entera
        THRES = 0.2;
        wavwrite(yf,fs,'voz'); %Graba en archivo .wav el audio q
  %pronunciamos
        axes(handles.axes2);
        plot(t,yf);grid on;% Grafica en los axes
        %NORMALIZACIÓN
        N=length(y); %calcula el tamaño del vector
        f=(0:N-1)*fs/N;
        metro=wavread('metro');
        deci=wavread('decimetro');
        centi=wavread('centimetro');
        atras=wavread('atras');
        orden25=normalizar(metro);
        orden26=normalizar(deci);
        orden27=normalizar(centi);
        orden30=normalizar(atras);
        transorden25=abs((fft(orden25)));
        transorden26=abs((fft(orden26)));
        transorden27=abs((fft(orden27)));
        transorden30=abs((fft(orden30)));

        %----- Proceso de Reconocimiento
        voz_usuario=wavread('voz');% leo el archivo de audio q
  %grabamos en tiempo real
        usuario=normalizar(voz_usuario); % normalizo el archivo de
  %audio
        transusuario=abs((fft(usuario)));% encuentro la transformada
  %rápida de Fourier y su valor absoluto
        si1=transorden6-transusuario;
```

```
      si21=transorden7-transusuario;
      si31=transorden8-transusuario;
      si41=transorden9-transusuario;
      si51=transorden10-transusuario;
      ru1=transorden11-transusuario;
      ru31=transorden12-transusuario;
      ru41=transorden13-transusuario;
      metro1=transorden25-transusuario; % realizo la resta de las
%señales procesadas (audio base de datos - audio grabado en
%tiempo real)
      deci1=transorden26-transusuario;
      centi1=transorden27-transusuario;
      atras1=transorden30-transusuario;

      error4(1)=mean(abs(metro1));  % mean = Devuelve los valores
%de media de los elementos
      error4(2)=mean(abs(deci1));
      error4(3)=mean(abs(centi1));
      error4(4)=mean(abs(si1));
      error4(5)=mean(abs(si21));
      error4(6)=mean(abs(si31));
      error4(7)=mean(abs(si41));
      error4(8)=mean(abs(si51));
      error4(9)=mean(abs(ru1));
      error4(10)=mean(abs(ru31));
      error4(11)=mean(abs(ru41));
      error4(12)=mean(abs(atras1));
      min_error4=min(error4);

      if (min_error4 == error4(1))

      set (handles.unidad, 'string', 'Metro')
      x3=1;

      elseif (min_error4 == error4(2))

      set (handles.unidad, 'string', 'Decímetro')
      x3=1;

      elseif (min_error4 == error4(3))

      set (handles.unidad, 'string', 'Centímetro')
      x3=1;

      elseif (min_error4 == error4(4))

      set (handles.unidad, 'string','1');

      elseif(min_error4 == error4(5))

          set (handles.unidad, 'string','2');

      elseif(min_error4 == error4(6))

          set (handles.unidad, 'string','3');

      elseif(min_error4 == error4(7))

          set (handles.unidad, 'string','4');

      elseif(min_error4 == error4(8))

          set (handles.unidad, 'string','5');

      elseif(min_error4 == error4(9))

          set (handles.unidad, 'string','ruido');
```

```
        elseif(min_error4 == error4(10))

            set (handles.unidad, 'string','ruido3');

        elseif(min_error4 == error4(11))

            set (handles.unidad, 'string','ruido4');

        elseif(min_error4 == error4(12))

            set (handles.unidad, 'string','atras');
            x3=1;
            x=0;
            x7=1;

        end
        end
    end

    if ((x3==1)&&(x7==0));
        x2=0;
        %ÓRDENES DIFUSAS
        while ((on==1)&&(x2==0));
        fs=11025; %frecuencia de muestreo
        tiempograb=1; %Tiempo de grabación
        y=wavrecord(tiempograb*fs,fs,1); %función de grabación
        %soundsc(y,fs); %Reproduce grabación
        ts=1/fs;
        t=0:ts:tiempograb-ts;  % vector de tiempo
        b=[1 -0.95];
        yf=filter(b,1,y); %Proceso de filtrado
        len = length(y); %longitud del vector
        avg_e = sum(y.*y)/len; %promedio señal entera
        THRES = 0.2;
        wavwrite(yf,fs,'voz'); %Graba en archivo .wav el audio q
%pronunciamos
        axes(handles.axes3);
        plot(t,yf);grid on;% Grafica en los axes
        %NORMALIZACIÓN
        N=length(y); %calcula el tamaño del vector
        f=(0:N-1)*fs/N;

        rap=wavread('rapido');
        len=wavread('lento');
        orden28=normalizar(rap);
        orden29=normalizar(len);
        transorden28=abs((fft(orden28)));
        transorden29=abs((fft(orden29)));

        %----- Proceso de Reconocimiento
        voz_usuario=wavread('voz');% leo el archivo de audio q
        %grabamos en tiempo real
        usuario=normalizar(voz_usuario); % normalizo el archivo de
         %audio
        transusuario=abs((fft(usuario)));% encuentro la transformada
%rápida de Fourier y su valor absoluto
        si1=transorden6-transusuario;
        si21=transorden7-transusuario;
        si31=transorden8-transusuario;
        si41=transorden9-transusuario;
        si51=transorden10-transusuario;
        ru1=transorden11-transusuario;
        ru31=transorden12-transusuario;
        ru41=transorden13-transusuario;
```

```
            rap1=transorden28-transusuario; % realizo la resta de las
        %señales procesadas (audio base de datos - audio grabado en
        %tiempo real)
            len1=transorden29-transusuario;

            error3(1)=mean(abs(rap1));  % mean = Devuelve los valores de
        %media de los elementos
            error3(2)=mean(abs(len1));
            error3(3)=mean(abs(si1));
            error3(4)=mean(abs(si21));
            error3(5)=mean(abs(si31));
            error3(6)=mean(abs(si41));
            error3(7)=mean(abs(si51));
            error3(8)=mean(abs(ru1));
            error3(9)=mean(abs(ru31));
            error3(10)=mean(abs(ru41));
            min_error3=min(error3);

            if (min_error3 == error3(1))

            set (handles.difuso, 'string', 'rapido')
            x2=1;

            elseif (min_error3 == error3(2))

            set (handles.difuso, 'string', 'lento')
            x2=1;

            elseif (min_error3 == error3(3))

            set (handles.difuso, 'string','1');

            elseif(min_error3 == error3(4))

                set (handles.difuso, 'string','2');

            elseif(min_error3 == error3(5))

                set (handles.difuso, 'string','3');

            elseif(min_error3 == error3(6))

                set (handles.difuso, 'string','4');

            elseif(min_error3 == error3(7))

                set (handles.difuso, 'string','5');

            elseif(min_error3 == error3(8))

                set (handles.difuso, 'string','ruido');

            elseif(min_error3 == error3(9))

                set (handles.difuso, 'string','ruido3');

            elseif(min_error3 == error3(10))

                set (handles.difuso, 'string','ruido4');

            end
            end
         end
end

if(min_error == error(2))
    set (handles.texto, 'string', 'FRENA');
```

```
x6=0;
while ((x6==0)&&(on == 1));
 %limpia la pantalla
fs=11025; %frecuencia de muestreo
tiempograb=1; %Tiempo de grabación
y=wavrecord(tiempograb*fs,fs,1); %función de grabación
%soundsc(y,fs); %Reproduce grabación
ts=1/fs;
t=0:ts:tiempograb-ts;  % vector de tiempo
b=[1 -0.95];
yf=filter(b,1,y); %Proceso de filtrado
len = length(y); %longitud del vector
avg_e = sum(y.*y)/len; %promedio señal entera
THRES = 0.2;
wavwrite(yf,fs,'voz'); %Graba en archivo .wav el audio q pronunciamos
axes(handles.axes2);
plot(t,yf);grid on;% Grafica en los axes
%NORMALIZACIÓN
N=length(y); %calcula el tamaño del vector
f=(0:N-1)*fs/N;

poco=wavread('poco');
mucho=wavread('mucho');

orden23=normalizar(poco);
orden24=normalizar(mucho);
transorden23=abs((fft(orden23)));
transorden24=abs((fft(orden24)));

%----- Proceso de Reconocimiento
voz_usuario=wavread('voz');% leo el archivo de audio q grabamos en
%tiempo real
usuario=normalizar(voz_usuario); % normalizo el archivo de audio
transusuario=abs((fft(usuario)));% encuentro la transformada rápida
%de Fourier y su valor absoluto
si1=transorden6-transusuario;
si21=transorden7-transusuario;
si31=transorden8-transusuario;
si41=transorden9-transusuario;
si51=transorden10-transusuario;
ru1=transorden11-transusuario;
ru31=transorden12-transusuario;
ru41=transorden13-transusuario;
poco1=transorden23-transusuario;
mucho1=transorden24-transusuario;

error5(1)=mean(abs(poco1));
error5(2)=mean(abs(mucho1));
error5(3)=mean(abs(si1));
error5(4)=mean(abs(si21));
error5(5)=mean(abs(si31));
error5(6)=mean(abs(si41));
error5(7)=mean(abs(si51));
error5(8)=mean(abs(ru1));
error5(9)=mean(abs(ru31));
error5(10)=mean(abs(ru41));
min_error5=min(error5);

if (min_error5 == error5(1))

    set (handles.difuso, 'string', 'Poco')
    x6=1;

elseif (min_error5 == error5(2))

    set (handles.difuso, 'string', 'Mucho')
    x6=1;
```

```
    elseif (min_error5 == error5(3))

        set (handles.difuso, 'string','1');

    elseif(min_error5 == error5(4))

        set (handles.difuso, 'string','2');

    elseif(min_error5 == error5(5))

        set (handles.difuso, 'string','3');

    elseif(min_error5 == error5(6))

        set (handles.difuso, 'string','4');

    elseif(min_error5 == error5(7))

        set (handles.difuso, 'string','5');

    elseif(min_error5 == error5(8))

        set (handles.difuso, 'string','ruido');

    elseif(min_error5 == error5(9))

        set (handles.difuso, 'string','ruido3');

    elseif(min_error5 == error5(10))

        set (handles.difuso, 'string','ruido4');
    end
    end
end

if(min_error == error(3))

    set (handles.texto, 'string', 'RETROCEDE');

end

if(min_error == error(4))

    set (handles.texto, 'string', 'IZQUIERDA');
    %ORDENES PARA EL COMANDO AVANZA
     x8=0;
    while ((x8==0)&&(on == 1));
     %limpia la pantalla
    fs=11025; %frecuencia de muestreo
    tiempograb=1; %Tiempo de grabación
    y=wavrecord(tiempograb*fs,fs,1); %función de grabación
    %soundsc(y,fs); %Reproduce grabación
    ts=1/fs;
    t=0:ts:tiempograb-ts;  % vector de tiempo
    b=[1 -0.95];
    yf=filter(b,1,y); %Proceso de filtrado
    len = length(y); %longitud del vector
    avg_e = sum(y.*y)/len; %promedio señal entera
    THRES = 0.2;
    wavwrite(yf,fs,'voz'); %Graba en archivo .wav el audio q pronunciamos
    axes(handles.axes2);
    plot(t,yf);grid on;% Grafica en los axes
    %NORMALIZACIÓN
    N=length(y); %calcula el tamaño del vector
    f=(0:N-1)*fs/N;

    grado15=wavread('15');
```

```
grado30=wavread('30');
grado45=wavread('45');
grado90=wavread('90');

orden31=normalizar(grado15);
orden32=normalizar(grado30);
orden33=normalizar(grado45);
orden34=normalizar(grado90);

transorden31=abs((fft(orden31)));
transorden32=abs((fft(orden32)));
transorden33=abs((fft(orden33)));
transorden34=abs((fft(orden34)));

%----- Proceso de Reconocimiento
voz_usuario=wavread('voz');% leo el archivo de audio q grabamos en
%tiempo real
usuario=normalizar(voz_usuario); % normalizo el archivo de audio
transusuario=abs((fft(usuario)));% encuentro la transformada rápida
%de Fourier y su valor absoluto
si1=transorden6-transusuario;
si21=transorden7-transusuario;
si31=transorden8-transusuario;
si41=transorden9-transusuario;
si51=transorden10-transusuario;
ru1=transorden11-transusuario;
ru31=transorden12-transusuario;
ru41=transorden13-transusuario;
grado151=transorden31-transusuario; % realizo la resta de las señales
%procesadas (audio base de datos - audio grabado en tiempo real)
grado301=transorden32-transusuario;
grado451=transorden33-transusuario;
grado901=transorden34-transusuario;

error6(1)=mean(abs(grado151));  % mean = Devuelve los valores de
%media de los elementos
error6(2)=mean(abs(grado301));
error6(3)=mean(abs(grado451));
error6(4)=mean(abs(grado901));
error6(5)=mean(abs(si1));
error6(6)=mean(abs(si21));
error6(7)=mean(abs(si31));
error6(8)=mean(abs(si41));
error6(9)=mean(abs(si51));
error6(10)=mean(abs(ru1));
error6(11)=mean(abs(ru31));
error6(12)=mean(abs(ru41));
min_error6=min(error6);

if (min_error6 == error6(1))

    set (handles.parametro, 'string', '15º')
    x8=1;

elseif (min_error6 == error6(2))

    set (handles.parametro, 'string', '30º')
    x8=1;

elseif (min_error6 == error6(3))

    set (handles.parametro, 'string', '45º')
    x8=1;

elseif (min_error6 == error6(4))

    set (handles.parametro, 'string', '90º')
```

```
        x8=1;
    elseif (min_error6 == error6(5))
        set (handles.parametro, 'string','1');
    elseif(min_error6 == error6(6))
        set (handles.parametro, 'string','2');
    elseif(min_error6 == error6(7))
        set (handles.parametro, 'string','3');
    elseif(min_error6 == error6(8))
        set (handles.parametro, 'string','4');
    elseif(min_error6 == error6(9))
        set (handles.parametro, 'string','5');
    elseif(min_error6 == error6(10))
        set (handles.parametro, 'string','ruido');
    elseif(min_error6 == error6(11))
        set (handles.parametro, 'string','ruido3');
    elseif(min_error6 == error6(12))
        set (handles.parametro, 'string','ruido4');
    end
    end
end

if(min_error == error(5))

    set (handles.texto, 'string', 'DERECHA');
    %ORDENES PARA EL COMANDO AVANZA
     x9=0;
    while ((x9==0)&&(on == 1));
     %limpia la pantalla
    fs=11025; %frecuencia de muestreo
    tiempograb=1; %Tiempo de grabación
    y=wavrecord(tiempograb*fs,fs,1); %función de grabación
    %soundsc(y,fs); %Reproduce grabación
    ts=1/fs;
    t=0:ts:tiempograb-ts;  % vector de tiempo
    b=[1 -0.95];
    yf=filter(b,1,y); %Proceso de filtrado
    len = length(y); %longitud del vector
    avg_e = sum(y.*y)/len; %promedio señal entera
    THRES = 0.2;
    wavwrite(yf,fs,'voz'); %Graba en archivo .wav el audio q pronunciamos
    axes(handles.axes2);
    plot(t,yf);grid on;% Grafica en los axes
    %NORMALIZACIÓN
    N=length(y); %calcula el tamaño del vector
    f=(0:N-1)*fs/N;

    grado15=wavread('15');
    grado30=wavread('30');
    grado45=wavread('45');
    grado90=wavread('90');
```

```
orden31=normalizar(grado15);
orden32=normalizar(grado30);
orden33=normalizar(grado45);
orden34=normalizar(grado90);

transorden31=abs((fft(orden31)));
transorden32=abs((fft(orden32)));
transorden33=abs((fft(orden33)));
transorden34=abs((fft(orden34)));

%----- Proceso de Reconocimiento
voz_usuario=wavread('voz');% leo el archivo de audio q grabamos en
%tiempo real
usuario=normalizar(voz_usuario); % normalizo el archivo de audio
transusuario=abs((fft(usuario)));% encuentro la transformada rápida
%de Fourier y su valor absoluto
si1=transorden6-transusuario;
si21=transorden7-transusuario;
si31=transorden8-transusuario;
si41=transorden9-transusuario;
si51=transorden10-transusuario;
ru1=transorden11-transusuario;
ru31=transorden12-transusuario;
ru41=transorden13-transusuario;
grado151=transorden31-transusuario; % realizo la resta de las señales
%procesadas (audio base de datos - audio grabado en tiempo real)
grado301=transorden32-transusuario;
grado451=transorden33-transusuario;
grado901=transorden34-transusuario;

error6(1)=mean(abs(grado151));  % mean = Devuelve los valores de
%media de los elementos
error6(2)=mean(abs(grado301));
error6(3)=mean(abs(grado451));
error6(4)=mean(abs(grado901));
error6(5)=mean(abs(si1));
error6(6)=mean(abs(si21));
error6(7)=mean(abs(si31));
error6(8)=mean(abs(si41));
error6(9)=mean(abs(si51));
error6(10)=mean(abs(ru1));
error6(11)=mean(abs(ru31));
error6(12)=mean(abs(ru41));
min_error6=min(error6);

if (min_error6 == error6(1))

    set (handles.parametro, 'string', '15º')
    x9=1;

elseif (min_error6 == error6(2))

    set (handles.parametro, 'string', '30º')
    x9=1;

elseif (min_error6 == error6(3))

    set (handles.parametro, 'string', '45º')
    x9=1;

elseif (min_error6 == error6(4))

    set (handles.parametro, 'string', '90º')
    x9=1;

elseif (min_error6 == error6(5))
```

```
        set (handles.parametro, 'string','1');
    elseif(min_error6 == error6(6))
        set (handles.parametro, 'string','2');
    elseif(min_error6 == error6(7))
        set (handles.parametro, 'string','3');
    elseif(min_error6 == error6(8))
        set (handles.parametro, 'string','4');
    elseif(min_error6 == error6(9))
        set (handles.parametro, 'string','5');
    elseif(min_error6 == error6(10))
        set (handles.parametro, 'string','ruido');
    elseif(min_error6 == error6(11))
        set (handles.parametro, 'string','ruido3');
    elseif(min_error6 == error6(12))
        set (handles.parametro, 'string','ruido4');
    end
    end
end
if(min_error == error(6))
    set (handles.texto, 'string','1');
end
if(min_error == error(7))
    set (handles.texto, 'string','2');
end
if(min_error == error(8))
    set (handles.texto, 'string','3');
end
if(min_error == error(9))
    set (handles.texto, 'string','4');
end
if(min_error == error(10))
    set (handles.texto, 'string','5');
end
if(min_error == error(11))
    set (handles.texto, 'string','ruido');
```

```
end

if(min_error == error(12))

    set (handles.texto, 'string','ruido3');

end

if(min_error == error(13))

    set (handles.texto, 'string','ruido4');

end


end%while
guidata(hObject, handles);
% hObject    handle to ordeneson (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function sonidoN=normalizar(sonido) % función para normalizar la señal
maximo=max(abs(sonido));  % saco el valor máximo y absoluto
n=length(sonido); %calcula el tamaño del vector
sonidoN=zeros(1,n); % zeros, me devuelve una matriz de ceros
for i=1:1:n
sonidoN(i)=sonido(i)/maximo;
end

% --- Executes on button press in ordenesoff.
function ordenesoff_Callback(hObject, eventdata, handles)
global on
on=0; %para que se desactive al pulsar ordenes off
% hObject    handle to ordenesoff (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function texto_Callback(hObject, eventdata, handles)
% hObject    handle to texto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of texto as text
%        str2double(get(hObject,'String')) returns contents of texto as a double


% --- Executes during object creation, after setting all properties.
function texto_CreateFcn(hObject, eventdata, handles)
% hObject    handle to texto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in salir.
function salir_Callback(hObject, eventdata, handles)
ans=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
```

```
if strcmp(ans,'No');
return;
end
clear,clc,close all
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function parametro_Callback(hObject, eventdata, handles)
% hObject    handle to parametro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of parametro as text
%        str2double(get(hObject,'String')) returns contents of parametro as a
double


% --- Executes during object creation, after setting all properties.
function parametro_CreateFcn(hObject, eventdata, handles)
% hObject    handle to parametro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function difuso_Callback(hObject, eventdata, handles)
% hObject    handle to difuso (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of difuso as text
%        str2double(get(hObject,'String')) returns contents of difuso as a double


% --- Executes during object creation, after setting all properties.
function difuso_CreateFcn(hObject, eventdata, handles)
% hObject    handle to difuso (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function unidad_Callback(hObject, eventdata, handles)
% hObject    handle to unidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of unidad as text
%        str2double(get(hObject,'String')) returns contents of unidad as a double
```

```
% --- Executes during object creation, after setting all properties.
function unidad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to unidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');

end
```

La interfaz creada para esta primera versión **(ilustración 120)** tiene tres graficas, donde se graficaba la señal de cada nivel a recocer. Un botón donde se activaba el reconocedor y otro botón que lo desactivaba.

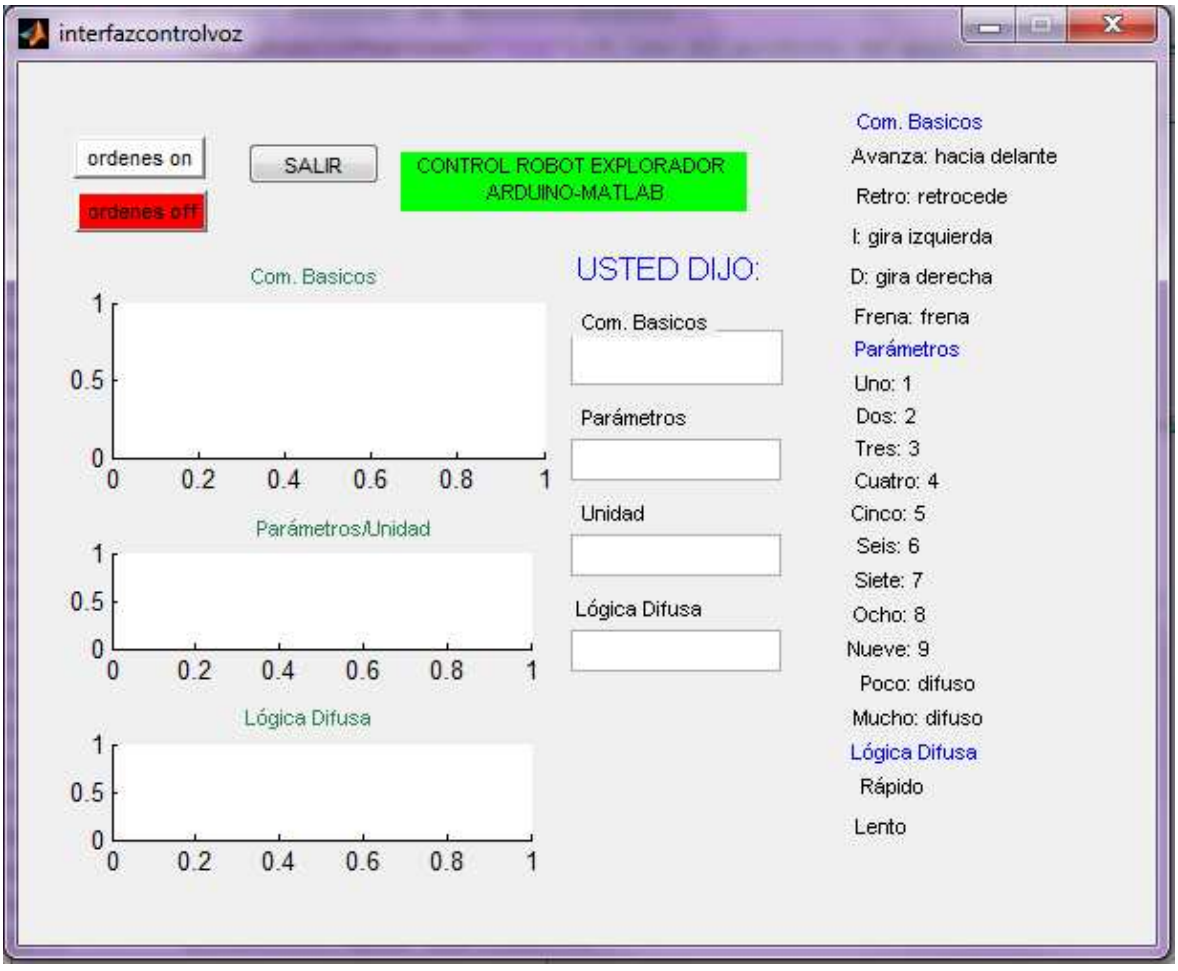



**Ilustración 120:** Interfaz primer reconocedor de voz.

Para la realización de la segunda versión del programa de reconocimiento de voz se hará siguiendo todo lo explicado en el punto **3.9 Reconocimiento de voz.** A continuación se irá explicando el procedimiento y elementos necesarios para la realización de este programa.

La frecuencia de muestreo elegida es de 44100Hz con una resolución de 16 bits y monocanal. Para mostrar en pantalla el audio a medida que se va adquiriendo con la función *wavrecord* no se puede hacer, ya que esta función gráfica cuando ha acabado la adquisición de audio, por lo tanto se recurre al código que se muestra más abajo y que se explicará brevemente. Este código permite hacer una adquisición de datos ya sea por la placa de sonido o por algún DSP conectado a la computadora de acuerdo a la configuración de instrucciones que se haga.

*hadles.AI=analoginput('winsound');* Esta línea creará un objeto de entrada analógica denominado handles.AI que manejará la placa de sonido.

*chan=addchannel(handles.AI, 1);* Se selecciona un solo canal.

*set(handles.AI,'SampleRate',44100);* Esta línea de codigo indica las muestras a tomar por segundo.

*set(handles.AI,'SamplesPerTrigger',duration*ActualRate);* Con este comando se indica la cantidad de muestras a tomar.

En el programa creado para el reconocimiento de voz se han colocado dos botones que realizan una adquisición de audio. Uno de ellos lleva incorporado las instrucciones anteriores y es el encargado de ingresar los patrones. El otro botón se utiliza para comenzar el reconocimiento de órdenes y se comienza por un umbral mínimo de audio en el micrófono. En este caso de reconocimiento de órdenes se deberá especificar las siguientes instrucciones para que inicie por presencia de audio en el micrófono.

*set(AI,'TriggerType','sofware');* Activación por presencia de audio.

*set(AI,'TriggerCondition','rising');* La activación de disparo es superado por un umbral.

*set(AI,'TriggerConditionValue',0.18');* El umbral es 0.18 voltios.

*set(AI,'TriggerChannel',AI.Channel(1));* Canal de disparo.

En cualquiera de los dos botones comienza la adquisión de audio con la instrucción *start(handles.AI)*. En el caso en el que se especifico el *trigger por umbral* los datos para Matlab solo interesarán a partir de que se supere el umbral y no desde que se ejecute *start.*

A continuación se verá el trozo de código que ingresa una determinada cantidad de muestras, plotearlas, continuar con la adquisión, plotear nuevamente y así hasta terminar con la adquisición de datos. El proceso al ser tan rápido parecerá que se grafica en tiempo real la señal. La función fundamental para esto es *drawnow.*

```
while handles.AI.SamplesAcquired<preview
end
while handles.AI.SamplesAcquired<duration*ActualRate
        data=peekdata(handles.AI,preview);
        for i=1:309
                if abs(data(i))<8e-3
                data(i)=0;
                end
        end
        set(handles.h,'ydata',data)
        drawnow
end
data=getdata(handles.AI);
handles.grafica=1:1:30900;
handles.h=plot(handles.grafica,data);
title('Orden');
xlabel('Muestras');
ylabel('Nivel de voz');
```

Una vez terminada la adquisición y la gráfica de la señal de audio se procede a la eliminación del objeto de entrada analógica que inicialmente se creó, con las siguientes funciones de MATLAB.

```
Delete(handles.AI);
clear handles.AI
```

### 4.9.1 Procesamiento de la señal.

Para el procesamiento de la señal se irán siguiendo los siguientes apartados e instrucciones.

#### *4.9.1.1 Filtrado de la señal de voz.*

Para el diseño del filtro para este caso, no se tenía la herramienta de MATLAB, por lo cual se utilizó un filtro ya creado para las mismas condiciones *[27]*.

El filtro escogido es un filtro FIR pasa-banda, tipo ventana Hamming con frecuencias de corte en 100Hz y 800Hz a una frecuencia de muestreo 44100Hz. El filtro que se obtuvo es un filtro de orden 800. Sobre el editor de MATLAB el filtro elegido se implementa a través de los siguientes comandos:

```
B=FIR1(800, [100 800]/22050;
patron=filter(B,1,ua4);
```

FIR1 devuelve los coeficientes del filtro. El tipo de ventana es la de Hamming que es la que se quiere para este caso. La función *filter* pasa la señal de voz (ua4) a través de los coeficientes que contiene B.

#### *4.9.1.2 Detección y eliminación de los períodos de silencio de la señal.*

Primero se hace un cálculo de la energía de la señal de audio calculando el valor absoluto de la señal y se obtiene el pico máximo. La señal de audio se divide por el valor antes obtenido para independizar la forma de onda respecto de la intensidad de la señal.

```
len=length(s);          longitud del vector.
d=max(abs(s));
s=s/d;
```

La señal normalizada se eleva al cuadrado y se divide por el número de muestras de la señal, para obtener la energía promedio de la señal.

```
avg_e=sum(s.*s)len;
```

Después se divide la señal una ver normalizada en ventanas de un número determinado de muestras, para posteriormente calcular la energía de ese trozo de señal y si la energía es mayor al umbral de decisión de la energía promedio de la señal completa entonces dicha ventana se conserva. Si la energía es menor la ventana se desecha ya que no llega su energía al umbral establecido, entonces esa ventana corresponde con un intervalo de silencio de la señal de audio.

```
THRES=0.02;
y=[0];
for i=1:400:len-400
        seg=s(i:i+399);
        e=sum(seg.*seg)/400;
        if(e>THRES*avg_e);
                y=[y;seg(1:end)];
        end
end
```

*THRES* es el umbral de decisión prefijado después de varias pruebas observando cuales eran los resultados de eliminación de silencios y corresponde al 2% de la energía promedio de la señal entera. Se eligieron 400 muestras para la definición de las ventanas que a la frecuencia de muestreo (44100Hz) corresponde a aproximadamente 10ms para evitar la posibilidad de que en el intento de eliminar silencio se elimine algún fonema de audio (cuya duración ronda entre los 10 y 20ms).

### 4.9.2 Parametrización de la señal de voz.

Para la obtener los coeficientes que caracterizan a los patrones guardados y/o a las órdenes de reconocimiento se utiliza la función *melcepst.m.* Esta función obtiene los coeficientes cepstrales en escala MEL. La elección de los coeficientes cepstrales en escala MEL se definió debido a que la mayoría de proyectos de reconocimiento de voz están realizados con este método.

La función *melcepst* tiene como argumentos los siguientes parámetros:

```
Melcepst(s,fs,w,nc,p,n,inc,f,fh);
```

Donde:

*s;* es la señal de audio con los intervalos de silencio eliminados.

*fs*; es la frecuencia de muestreo que en nuestro caso es de 44100Hz. Si se omite este parámetro, el algoritmo toma por defecto 11025Hz.

*w*; es el tipo de filtro que conforma el banco de filtros. Puede ser de Hamming, Hanning, rectangular, triangular, entre otros. Para este caso se utiliza la ventana Hamming.

*nc*; es el numero de coeficientes cepstrales que caracterizarán a la parametrización. Por defecto son 12 pero en nuestro caso se utilizan 14 coeficientes ya que se obtuvieron mejores resultados.

*p;* indica el número de filtros que tendrá el banco de filtros característicos de este tipo de parametrización. Para este caso se ha definido 30 filtros para el banco.

*n*; longitud de las ventanas de muestras. La función *melcepst* además de la parametrización hace el ventaneo por lo que es un parámetro a especificar. La función impone que sea una potencia de 2 entonces se eligió n=1024 que corresponde a una ventana de aproximadamente 23ms de duración, valor que se encuentra dentro del rango (20 a 40ms) donde la señal se considera estacionaria.

*inc;* indica el solapamiento entre ventanas. El por qué del solapamiento se explicó en la parte teórica del ventaneo. El valor por defecto es n/2 que según nuestra elección de *n* sería 512 muestras pero se eligió inc=410 por mejores resultados.

Cuando se ejecuta *melcepst* con los argumentos deseados, esta función devuelve una matriz con los coeficientes que caracterizan a la señal de audio ingresada.

Otra consideración a tener en cuenta es que algunas órdenes a reconocer empezaban por la misma palabra, como es más deprisa y más despacio o gira izquierda y gira derecha, por lo tanto se ha optado por obtener dos matrices por cada patrón u orden de reconocimiento ingresada dividiendo la señal de audio de entrada en 2 partes. Estas coincidencias entorpecían la identificación y la solución fue la división de la señal en 2 partes. La división de la señal favorece la identificación pero como la duración de las diferentes

palabras no la misma se realizó un solapamiento en la división de la señal que eliminará el inconveniente. Este solapamiento es tomar como primera parte de la orden desde la muestra 1 a la última menos 4000, y como segunda parte de la orden, se toma de la última menos 8000. Es decir hay un solapamiento de 4000 muestras que soluciona lo antes mencionado. Traducido a código queda como sigue:

```
c01=melcepst(q(1:end-4000),fs,'M',14,30,1024,410,0,0.1815);
c02=melcepst(q(end-8000:end),fs,'M',14,30,1024,410,0,0.1815);
```

Este código da como resultado dos vectores.

### 4.9.3 Reconocimiento del habla. Decisión.

Una vez que se tiene la parametrización de la señal se pasa al siguiente nivel que será el reconocimiento del habla, es decir, que una señal hablada sea comparada con los patrones ingresados y que este proceso reconozca si coincide o no con algún patrón.

Para la realización de este nivel se realizan los siguientes procedimientos descritos a continuación.

#### *4.9.3.1 Distancia de las matrices de parametrización.*

Para comparar las matrices se debe definir una medida de distancia entre los vectores característicos. En el algoritmo de reconocimiento en MATLAB se utiliza una distancia Euclidea, definida del siguiente modo: por ejemplo si *fi y fi'*, con *i=0, 1, 2, …, D* son las componentes de dos vectores característicos *f y f'*, pueden definirse la siguiente métrica.

$$d = \sqrt{\sum_{i=1}^{D} |f_i - f'_i|^2}$$

Para la implementación de esta fórmula se creó la función *disteu* que a su vez utiliza la función *disteusq* de la librería Voicebox. La función *disteusq* devuelve un vector distancia que la función *disteu* transforma en un único valor. Como hay dos matrices; una de ellas corresponde al patrón y otra de la orden a reconocer, esta función se ejecuta tantas veces como patrones haya. Por lo tanto cuanto menor número de patrones a reconocer más rápido

será el programa. Ya que la comparación es uno a uno. Como resultado se obtiene un vector que contiene la distancia de la matriz característica de la orden respecto a cada patrón guardado. El siguiente vector es un resultado de un patrón ingresado.

*dt =*
*Columns 1 through 14*
*7.9101 11.3892 6.0783 6.9842 8.9279 10.7768 14.5465 12.2339 16.3177 15.3169 17.3371 17.2823 10.5972 11.8169*
*Columns 15 through 28*
*13.4350 13.8587 12.0490 12.0799 6.8557 9.2736 6.4153 12.9387 8.8501 8.6826 11.5952 10.2881 23.5016 11.7178*

El vector *dt* contiene todas las distancias y como se puede observar la menor de ellas es la que se encuentra en el tercer elemento del vector. Una vez obtenida la menor se procede mediante código a obtener el menor elemento que indicará la menor distancia y si este es menor a un valor prefijado por el programador o persona que vaya a utilizar el reconocedor se determina que esa es el comando identificado, que indica la menor distancia. Esto es:

| Elemento | Orden |
|---|---|
| 1 | Para |
| 2 | Más Despacio |
| 3 | Más Deprisa |
| 4 | Gira Izquierda |
| 5 | Gira Derecha |
| 6 | Retrocede |
| 7 | Avanza |

**Tabla 1:** Órdenes.

Por ejemplo para este caso en el que la menor distancia se encuentra en la posición 3 corresponderá a la orden Más Deprisa.

### 4.9.4 Interfaz Gráfica.

Para la creación de la interfaz gráfica del reconocedor de voz se recurrió al uso de la GUIDE de MATLAB *[2, 11]*. GUIDE nos permite diseñar primero gráficamente la interfaz de usuario para luego cargar dicha interfaz a código. Se pueden ir colocando sobre un marco diferentes tipos de botones, textos fijos, textos dinámicos modificables por el usuario, checkbox, menús desplegables, imágenes, gráficos, entre otros.

A continuación se muestra la interfaz gráfica que se ha creado para el programa de reconocimiento de voz para este caso **(ilustración 121)**.

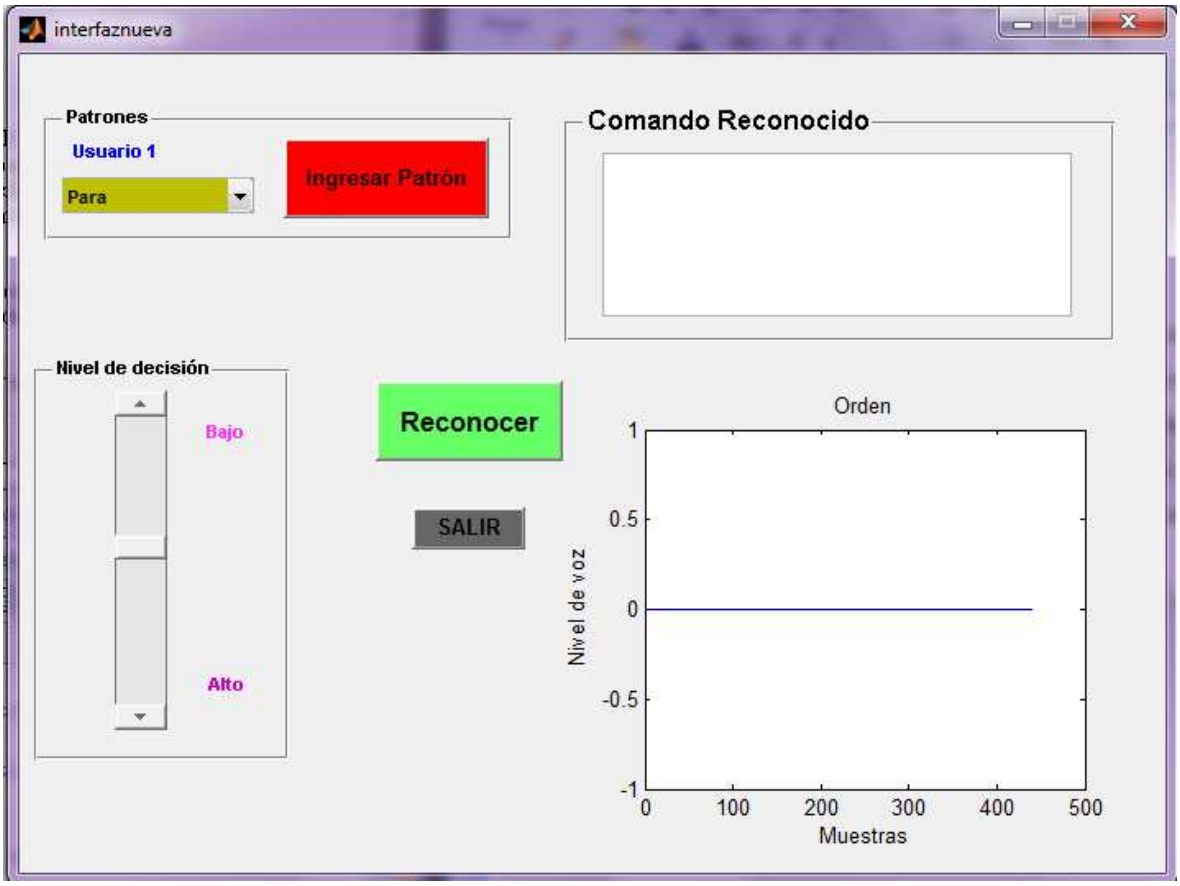



**Ilustración 121:** interfaz gráfica programa reconocedor de voz.

En la interfaz creada aparece una serie de botones y elementos que se explicarán a continuación:

**Patrones:** en la sección patrones es donde se eligen los patrones que se desean ingresar. Hay un menú desplegable con la lista de patrones **(ilustración 125)** que se quieran guardar. Para ingresar un patrón primero hay que pulsa el botón *ingresar patrón* y a continuación se debe decir el patrón deseado, cuando se haya terminado la grabación aparecerá un mensaje ***Patrón Ingresado*** **(ilustración 122).** Una vez dicho el patrón hay que guardarlo en la lista de patrones haciendo clic en el nombre en el que se desea guardar el patrón, una vez hecho el clic aparecerá un mensaje diciendo ***Patrón Guardado*** **(ilustración 123).** En el caso de que se presione los menús pop up y no se haya ingresado con anterioridad un patrón de audio nuevo se mostrará un mensaje de error diciendo ***Primero ingrese el Patrón*** **(ilustración 124).**


**Ilustración 122:** Mensaje Patrón ingresado.


**Ilustración 123:** Mensaje Patrón Guardado.


**Ilustración 124:** Mensaje Primero Ingrese el Patrón.

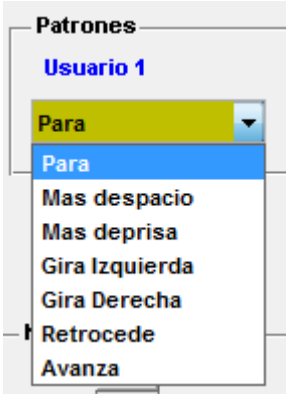



**Ilustración 125:** Lista de patrones.

Como anteriormente se dividió la señal en dos partes, y se definió que una de las muestras fuera de (end-8000) se deberá tener un patron como minimo de 10000 muestras, de lo contrario aparecera un mensaje de aviso diciendo ***Datos insuficientes*** **(ilustración 126).**


**Ilustración 126:** Mensaje datos insuficientes.

Cuando se pulsa el botón ***Salir*** de la interfaz se muestra el siguiente mensaje **(ilustración 127)**:


**Ilustración 127:** Mensaje de opción salir.

Cuando se activa el botón ***Reconocer*** de la interfaz del programa **(ilustración 121)** el botón cambia de nombre a ***Escape.*** Una vez que el botón se ha puesto en *escape* el programa está listo para reconocer, que se iniciará cuando el nivel de voz supere el umbral establecido anteriormente explicado. Para dejar de reconocer pulsar sobre ***Escape.***

En el cuadro donde pone "comando reconocido" aparecerá el comando que haya reconocido el programa. Si la orden dicha no coincide con los patrones guardados aparecerá "***Patrón Desconocido***"**.**

En la gráfica de la parte inferior derecha de la interfaz **(ilustración 121)** se grafica la señal de audio **(ilustración 128)**.

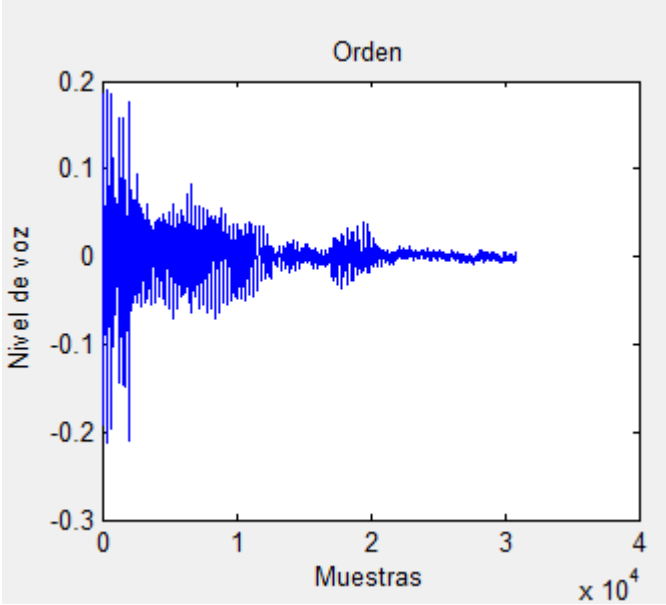



**Ilustración 128:** Gráfica señal de audio.

La barra deslizadora con el nombre ***nivel de decisión*** **(ilustración 129)** se utiliza para comparar la distancia calculada con un nivel que nosotros prefijemos. Con esto se consigue mayor precisión a la hora de reconocer las palabras exactas.

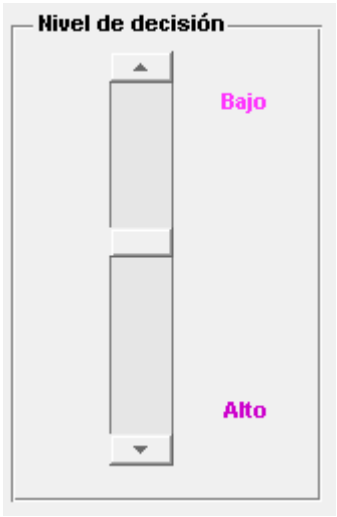



**Ilustración 129:** Barra nivel de decisión.

Para que el programa de reconocimiento de voz funcione correctamente se deberá descargar la librería ***Voicebox*** de MATLAB *[27]*, ya que las funciones creadas hacen uso a su vez de otras funciones de esta librería. Una vez descargada la librería se debe descomprimir en la misma carpeta donde esté guardado el programa de reconocimiento de voz y sus funciones. A continuación se pondrán los códigos de los programas para la creación el reconocimiento de voz.

**Programa *Interfaznueva.m*:** Esta librería es la principal, la que hace la llamada al resto de funciones de procesamiento de señal, parametrización, lógica difusa y comunicación con Arduino. También es la encargada de crear la interfaz gráfica.

```
function varargout = interfaznueva(varargin)
% INTERFAZNUEVA M-file for interfaznueva.fig
%      INTERFAZNUEVA, by itself, creates a new INTERFAZNUEVA or raises
%      the existing singleton*.
%
%      H = INTERFAZNUEVA returns the handle to a new INTERFAZNUEVA or the
%      handle to the existing singleton*.
%
%      INTERFAZNUEVA('CALLBACK',hObject,eventData,handles,...) calls the
%      local function named CALLBACK in INTERFAZNUEVA.M with the given
%      input arguments.
%
%      INTERFAZNUEVA('Property','Value',...) creates a new INTERFAZNUEVA
%      or raises the existing singleton*.
%
%      Starting from the left, property value pairs are applied to the
%      GUI before interfaznueva_OpeningFcn gets called.
%
%      An unrecognized property name or invalid value makes property
%      application stop.
%      All inputs are passed to interfaznueva_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
%      one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help interfaznueva

% Last Modified by GUIDE v2.5 31-Jan-2014 01:07:38

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interfaznueva_OpeningFcn, ...
                   'gui_OutputFcn',  @interfaznueva_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before interfaznueva is made visible.
function interfaznueva_OpeningFcn(hObject, eventdata, handles, varargin)
% Asignación del objeto axes para la imagen adjunta.
%axes(handles.imagen)
%[x,map]=imread('voip-icomuni.jpg','jpg');
%image(x),colormap(map),axis off,hold on
axes(handles.grafica);
handles.grafica=1:1:441;
handles.h = plot(handles.grafica,zeros(441,1));
```

```
title('Orden')
xlabel('Muestras')
ylabel('Nivel de voz')
%Carga_Patron;
handles.estruc_U1=load('para.mat');
handles.estruc_U2=load('masdespacio.mat');
handles.estruc_U3=load('masdeprisa.mat');
handles.estruc_U4=load('giraizquierda.mat');
handles.estruc_U5=load('giraderecha.mat');
handles.estruc_U6=load('retrocede.mat');
handles.estruc_U7=load('avanza.mat');
handles.ctrl1=0;
handles.indice=0;
handles.sliderboton=load('slider2.mat');
warning('off','MATLAB:dispatcher:InexactCaseMatch');

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interfaznueva (see VARARGIN)

% Choose default command line output for interfaznueva
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes interfaznueva wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = interfaznueva_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
handles.slider2=get(hObject,'Value'); %Carga en handles.slider2 el valor %
delSlider
handles.slider2=handles.slider2.*15+2;
sld2=handles.slider2;
save('slider2.mat','sld2');
handles.sliderboton=load ('slider2.mat');
guidata(hObject, handles);
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
handles.sliderboton=load ('slider2.mat');
val=(handles.sliderboton.sld2-2)/15;
```

```
set(hObject,'Value',val);
usewhitebg = 1;
if usewhitebg
set(hObject,'BackgroundColor',[.9 .9 .9]);
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
handles.ctrl1=0;
P1=get(handles.popupmenu1,'Value');
if (handles.ctrl1~=handles.indice)
handles.ctrl1=handles.indice;
ctrl1=handles.indice;
c1 = handles.patron_grabado1;
c2 = handles.patron_grabado2;
U = handles.patron_senial;
switch P1
    case 1,
        save('para.mat','c1','c2','U');
        handles.estruc_U1=load ('para.mat');
    case 2,
        save('masdespacio.mat','c1','c2','U');
        handles.estruc_U2=load ('masdespacio.mat');
    case 3,
        save('masdeprisa.mat','c1','c2','U');
        handles.estruc_U3=load ('masdeprisa.mat');
    case 4,
        save('giraizquierda.mat','c1','c2','U');
        handles.estruc_U4=load ('giraizquierda.mat');
    case 5,
        save('giraderecha.mat','c1','c2','U');
        handles.estruc_U5=load ('giraderecha.mat');
    case 6,
        save('retrocede.mat','c1','c2','U');
        handles.estruc_U6=load ('retrocede.mat');
    case 7,
        save('avanza.mat','c1','c2','U');
        handles.estruc_U7=load ('avanza.mat');
        end
msgbox('Patron Guardado');
else
warndlg('Primero ingrese el Patron' ,'AVISO');
end
guidata(hObject, handles);
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 %contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from %popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns %called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


% --- Executes on button press in Patron_nuevo.
function Patron_nuevo_Callback(hObject, eventdata, handles)
set(handles.Patron_nuevo,'enable','inactive');
set(handles.Salir,'enable','inactive');
set(handles.popupmenu1,'enable','inactive');
[data2]=ingresoaudio;
wavwrite(data2,44100,16,'patron.wav');
ua4=wavread('patron.wav');
set(handles.Patron_nuevo,'enable','on');
set(handles.Salir,'enable','on');
set(handles.popupmenu1,'enable','on');
B = FIR1(800,[100 8000]/22050);
patron=filter(B,1,ua4);
Aa=0;
[cp1,cp2,Aa]=reconocedor(patron);
Aa;
if Aa(1)>10000
handles.patron_grabado1 = cp1;
handles.patron_grabado2 = cp2;
handles.patron_senial = ua4;
handles.indice = sum(patron)/30900;
msgbox('Patron ingresado');
guidata(hObject, handles);
else
warndlg({'Datos insuficientes.',' Ingresar comando de nuevo'} ,'AVISO');
end
guidata(hObject, handles);
% hObject    handle to Patron_nuevo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in Salir.
function Salir_Callback(hObject, eventdata, handles)
ans=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(ans,'No')
guidata(hObject, handles);
return;
end
clear,clc,close all
% hObject    handle to Salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in togglebutton4.
function togglebutton4_Callback(hObject, eventdata, handles)
handles.toogle=get(hObject,'Value');
toogle=handles.toogle;
set(handles.Patron_nuevo,'enable','inactive');
set(handles.Salir,'enable','inactive');
set(handles.popupmenu1,'enable','inactive');
set(handles.togglebutton4,'string','Reconocer');
Comp(1)=handles.estruc_U1;
Comp(2)=handles.estruc_U2;
Comp(3)=handles.estruc_U3;
Comp(4)=handles.estruc_U4;
Comp(5)=handles.estruc_U5;
Comp(6)=handles.estruc_U6;
Comp(7)=handles.estruc_U7;
for ff=1:1:1000
handles.grafica=1:1:309; %441 antes 882
handles.h = plot(handles.grafica,zeros(309,1));
```

```
h=handles.h;
graf=handles.grafica;
data=vozgrabar(graf,h,hObject,handles);
[m,n]=size(data);
if [m,n]~=[30900,1]
return
elseif isempty(data)==1
return
else
wavwrite(data,44100,16,'test.wav');
ua2=wavread('test.wav');
B = FIR1(800,[100 8000]/22050);
test=filter(B,1,ua2);
[c01,c02,Ab]=reconocedor(test);
handles.toogle=get(hObject,'Value');
toogle=handles.toogle;
if Ab(1)>10000 && toogle==1
for i=1:7
d1(i)=disteu(Comp(i).c1,c01);
d2(i)=disteu(Comp(i).c2,c02);
dt(i)=(d1(i)+d2(i))/2;
end
[DD,Ind]=min(dt);
AA=handles.sliderboton.sld2;
if DD<AA
     %Se crea la comunicación serial en COM7
        arduino = serial('COM7','BaudRate',9600,'Terminator','CR/LF');
        warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
        %Se abre el puerto poder enviar los datos
        fopen(arduino);
        fprintf(arduino,'%s',char(Ind)); %Se envía el DATO
        fclose(arduino); %Se cierra el puerto
    if (Ind==1)
        set (handles.ComandoReconocido, 'string', 'Para')
    elseif (Ind==2)
        set (handles.ComandoReconocido, 'string', 'Más Despacio')
        fopen(arduino);
            fprintf(arduino,'%s',char(2));
            Nvalores=2; %Cantidad de valores que queremos a leer
            m1=zeros(1,Nvalores);
            i=1;
            k=0;
            fprintf(arduino,'%s',char(8));
            while k<Nvalores
             %Leer el puerto serie
             a = fscanf(arduino,'%f.%f');
                   m1(i)=a(1);
             %Incrementar el contador
             i=i+1;
             k=k+1;
            end
            fismat=readfis('logdifu');
            incremento=evalfis([m1(2), m1(1), 1],fismat);
            fprintf(arduino,'%s',char(9));
            fprintf(arduino,'%s',char(incremento));
            fclose(arduino);
    elseif (Ind==3)
        set (handles.ComandoReconocido, 'string', 'Más Deprisa')
            fopen(arduino);
            fprintf(arduino,'%s',char(3));
            Nvalores=2; %Cantidad de valores que queremos a leer
            m1=zeros(1,Nvalores);
            i=1;
            k=0;
            fprintf(arduino,'%s',char(8));
            while k<Nvalores
             %Leer el puerto serie
```

```
            a = fscanf(arduino,'%f.%f');
            m1(i)=a(1);
            %Incrementar el contador
            i=i+1;
            k=k+1;
           end
           fismat=readfis('logdifu');
           incremento=evalfis([m1(2), m1(1), 2],fismat);
           fprintf(arduino,'%s',char(9));
           fprintf(arduino,'%s',char(incremento));
           fclose(arduino);
    elseif (Ind==4)
        set (handles.ComandoReconocido, 'string', 'Gira Izquierda')
    elseif (Ind==5)
        set (handles.ComandoReconocido, 'string', 'Gira Derecha')
    elseif (Ind==6)
        set (handles.ComandoReconocido, 'string', 'Retrocede')
    elseif (Ind==7)
        set (handles.ComandoReconocido, 'string', 'Avanza')
    end
else
      set (handles.ComandoReconocido, 'string', 'Patron Desconocido')
end
elseif toogle==1
      set (handles.ComandoReconocido, 'string', 'Datos Insuficientes')
else
      set(handles.togglebutton4,'string','Reconocer');
return
end
set(handles.togglebutton4,'string','Reconocer');
pause(0.25)
set(handles.togglebutton4,'string','Reconocer');
clear data ua2 test Ab c01 c02 DD d1 d2 dt Ind;
end
end
guidata(hObject, handles);

% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton4



function ComandoReconocido_Callback(hObject, eventdata, handles)
% hObject    handle to ComandoReconocido (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ComandoReconocido as %text
%        str2double(get(hObject,'String')) returns contents of %ComandoReconocido
as a double


% --- Executes during object creation, after setting all properties.
function ComandoReconocido_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ComandoReconocido (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns %called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

**Función *ingresoaudio.m*:** Con esta función se ingresa los nuevos patrones. Con un tiempo de grabación aproximado de 0.7 segundos con 30900 muestras.

```
% FUNCIÓN INGRESO AUDIO
function [data]=ingresoaudio
handles.AI = analoginput('winsound');
chan = addchannel(handles.AI,1);
duration = 0.7006802721088435; % Tiempo de grabación.
set(handles.AI,'SampleRate',44100)
ActualRate = get(handles.AI,'SampleRate');
set(handles.AI,'SamplesPerTrigger',duration*ActualRate)
preview = duration*ActualRate/100;
set(gcf,'doublebuffer','on')
handles.grafica=1:1:309;
handles.h = plot(handles.grafica,zeros(309,1));
title('Orden')
xlabel('Muestras')
ylabel('Nivel de voz')
data =zeros(30900,1);
start(handles.AI)
while handles.AI.SamplesAcquired < preview
end
while handles.AI.SamplesAcquired < duration*ActualRate
       data = peekdata(handles.AI,preview);
       for i=1:309
              if abs(data(i))<8e-3
              data(i) =0;
              end
       end
       set(handles.h,'ydata',data)
       drawnow
end
data = getdata(handles.AI);
handles.grafica=1:1:30900;
handles.h = plot(handles.grafica,data);
title('Orden')
xlabel('Muestras')
ylabel('Nivel de voz')
delete(handles.AI)


clear handles.AI
```

**Función *vozgrabar.m*:** Con esta función se ingresa las frases a ser reconocidas. Hay un umbral de voz que inicia la adquisición del audio haciéndolo muy práctico ya que con pequeños ruidos no se inicia, evitando los problemas de la primera versión de reconocimiento de voz.

```
% FUNCIÓN VOZGRABAR
function [data]=vozgrabar(graf,h,hObject,handles)
AI = analoginput('winsound');
chan = addchannel(AI,1);
duration = 0.7006802721088435;
set(AI,'SampleRate',44100);
ActualRate = get(AI,'SampleRate');
set(AI,'SamplesPerTrigger',duration*ActualRate);
set(AI,'TriggerType','software');
set(AI,'TriggerCondition','rising');
set(AI,'TriggerConditionValue',0.18); %0.18 umbral voltios inicio
```

```
set(AI,'TriggerChannel',AI.Channel(1));
set(AI,'TimeOut',600);
preview = duration*ActualRate/100;
set(gcf,'doublebuffer','on');
data =[];
%set(handles.togglebutton4,'string','Reconocer');
start(AI)
%guidata(hObject, handles);
handles.toogle=get(hObject,'Value');
if handles.toogle==0
       set(handles.Patron_nuevo,'enable','on');
       %set(handles.slider2,'enable','on');
       set(handles.Salir,'enable','on');
       set(handles.popupmenu1,'enable','on');
       %set(handles.togglebutton4,'string','Escape');
       stop(AI)
       delete(AI)
       clear AI
       pause(0.25)
       return
else
       while AI.SamplesAcquired < preview
       end
       while AI.SamplesAcquired < duration*ActualRate
              guidata(hObject, handles);
              handles.toogle=get(hObject,'Value');
              if handles.toogle==0
              stop(AI)
              delete(AI)
              clear AI
              pause(0.25)
              return
              else
              data = peekdata(AI,preview);
              for i=1:1:309
                     if abs(data(i))<8e-3
                     data(i) =0;
                     end
              end
              set(h,'ydata',data);
              drawnow
              set(handles.togglebutton4,'string','Escape');
              end
       end
       data = getdata(AI);
       graf=1:1:30900;
       h = plot(graf,data);
       title('Orden')
       xlabel('Muestras')
       ylabel('Nivel de voz')
end
delete(AI)
clear AI


pause(0.25)
```

**Función *reconocedor.m*:** Esta función encadena a diferentes funciones para realizar el estudio del audio ingresado ya sea como patrón o como orden a reconocer.

```
% FUNCIÓN RECONOCEDOR
function [c01,c02,A]=reconocedor(y)
fs=44100;
q=silencio(y);
```

```
A=size(q);
if A(1)>10000
       c01=melcepst(q(1:end-4000),fs,'M',14,30,1024,410,0,0.1815);
       c02=melcepst(q(end-8000:end),fs,'M',14,30,1024,410,0,0.1815);
else
       c01=zeros(31,14);
       c02=zeros(61,14);

end
```

**Función *silencio.m*:** Con esta función se consigue la eliminación de los períodos de silencio de la señal ingresada como patrón o a identificar.

```
% silencio
%Corta el silencio en la señal completa
function y = silencio(s)
len = length(s);% length del vector
d=max(abs(s));
s=s/d;
avg_e = sum(s.*s)/len; %promedio señal entera
THRES = 0.02;
y = [0];
for i = 1:400:len-400 % cada 10ms
       seg = s(i:i+399);% segmentos
       e = sum(seg.*seg)/400; % promedio de cada segmento
       if( e> THRES*avg_e) % si el promedio energético es mayor que la
       %señal completa por el valor umbral
              y=[y;seg(1:end)];% almacena en y si no es eliminado como
       espacio en blanco
       end;

end;
```

**Función *melcepst.m*:** Con esta función se obtiene los coeficientes cepstrales en escala de MEL.

```
function c=melcepst(s,fs,w,nc,p,n,inc,fl,fh)
%MELCEPST Calculate the mel cepstrum of a signal %C=(S,FS,W,NC,P,N,INC,FL,FH)
%
%
% Simple use: c=melcepst(s,fs) % calculate mel cepstrum with 12 coefs,
% 256 sample frames
% c=melcepst(s,fs,'e0dD') % include log energy, 0th cepstral coef, delta % and
delta-delta coefs
%
% Inputs:
% s speech signal
% fs sample rate in Hz (default 11025)
% nc number of cepstral coefficients excluding 0'th coefficient (default % 12)
% n length of frame in samples (default power of 2 < (0.03*fs))
% p number of filters in filterbank (default: floor(3*log(fs)) = approx
% 2.1 per ocatave)
% inc frame increment (default n/2)
% fl low end of the lowest filter as a fraction of fs (default = 0)
% fh high end of highest filter as a fraction of fs (default = 0.5)
%
% w any sensible combination of the following:
%
```

```
% 'R' rectangular window in time domain
% 'N' Hanning window in time domain
% 'M' Hamming window in time domain (default)
%
% 't' triangular shaped filters in mel domain (default)
% 'n' hanning shaped filters in mel domain
% 'm' hamming shaped filters in mel domain
%
% 'p' filters act in the power domain
% 'a' filters act in the absolute magnitude domain (default)
%
% '0' include 0'th order cepstral coefficient
% 'e' include log energy
% 'd' include delta coefficients (dc/dt)
% 'D' include delta-delta coefficients (d^2c/dt^2)
%
% 'z' highest and lowest filters taper down to zero (default)
% 'y' lowest filter remains at 1 down to 0 frequency and
% highest filter remains at 1 up to nyquist freqency
%
% If 'ty' or 'ny' is specified, the total power in the fft is preserved.
%
% Outputs: c mel cepstrum output: one frame per row. Log energy, if
& requested, is the
% first element of each row followed by the delta and then the delta
% delta
% coefficients.
%
% BUGS: (1) should have power limit as 1e-16 rather than 1e-6 (or
% possibly a better way of choosing this)
% and put into VOICEBOX
% (2) get rdct to change the data length (properly) instead of doing it
% explicitly (wrongly)
% Copyright (C) Mike Brookes 1997
% Version: $Id: melcepst.m,v 1.7 2009/10/19 10:20:32 dmb Exp $
%
% VOICEBOX is a MATLAB toolbox for speech processing.
% Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 2 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You can obtain a copy of the GNU General Public License from
% http://www.gnu.org/copyleft/gpl.html or by writing to
% Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin<2
      fs=11025;
end
if nargin<3
      w='M';
end
if nargin<4
      nc=12;
end
if nargin<5
      p=floor(3*log(fs));
```

```
end
if nargin<6
      n=pow2(floor(log2(0.03*fs)));
end
if nargin<9
      fh=0.5;
      if nargin<8
            fl=0;
            if nargin<7
                  inc=floor(n/2);
            end
      end
end
if length(w)==0
      w='M';
end
if any(w=='R')
      z=enframe(s,n,inc);
elseif any (w=='N')
      z=enframe(s,hanning(n),inc);
else
      z=enframe(s,hamming(n),inc);
end
f=rfft(z.');
[m,a,b]=melbankm(p,n,fs,fl,fh,w);
pw=f(a:b,:).*conj(f(a:b,:));
pth=max(pw(:))*1E-20;
if any(w=='p')
      y=log(max(m*pw,pth));
else
      ath=sqrt(pth);
      y=log(max(m*abs(f(a:b,:)),ath));
end
c=rdct(y).';
nf=size(c,1);
nc=nc+1;
if p>nc
      c(:,nc+1:end)=[];
elseif p<nc
      c=[c zeros(nf,nc-p)];
end
if ~any(w=='0')
      c(:,1)=[];
      nc=nc-1;
end
if any(w=='e')
      c=[log(sum(pw)).' c];
      nc=nc+1;
end
% calculate derivative
if any(w=='D')
      vf=(4:-1:-4)/60;
      af=(1:-1:-1)/2;
      ww=ones(5,1);
      cx=[c(ww,:); c; c(nf*ww,:)];
      vx=reshape(filter(vf,1,cx(:)),nf+10,nc);
      vx(1:8,:)=[];
      ax=reshape(filter(af,1,vx(:)),nf+2,nc);
      ax(1:2,:)=[];
      vx([1 nf+2],:)=[];
      if any(w=='d')
            c=[c vx ax];
      else
            c=[c ax];
      end
elseif any(w=='d')
      vf=(4:-1:-4)/60;
```

```
        ww=ones(4,1);
        cx=[c(ww,:); c; c(nf*ww,:)];
        vx=reshape(filter(vf,1,cx(:)),nf+8,nc);
        vx(1:8,:)=[];
        c=[c vx];
end
if nargout<1
        [nf,nc]=size(c);
        t=((0:nf-1)*inc+(n-1)/2)/fs;
        ci=(1:nc)-any(w=='0')-any(w=='e');
        imh = imagesc(t,ci,c.');
        axis('xy');
        xlabel('Time (s)');
        ylabel('Mel-cepstrum coefficient');
        map = (0:63)'/63;
        colormap([map map map]);
        colorbar;
end
```

**Función *disteu.m:*** Obtiene la distancia entre la matriz característica de un patrón determinado y la matriz característica de la orden a identificar. El valor se obtiene del vector distancia que le pasa la función *disteusq.*

```
function dist=disteu(f00,f01)
d=disteusq(f00,f01,'x');
dist=sum(min(d,[],2))/size(d,1);
```

**Función *disteusq.m*:** Determina el vector distancia entre la matriz característica de la orden a identificar y la matriz característica de un patrón determinado. Este vector se pasa a la función *disteu.*

```
function d=disteusq(x,y,mode,w)
%DISTEUSQ calculate euclidean, squared euclidean or mahanalobis distance %
D=(X,Y,MODE,W)
%
% Inputs: X,Y Vector sets to be compared. Each row contains a data
% vector.
% X and Y must have the same number of columns.
%
% MODE Character string selecting the following options:
% 'x' Calculate the full distance matrix from every row of X to every row
% of Y
% 'd' Calculate only the distance between corresponding rows of X and Y
% The default is 'd' if X and Y have the same number of rows otherwise
% 'x'.
% 's' take the square-root of the result to give the euclidean distance.
%
% W Optional weighting matrix: the distance calculated is (x-y)*W*(x-y)'
% If W is a vector, then the matrix diag(W) is used.
%
% Output: D If MODE='d' then D is a column vector with the same number of
% rows as the shorter of X and Y.
% If MODE='x' then D is a matrix with the same number of rows as X and
% the same number of columns as Y'.
%
% Copyright (C) Mike Brookes 1998
% Version: $Id: disteusq.m,v 1.6 2010/04/19 16:56:22 dmb Exp $
%
```

```
% VOICEBOX is a MATLAB toolbox for speech processing.
% Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 2 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% GNU General Public License for more details.
%
% You can obtain a copy of the GNU General Public License from
% http://www.gnu.org/copyleft/gpl.html or by writing to
% Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[nx,p]=size(x);
ny=size(y,1);
if nargin<3 | isempty(mode)
    mode='0';
end
if any(mode=='d') | (mode~='x' & nx==ny)
      % Do pairwise distance calculation
      nx=min(nx,ny);
      z=double(x(1:nx,:))-double(y(1:nx,:));
      if nargin<4
            d=sum(z.*conj(z),2);
      elseif min(size(w))==1
            wv=w(:).';
            d=sum(z.*wv(ones(size(z,1),1),:).*conj(z),2);
      else
            d=sum(z*w.*conj(z),2);
      end
else
      % Calculate full distance matrix
      if p>1
      % x and y are matrices
            if nargin<4
                  z=permute(double(x(:,:,ones(1,ny))),[1 3 2])-
            permute(double(y(:,:,ones(1,nx))),[3 1 2]);
                  d=sum(z.*conj(z),3);
            else
                  nxy=nx*ny;
                  z=reshape(permute(double(x(:,:,ones(1,ny))),[1 3 2])-
                  permute(double(y(:,:,ones(1,nx))),[3 1 2]),nxy,p);
                  if min(size(w))==1
                  wv=w(:).';
                  d=reshape(sum(z.*wv(ones(nxy,1),:).*conj(z),2),nx,ny);
                  else
                  d=reshape(sum(z*w.*conj(z),2),nx,ny);
                  end
            end
      else
      % x and y are vectors
            z=double(x(:,ones(1,ny)))-double(y(:,ones(1,nx))).';
            if nargin<4
                  d=z.*conj(z);
            else
                  d=w*z.*conj(z);
            end
      end
end
if any(mode=='s')
      d=sqrt(d);
```

```
end
```

En la elaboración por completo de la segunda versión del reconocedor de voz aparecieron problemas de reconocimiento, ya que no reconocía bien las órdenes. Estos problemas se fueron resolviendo cambiando parámetros como son el tiempo de grabación, cantidad de muestras, frecuencia de muestreo, etc. Llegando a esta última versión donde se obtiene un resultado satisfactorio. En esta versión no se incorporaron órdenes de distancias concretas para hacer un reconocedor mucho más rápido y robusto.

## 4.10 Programa control difuso (Toolbox fuzzy).

Primeramente se deberá diseñar el sistema difuso que cumpla con las especificaciones para el que es diseñado. El diseño de un sistema difuso para este robot tiene como principal objetivo controlar la aceleración y el frenado del vehículo, de forma suavizada, en base a los objetos que rodean al coche.

Un sistema difuso necesita unas variables de entrada. Cada variable de entrada se define mediante un dominio de definición: conjuntos difusos que abarca el rango de valores que podría tomar esa variable.

Nuestro sistema tendrá cuatro dominios de definición (tres variables de entrada y una de salida), en las que cada dominio estará comprendido por diferentes conjuntos difusos. Estos conjuntos están asociados con etiquetas lingüísticas para una mejor interpretación.

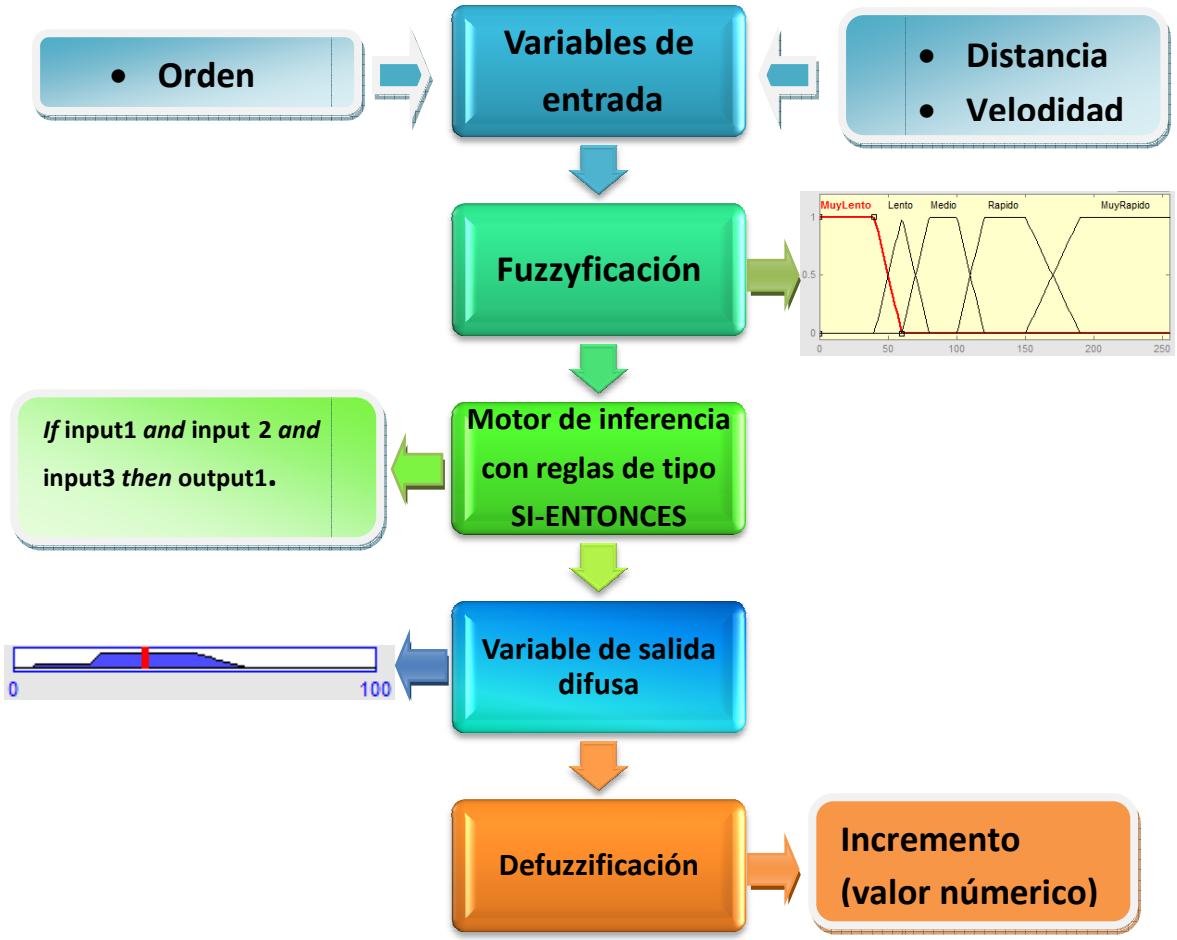
**El primer dominio de definición será la velocidad** con un rango de valores de 0 a 255, que estará comprendido en 5 conjuntos difusos con las siguientes etiquetas lingüísticas: **muy lento, lento, medio, rápido y muy rápido.**

**El segundo dominio de definición es la distancia** con un rango de valores de 0 a 500, que estará comprendido en 4 conjuntos difusos con las siguientes etiquetas: **Cerca, medio, lejos y muy lejos.**

**El tercer dominio de definición es la orden** con un rango de definición de 0 a 3, que estará comprendido en 2 conjuntos difusos con las siguientes etiquetas lingüísticas: **más**

**despacio y más deprisa,** estos conjuntos difusos solo podrán tener un valor; para más despacio un 1 y para más deprisa un 2, ya que son las órdenes.

El proceso que seguirá el sistema de lógica difusa será el siguiente:




El primer paso nuestro sistema de lógica difusa es convertir las señales de entrada (velocidad, distancia y orden) en un conjunto de variables difusas ***(Fuzzificación)***. Se asignan valores (valores difusos) a partir de un conjunto de funciones de pertenencia. El *fuzzificador* se utiliza para saber el nivel de pertenencia al que pertece las señales de entrada.

El segundo paso es el bloque de decisión, ya que es el encargado de la activación de las reglas en función del nivel de pertenencia, determinando el valor de la salida del sistema (variación de la velocidad).

El tercer paso *defuzzificación* es el encargado de pasar los valores difusos a valores reales para pasárselo en nuestro caso al motor.

Para la creación del sistema de lógica difusa se ha utilizado la herramienta toolbox fuzzy *[8, 20]* de MATLAB como ya se explicó en apartados anteriores. A continuación se irá explicando el procedimiento que se ha llevado para la creación del programa.

Primero se abre MATLAB y para acceder a la herramienta toolbox fuzzy se escribe *fuzzy*. Una vez escrito el comando aparecerá una ventana sobre la que se creará el programa.

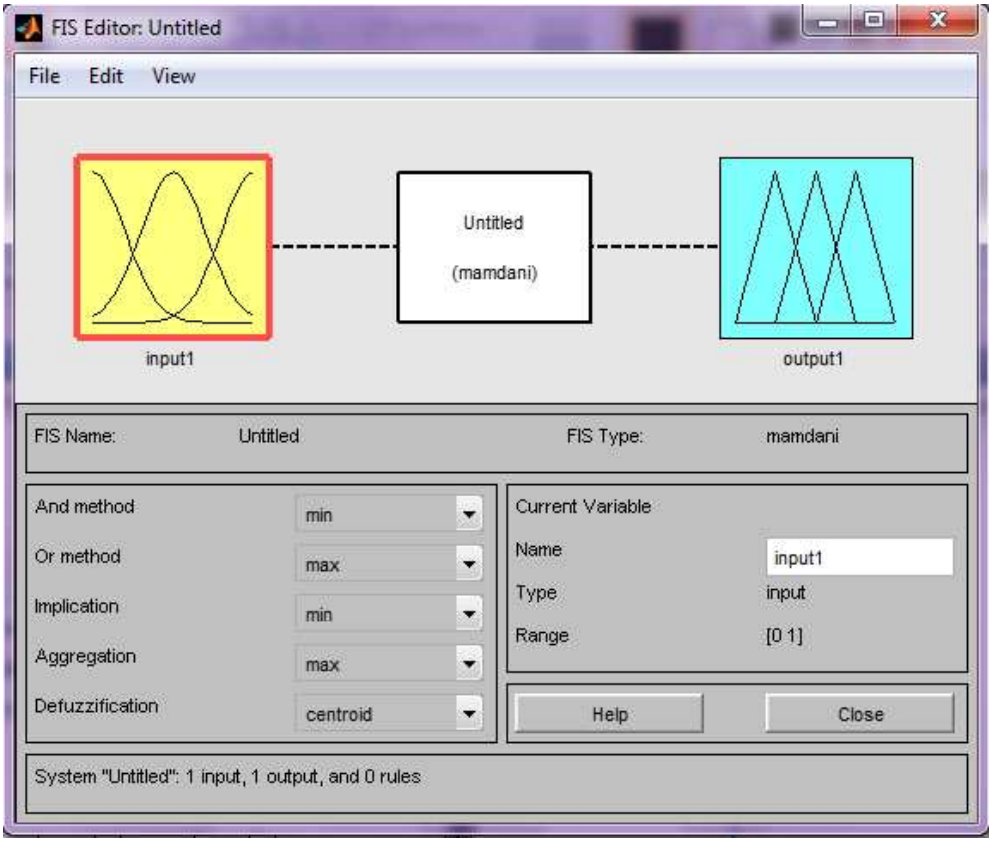



**Ilustración 130:** Herramienta toolbox fuzzy de MATLAB.

Una vez abierto el programa y seleccionado el modo en el que se desea hacer la lógica (en este caso se hará con Mamdani) hay que añadir las variables del sistema. Para este caso hay 3 variables de entrada (velocidad, distancia y comando), por lo tanto hay que crear tres variables que se añaden en ***edit→Add Variable…*** Una vez añadidas las tres variables de entrada se pasa a editarlas haciendo doble clic sobre una de ellas.

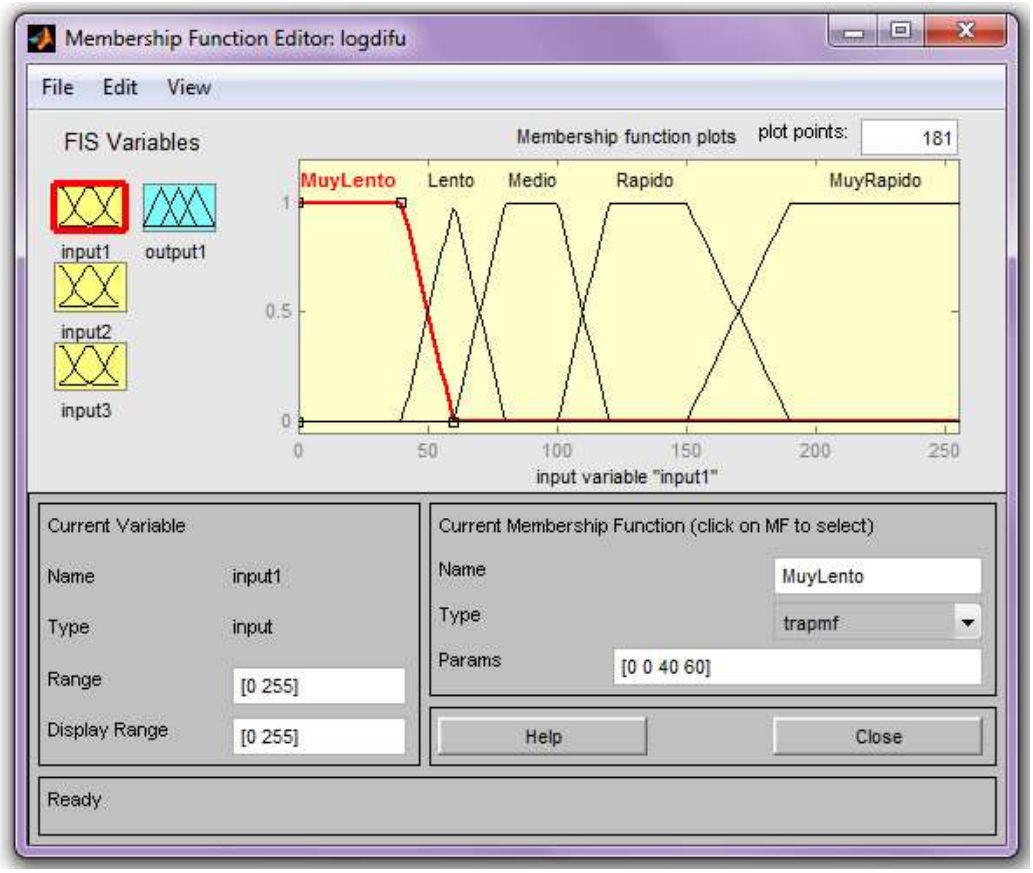



**Ilustración 131:** Editor de variables lógica difusa.

La variable de entrada 1 será la velocidad, donde se ha definido así como se muestra en la ilustración 131 y se explicará a continuación.

A la variable de entrada velocidad **(ilustración 132)** se le da 5 casos diferentes (muy lento, lento, medio, rápido, muy rápido). Para agregar casos se hace clic sobre ***edit→Add MFs*** y se agregan los casos necesarios.

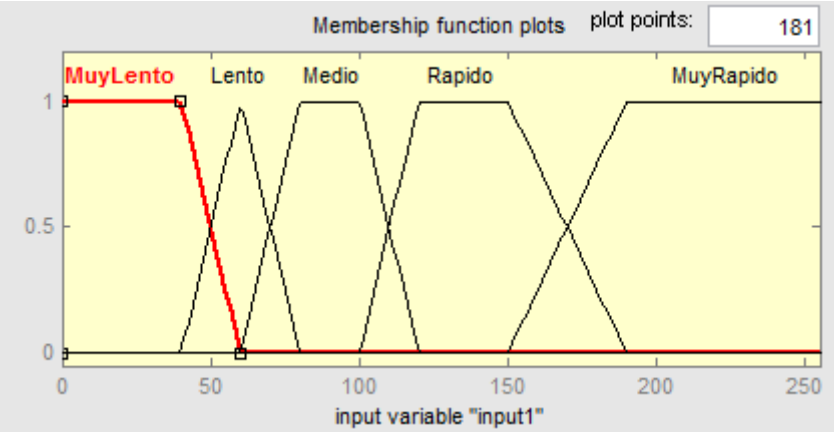



**Ilustración 132:** Variable velocidad.

Antes de explicar la variable ***velocidad***, decir que los valores dados no son la medida real de velocidad del robot, si no los valores analógicos (desde 0-255) con los que trabaja Arduino en sus salidas para el control de velocidad de los motores (PWM). Con estos valores se han obtenido buenos resultados y se puede comparar con la velocidad sin problemas.

El caso ***muy lento*** es un trapecio y se ha definido en [0 0 40 60] ya que el robot empieza a moverse a partir de 30 aproximadamente según el tipo de terreno y cuando el valor es 60 el robot se ha considerado que sigue lento. Se ha elegido un trapecio por la movilidad del robot, ya que dependiendo del terreno a valores entre [30-40] empieza a moverse, por eso se define como una línea recta, y a partir de 40 el robot empieza a moverse aumentando la velocidad, por eso se define como una pendiente descendiente en los valores [40-60].

Para el caso ***lento*** se ha definido como un triángulo [40 60 80] porque cuando se le ordena al robot que *avance* estando parado, empieza a moverse siempre desde el valor 60. A partir de 60 el robot va cogiendo velocidad más rapido, por eso empieza a descender para entrar enseguida en el caso ***medio.***

En el caso ***medio*** se ha definido como un trapecio [60 80 100 120] con la recta entre 80-100 ya que se considera que el robot no va aun rápido, pero a partir de 100 el robot ya empieza a ir rápido. Ambas pendientes del trapecio tienen la misma inclinación ya que se ha considerado que la velocidad del robot empieza a ser “medio” igual a la que empieza a ser “rápido”.

El caso ***Rápido*** se encuentra definido en [100 120 150 190] y es un trapecio, la pendiente de 100-120 es así por la pendiente descendiente del caso *medio*. A un valor de 110 se considera que el robot va 50% Medio y 50% Rápido. A partir de 120 la velocidad del robot es rápido manteniendose en ese estado hasta el valor de 150. Una vez pasado el valor 150 el robot va acelerando pero cada vez lo va haciendo más despacio, por eso la pendiente es menos inclinada que los casos anteriores. Al valor de 170 se encuentra en mitad de *rápido* y *muy rápido.*

Para el caso ***Muy rápido*** está definido en un trapecio [150 190 255 255] con la misma pendiente que *rápido* debido a que se considera que en la misma forma que deja de ser

*rápido* pasa a ser *muy rápido.* A partir de 190 hasta 255 se considera que el robot es muy rápido. La segunda pendiente del trapecio esta definida en 255-255 ya que el robot no puede pasar de estos valores.

Una vez definido la variable 1 de entrada se pasa a definir la variable de entrada 2. Esta segunda variable se llama ***distancia*** **(ilustración133)**. La distancia, en centimetros, es medida por el sensor ultrasónico incorporado en el robot, esta medida se manda al ordenador para su posterior procesamiento. La variable *distancia* está definida por 4 casos (cerca, medio, lejos, muy lejos), que se explica con detalle a continuación.

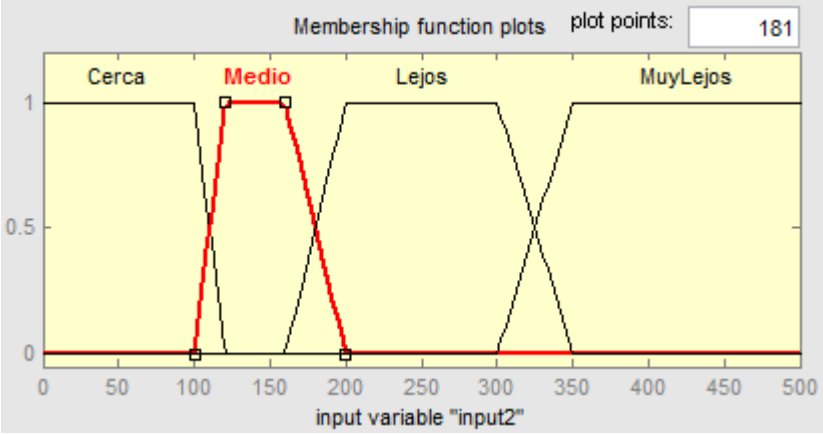



**Ilustración 133:** Variable distancia.

Caso ***cerca*** está definido como un trapecio [0 0 100 120] la primera pendiente definida en [0 0] es porque el sensor no hace medidas negativas. Se considera que de 0 a 100 cm el robot se encuentra cerca del objeto. A partir de 100 hasta 120 cm pasa por una pendiente muy inclinada porque se considera que pasa muy rapido de cerca a medio.

El caso ***medio*** es un trapecio definido en [100 120 160 200] con una recta pequeña entre 120 y 160 cm considerando esas medidas como una distancia media. A partir de 160 hasta 200 se encuentra la pendiente, ya que el robot empieza a estar lejos del objeto.

Para el caso de ***lejos*** también esta definido como un trapecio [160 200 300 350] pero con una recta mucho mayor que la anterior. La recta esta comprendida entre 200 y 300 cm porque se ha considerado que el robot a esas distancias del objeto está lejos. La pendiente desde 300 a 350 cm es un poco mas inclinada que la otra ya que se ha considerado que de lejos a muy lejos es menor la diferencia con la que pasa de un caso a otro.

El último caso ***muy lejos*** esta definido en [300 350 500 500]. Para unas distancias entre 350 y 500 cm se considera que el robot se encuentra muy lejos del obstáculo. Se ha definido lo último en [500 500] porque el sensor ultrasónico no mide mas de 500 cm.

La tercera variable de entrada representa las ***ordenes*** **(ilustración 134)** (más despacio, más deprisa), definidas como triángulos. La orden ***más despacio*** está definida en [0.99 1 1.01], se ha definido así ya que el único valor que puede tomar esa orden es de 1, por lo tanto siempre que se ordene entrará en pertenecia absoluta. La segunda orden ***más deprisa*** se encuentra definida en [1.99 2 2.01] es igual que la orden *más despacio* lo único que para ordenar esta orden sería el 2. En definitiva esta variable solo puede tomar 2 valores, 1 ó 2. Esta orden será ordenada mediante voz. El roconocedor de voz reconocerá *más deprisa o más despacio* e internamente le asignara un 1 ó un 2 según corresponda.

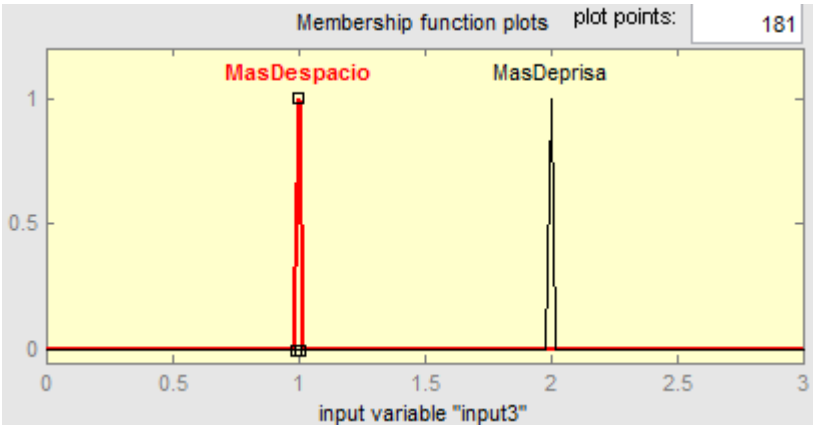



**Ilustración 134:** Variable de entrada orden.

Si se quisiera cambiar el comportamiento del coche, sería tan fácil como cambiar los conjuntos difusos de las variables de entrada sin la necesidad de cambiar una sola línea de programación.

Una vez defina las variables de entrada, se debe definir las variables de salida. En este caso solo hay una variable de salida llamada ***incremento*** **(ilustración 135).** Esta variable de salida es el resultado de las reglas definidas del programa que se verá más adelante. El *incremento* representa el aumento o disminución del parámetro velocidad. A continuación se verá con más detalle.

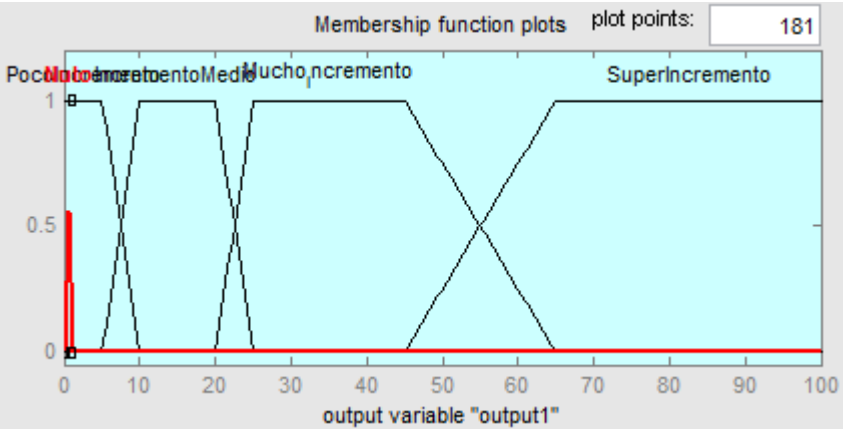



**Ilustración 135:** Variable de salida incremento.

La variable de salida *incremento* **(ilustración 135)** esta definido en un dominio de definición de la variable, rango de valores posible de 0 a 100, cubiertos por 5 conjuntos difusos con etiquetas lingüísticas asociadas (nulo, poco incremento, incremento medio, mucho incremento, súper incremento).

El caso ***nulo*** está definido como un triángulo en [0 1 1.01], está definido así porque en el caso que no se quiera como resultado un incremento, es decir, que el robot se mantenga como esté antes de ordenar una nueva orden de más despacio o más deprisa. Si sale como resultado este caso el incremento será prácticamente nulo.

Para el caso ***poco incremento*** definido de forma trapezoidal en [0 0 5 10] da como respuesta un incremento muy pequeño comprendido entre aproximadamente 5-10. Lo mismo para el caso ***incremento medio*** definido en [5 10 20 25] que da como respuesta un incremento comprendido entre 10-20 aproximadamente. Cabe decir que en el caso de poco incremento se puede dar un incremento como resultado distinto de 5-10, por ejemplo de 4 ó 12, esto es según la pertenencia de cada estado de las variables, pero siempre suele estar cerca de las rectas definidas. Estos dos estados se han creado para cuando no se quiera una aceleración o desaceleración muy brusca debido a la distancia que quede al obstáculo.

En el caso de ***mucho incremento*** definido también en trapecio [20 25 45 65] el incremento es mucho mayor, que en los casos anteriores. La pendiente que va entre 45 y 65 se ha definido menos inclinada para que el paso de un caso a otro sea más suave, es decir, que el robot va a tener un rango entre 45-65 que se encontrará en los dos estados y dependiendo del grado de pertenencia pertenecerá a un estado más que a otro. Este estado ha sido creado

para cuando se necesite un resultado de una aceleración o desaceleración un poco más rápida.

Para el último caso ***súper incremento*** definido como un trapecio en [45 65 100 100] da como resultado un incremento muy grande. El robot en este caso cambiaría su velocidad bruscamente pasando a velocidades muy altas o pasaría de velocidades altas a bajas, según la necesidad que se desea. Este estado se ha hecho para casos extremos en los que el robot necesite reducir su velocidad muy rápido o aumentar la velocidad mucho.

Cuando se tienen todas las variables de entrada y salida definidas se deben definir las reglas del sistema, **estas reglas constituyen el motor de razonamiento del sistema artificial**. Antes de empezar a definir las reglas se explicará el comportamiento del robot que se desea obtener.

A partir de los valores de las variables de entrada, se usan las funciones de pertenencia a los conjuntos difusos, y se sustituye ese valor numérico por etiquetas lingüísticas. En segundo lugar, se mira la activación de las reglas. Una regla se activa cuando se satisfacen sus antecedentes de forma parcial o absoluta. El consecuente de las reglas activadas, determina el valor de salida del sistema (variación de la velocidad).

Se pretende que el robot tenga una respuesta no tan prefijada dependiendo de una serie de variables. Estas variables son las definidas anteriormente (velocidad, distancia y órdenes), en función de la situación que se encuentre el robot ante estas variables la respuesta será una u otra, no siendo siempre la misma, sino que, para cada situación habrá una respuesta diferente.

Se quiere conseguir que para la orden *más deprisa*, si la velocidad del robot es baja y la distancia a la que se encuentra el obstáculo es grande, el robot aumente su velocidad más rápidamente que si el robot se encuentra en la situación de que la velocidad del robot sea media o rápida y la distancia al obstáculo sea mucho menor. Y para la orden *más despacio* se quiere conseguir lo contrario, es decir, si el robot va muy deprisa y el obstáculo se encuentra cerca, la desaceleración será mucho mayor que si el robot va medianamente deprisa y el obstáculo esté lejos.

Una vez explicado el comportamiento que se quiere dar al robot, hay que definir ese comportamiento en reglas teniendo en cuenta todas las variables declaradas anteriormente, sin que se quede ningún caso sin definir, ya que habría situaciones en el que el robot no sabría qué hacer.

Para definir las reglas se hace clic sobre ***edit→Rules…*** y aparecerá una ventana donde se hará la definición de reglas.

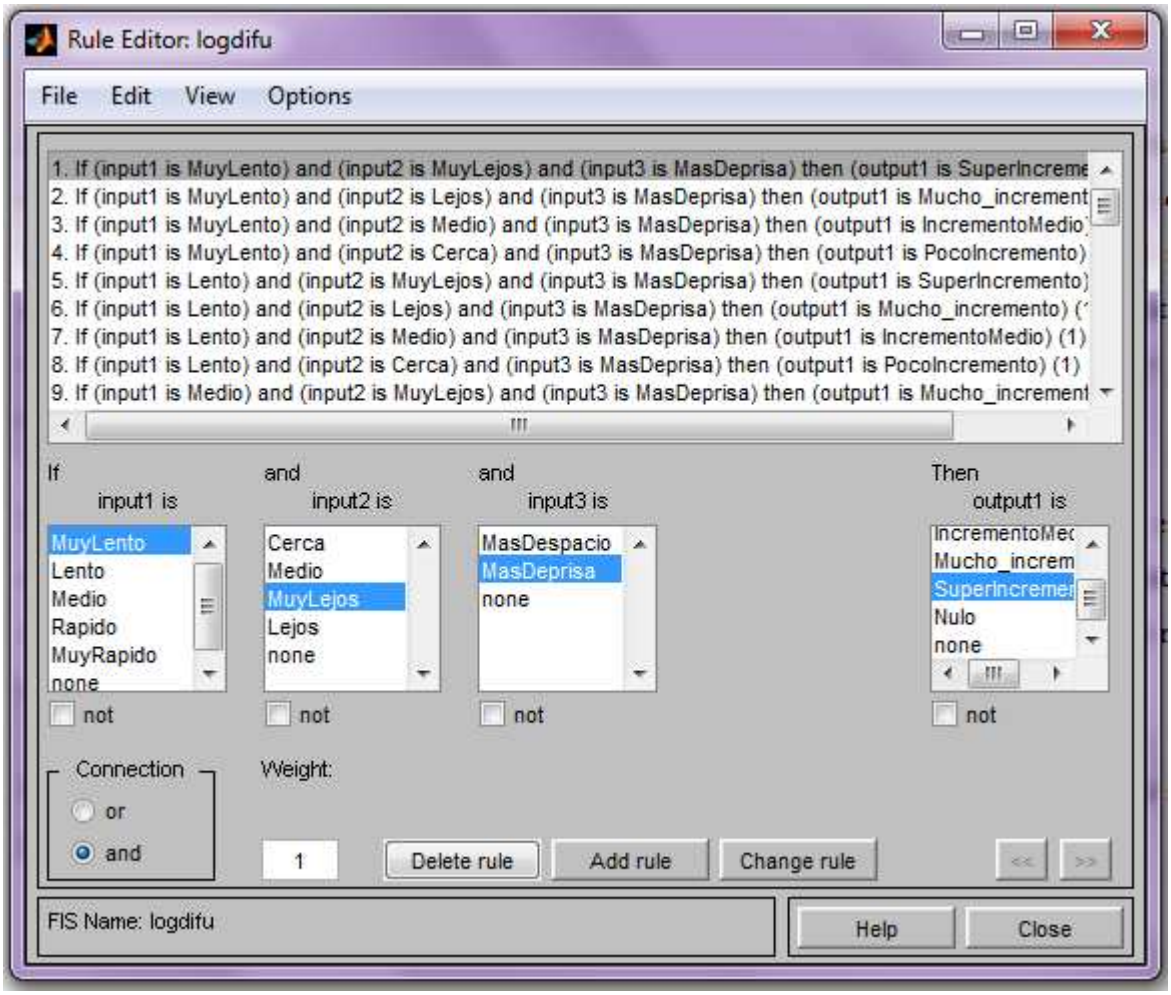



**Ilustración 136:** Definición de reglas lógica difusa.

Para definir las reglas hay que seleccionar los casos que sean necesarios y el resultado que queremos obtener cuando se dan esos casos. Una vez seleccionado los casos se hace clic sobre ***Add rule*** y la regla quedará guardada. Este procedimiento hay que hacerlo para todos los casos posibles que se puedan dar. A continuación se verán todas las reglas que se han creado para el comportamiento del robot.

1. If (input1 is MuyLento) and (input2 is MuyLejos) and (input3 is MasDeprisa) then (output1 is SuperIncremento) (1)
2. If (input1 is MuyLento) and (input2 is Lejos) and (input3 is MasDeprisa) then (output1 is Mucho_incremento) (1)
3. If (input1 is MuyLento) and (input2 is Medio) and (input3 is MasDeprisa) then (output1 is IncrementoMedio) (1)
4. If (input1 is MuyLento) and (input2 is Cerca) and (input3 is MasDeprisa) then (output1 is PocoIncremento) (1)
5. If (input1 is Lento) and (input2 is MuyLejos) and (input3 is MasDeprisa) then (output1 is SuperIncremento) (1)
6. If (input1 is Lento) and (input2 is Lejos) and (input3 is MasDeprisa) then (output1 is Mucho_incremento) (1)
7. If (input1 is Lento) and (input2 is Medio) and (input3 is MasDeprisa) then (output1 is IncrementoMedio) (1)
8. If (input1 is Lento) and (input2 is Cerca) and (input3 is MasDeprisa) then (output1 is PocoIncremento) (1)
9. If (input1 is Medio) and (input2 is MuyLejos) and (input3 is MasDeprisa) then (output1 is Mucho_incremento) (1)
10. If (input1 is Medio) and (input2 is Lejos) and (input3 is MasDeprisa) then (output1 is IncrementoMedio) (1)
11. If (input1 is Medio) and (input2 is Medio) and (input3 is MasDeprisa) then (output1 is PocoIncremento) (1)
12. If (input1 is Rapido) and (input2 is MuyLejos) and (input3 is MasDeprisa) then (output1 is Mucho_incremento) (1)
13. If (input1 is Rapido) and (input2 is Lejos) and (input3 is MasDeprisa) then (output1 is IncrementoMedio) (1)
14. If (input1 is Rapido) and (input2 is Medio) and (input3 is MasDeprisa) then (output1 is PocoIncremento) (1)
15. If (input1 is MuyRapido) and (input2 is MuyLejos) and (input3 is MasDeprisa) then (output1 is IncrementoMedio) (1)
16. If (input1 is MuyRapido) and (input2 is Lejos) and (input3 is MasDeprisa) then (output1 is PocoIncremento) (1)
17. If (input1 is MuyRapido) and (input2 is MuyLejos) and (input3 is MasDespacio) then (output1 is IncrementoMedio) (1)
18. If (input1 is MuyRapido) and (input2 is Lejos) and (input3 is MasDespacio) then (output1 is IncrementoMedio) (1)
19. If (input1 is MuyRapido) and (input2 is Medio) and (input3 is MasDespacio) then (output1 is SuperIncremento) (1)
20. If (input1 is MuyRapido) and (input2 is Cerca) and (input3 is MasDespacio) then (output1 is SuperIncremento) (1)
21. If (input1 is Rapido) and (input2 is MuyLejos) and (input3 is MasDespacio) then (output1 is IncrementoMedio) (1)
22. If (input1 is Rapido) and (input2 is Lejos) and (input3 is MasDespacio) then (output1 is IncrementoMedio) (1)
23. If (input1 is Rapido) and (input2 is Medio) and (input3 is MasDespacio) then (output1 is Mucho_incremento) (1)
24. If (input1 is Rapido) and (input2 is Cerca) and (input3 is MasDespacio) then (output1 is SuperIncremento) (1)
25. If (input1 is Medio) and (input2 is MuyLejos) and (input3 is MasDespacio) then (output1 is PocoIncremento) (1)
26. If (input1 is Medio) and (input2 is Lejos) and (input3 is MasDespacio) then (output1 is IncrementoMedio) (1)
27. If (input1 is Medio) and (input2 is Medio) and (input3 is MasDespacio) then (output1 is Mucho_incremento) (1)
28. If (input1 is Medio) and (input2 is Cerca) and (input3 is MasDespacio) then (output1 is SuperIncremento) (1)
29. If (input1 is Lento) and (input2 is MuyLejos) and (input3 is MasDespacio) then (output1 is PocoIncremento) (1)
30. If (input1 is Lento) and (input2 is Lejos) and (input3 is MasDespacio) then (output1 is PocoIncremento) (1)
31. If (input1 is Lento) and (input2 is Medio) and (input3 is MasDespacio) then (output1 is IncrementoMedio) (1)
32. If (input1 is Lento) and (input2 is Cerca) and (input3 is MasDespacio) then (output1 is IncrementoMedio) (1)
33. If (input1 is MuyLento) and (input2 is Medio) and (input3 is MasDespacio) then (output1 is PocoIncremento) (1)
34. If (input1 is MuyLento) and (input2 is Cerca) and (input3 is MasDespacio) then (output1 is PocoIncremento) (1)
35. If (input1 is Medio) and (input2 is Cerca) and (input3 is MasDeprisa) then (output1 is Nulo) (1)
36. If (input1 is Rapido) and (input2 is Cerca) and (input3 is MasDeprisa) then (output1 is Nulo) (1)
37. If (input1 is MuyRapido) and (input2 is Medio) and (input3 is MasDeprisa) then (output1 is Nulo) (1)
38. If (input1 is MuyRapido) and (input2 is Cerca) and (input3 is MasDeprisa) then (output1 is Nulo) (1)
39. If (input1 is MuyLento) and (input2 is MuyLejos) and (input3 is MasDespacio) then (output1 is Nulo) (1)
40. If (input1 is MuyLento) and (input2 is Lejos) and (input3 is MasDespacio) then (output1 is Nulo) (1)

**Ilustración 137:** Reglas comportamiento robot.

Una vez que se tienen todas las reglas definidas se tiene que guardar el programa haciendo clic sobre ***file→Export→To File…*** y se guarda con el nombre deseado, en este caso se ha guardado como ***logdifu.*** Es importante el nombre con el que se guarda el archivo, ya que luego hay que llamarlo en el programa general de MATLAB para su utilización.

Para la utilización en MATLAB del programa generado de lógica difusa se hace mediante códigos, pero primeramente hay que exportarlo al espacio de trabajo de MATLAB de la siguiente manera: hacer clic en ***file→Export→Workspace*** y se elige el nombre que se desea para después su utilización. Una vez exportado ya se puede hacer uso de él mediante código. A continuación se verá el código que se ha utilizado en el programa de MATLAB de reconocimiento de voz para hacer uso de la lógica difusa.

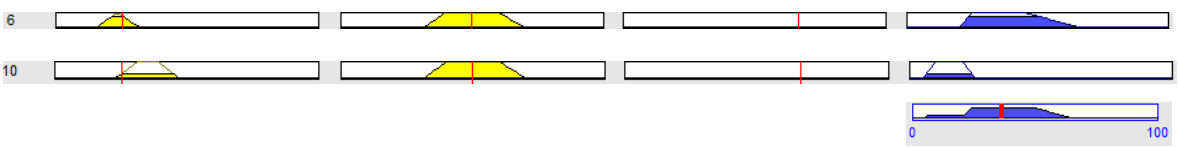
```
fismat=readfis('logdifu');
incremento=evalfis([m1(2), m1(1), 2],fismat);
```

*fismat=readfis('logdifu')* es la función que carga el programa de lógica difusa creado con ese nombre. La función *evalfis* es la encargada de pasar el valor de las variables al programa de lógica difusa, donde en este caso *m1(2)* es la velocidad y *m1(1)* es la distancia, y el valor 2 es la orden que se desea que haga (*más deprisa)*. Luego una vez pasado los valores el programa nos devuelve el resultado guardado en *incremento*, que será el incremento que se sumará o restará a la velocidad del robot en ese momento.

Para la modificación de los valores de las variables no hace falta cambiar todo el programa, solo basta con modificar la definición de las variables, con esto es suficiente y sin necesidad de modificar las reglas. Si se cambian los valores el robot tendrá un comportamiento totalmente distinto aunque atienda a las mismas reglas definidas.

Durante la realización de esta fase aparecieron nuevos problemas, en los que el robot no tenía una buena respuesta en función de la distancia al obstáculo. Estos problemas se fueron resolviendo haciendo un estudio con diferentes formas de definir las variables del sistema, llegando a la versión final de lógica difusa con un resultado satisfactorio.

Una vez diseñado el sistema se hará un ejemplo del funcionamiento. Se cogerá una "velocidad" de 65, distancia 250 y orden más deprisa (2). Como resultado se obtiene que la velocidad pertenece 0.75 a lento y 0.25 a medio, la distancia pertenece obsolutamente a lejos y la orden también pertece absolutamente a más deprisa. Como consecuente se activan dos reglas:




Las reglas activadas son la 6 y la 10 que dicen lo siguiente:

6. If (input1 is Lento) and (input2 is Lejos) and (input3 is MasDeprisa) then (output1 is Mucho_incremento) (1)
10. If (input1 is Medio) and (input2 is Lejos) and (input3 is MasDeprisa) then (output1 is IncrementoMedio) (1)

Como para una “velocidad” de 65 pertenece a dos conjuntos difusos con un cierto grado de pertenencia se activan ambas reglas. En las dos reglas la distancia es lejos con pertenencia 1.

Para la regla 6 se obtiene: 0.75 lento *and* 1 lejos *and* 1 más deprisa *then* mucho incremento, por lo tanto se coge el menor, o sea se coge el 0.75 correspondiente a mucho incremento. 

Para la regla 10 se obtiene: 0.25 lento *and* 1 lejos *and* 1 más deprisa *then* mucho incremento, por lo tanto se coge el menor, o sea se coge el 0.25 correspondiente a incremento medio. 

Como resultado tenemos una respuesta de la siguiente manera:


El valor de la salida será el resultado de la *defuzzificación,* ya que es el encargado de pasar el valor difuso a valores numéricos. El método de *defuzzificación* que se lleva ha cabo en este sistema es por el método del centroide. El valor de incremento que se da es 36.7.

El método del centroide se transforma la salida difusa en un número real el cual es la coordenada del centro de gravedad de tal conjunto difuso de salida.

$$y_d = \frac{\int_S y\mu_Y(y)dy}{\int_S \mu_Y(y)dy}$$

Donde $\mu_y$ es la función de pertenencia del conjunto de salida *Y,* cuya variable de salida es *y*. *S* es el dominio o rango de integración. El método es costoso de hacer pero el más preciso, por lo tanto con un ordenador este método se puede hacer como se ha hecho en este caso. Como se puede ver aquí 

la franja roja es el centro de gravedad del conjunto de pertenencia de salida, es este caso con un valor de 36.7.

## 4.11 Programación Arduino.

Para programar el robot se hace con el programa Arduino y se basa en lenguaje de programación C/C++ *[21, 23]*.

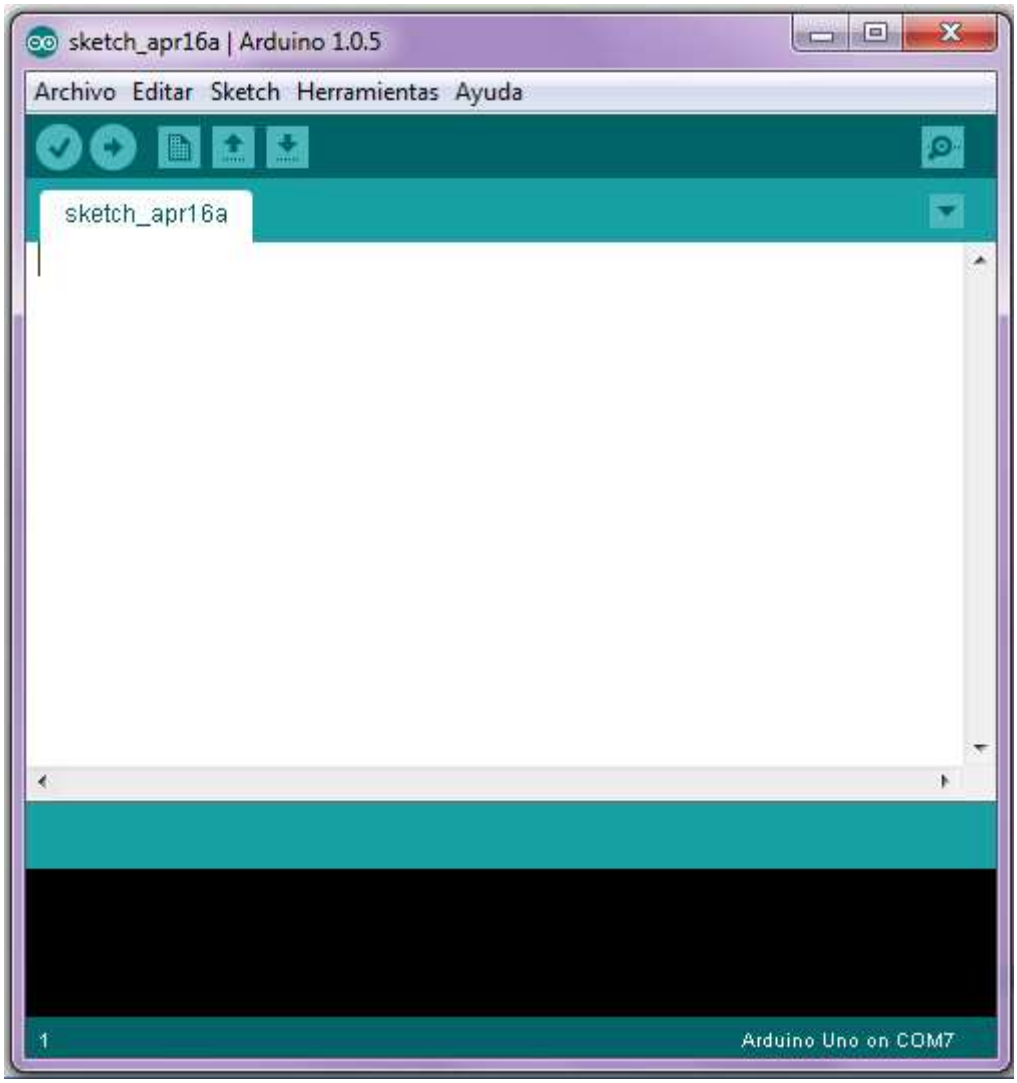



**Ilustración 138:** Programa Arduino.

Para poder controlar el robot hace falta programar unas series de procesos para que el robot se comporte de la forma deseada.

La forma de controlar el robot será mediante la voz, pero para que el robot pueda interpretar esas órdenes hace falta un segundo programa que estará internamente en el robot (placa Arduino). Estas órdenes serán mandadas desde el PC al robot mediante **radiofrecuencia,** por lo tanto la placa Arduino que será la encargada de procesar todas las

órdenes, tendrá que tener un programa para que pueda captar la información mandada desde el PC y a su vez ordenar internamente el movimiento de motores.

También hay que programar una sección donde el robot tome las medidas del obstáculo y saber el valor que se le está dando a los motores para el control de la velocidad y ambas enviarlas por radiofrecuencia al PC para que puedan ser procesadas en el programa de lógica difusa.

Internamente el robot también podrá tomar decisiones propias ante situaciones que el obstáculo se encuentre muy cerca. Se programará para que a partir de 60 cm el robot empiece a disminuir su velocidad en la misma medida que la distancia. Es decir, cuando llegue a 60 cm el robot se ponga a parámetro de velocidad 60 y vaya disminuyendo en el mismo valor que la distancia, con esto se conseguirá una reducción de velocidad suave. Cuando el obstáculo este a 30 cm el robot se detiene por completo.

Otra cosa que se deberá programar es el control del robot con el joystick, haciendo independiente el control de voz con este control.

A continuación se verá todo el código que hace todas las funciones anteriormente dichas.

```
#define MOTOR1_I1 2 //I1
#define MOTOR1_I2 4 //I2 //MOTOR1 RUEDA DERECHA
#define MOTOR1_PWM 3 //EA

#define MOTOR2_I3 5 //I3
#define MOTOR2_I4 7 //I4 //MOTOR2 RUEDA IZQUIERDA
#define MOTOR2_PWM 6 //EB
//El código de arriba es lo que va a definir los sentidos de los motores.

#define COCHE_ADELANTE 0
#define COCHE_ATRAS 1
#define COCHE_IZQUIERDA 2 // todo este párrafo de código define los
#define COCHE_DERECHA 3 // movimientos del robot
#define COCHE_STOP 4
#define MOTOR_STOP 5

#include <Ultrasonic.h>
Ultrasonic ultrasonido(8,12,26100);// definición del sensor ultrasonico
float distancia;
int control;
int orden2;
int der;
int izq;
int i;
int caso=0;
int on;
int set;
double fuzzy;
```

```
int dato;
int y;
double velocidad;
void setup(){
  Serial.begin(9600);
  // DEFINIR PINES MOTOR 1
  pinMode(MOTOR1_I1, OUTPUT);
  pinMode(MOTOR1_I2, OUTPUT);
  pinMode(MOTOR1_PWM, OUTPUT);
  // DEFINIR PINES MOTOR 2
  pinMode(MOTOR2_I3, OUTPUT);
  pinMode(MOTOR2_I4, OUTPUT);
  pinMode(MOTOR2_PWM, OUTPUT);
}
void setSpeed(char motor_num, char velocidad_motor){
  // esta función es la que controla la velocidad de los motores.
  // Los parámetros der y izq son los encargados de incrementar
  // la velocidad de cada rueda de forma individual, ordenado
  // a través de los botones 5, 6, 7, 8 del mando.
  if (motor_num==1){
    analogWrite(MOTOR1_PWM, (velocidad_motor)+(der));
  }
  else if (motor_num==2){
    analogWrite(MOTOR2_PWM, (velocidad_motor)+(izq));
  }
  else if (motor_num==3){
    analogWrite(MOTOR1_PWM, (velocidad_motor)+(der));
    analogWrite(MOTOR2_PWM, (velocidad_motor)+(izq));
  }
}
void motorStart(char motor_num, byte direccion){
  // esta función es la que activa y desactiva los motores.
  char pin_control1;
  char pin_control2;
  char pin_control3;
  char pin_control4;
  if (motor_num==1){
    pin_control1=MOTOR1_I1;
    pin_control2=MOTOR1_I2;
  }
  else if (motor_num==2){
    pin_control1=MOTOR2_I3;
    pin_control2=MOTOR2_I4;
  }
  else if (motor_num==3){
    pin_control1=MOTOR1_I1;
    pin_control2=MOTOR1_I2;
    pin_control3=MOTOR2_I3;
    pin_control4=MOTOR2_I4;
  }
  switch (direccion){ // controla la dirección del robot
    case COCHE_ADELANTE:
    digitalWrite(pin_control1, HIGH);
    digitalWrite(pin_control2, LOW);
    digitalWrite(pin_control3, HIGH);
    digitalWrite(pin_control4, LOW);
    break;
    case COCHE_ATRAS:
    digitalWrite(pin_control1, LOW);
    digitalWrite(pin_control2, HIGH);
    digitalWrite(pin_control3, LOW);
    digitalWrite(pin_control4, HIGH);
    break;
    case COCHE_IZQUIERDA:
    digitalWrite(pin_control1, HIGH);
    digitalWrite(pin_control2, LOW);
    break;
```

```
    case COCHE_DERECHA:
    digitalWrite(pin_control1, HIGH);
    digitalWrite(pin_control2, LOW);
    break;
    case COCHE_STOP:
    digitalWrite(pin_control1, LOW);
    digitalWrite(pin_control2, LOW);
    digitalWrite(pin_control3, LOW);
    digitalWrite(pin_control4, LOW);
    break;
    case MOTOR_STOP:
    digitalWrite(pin_control1, LOW);
    digitalWrite(pin_control2, LOW);
    break;
  }
}
void loop(){ //programa que define el comportamiento del robot

  //INICIO CONTROL POR VOZ
  distancia=ultrasonido.Ranging(CM);
  if (distancia<60){
          velocidad=distancia;
          setSpeed(3, velocidad);
        }
        if (distancia<30){
          motorStart(1, COCHE_STOP);
        velocidad=0;
        setSpeed(3, velocidad);
        }
  if (Serial.available() > 0) {
    control = Serial.read();
    der=0;
    izq=0;
    switch (control){
      case 1: // orden parar (voz)
      motorStart(3, COCHE_STOP);
      velocidad=0;
      setSpeed(3, velocidad);
      break;
      case 7: // orden avanza (voz)
      distancia=ultrasonido.Ranging(CM);
      set=0;
      while (distancia>30&&control!=1&&control!=3&&control!=2){
        motorStart(3, COCHE_ADELANTE);
        if (set==0){
          velocidad=150;
          setSpeed(3, velocidad);
          set=1;
          delay(10);
          velocidad=60;
        }
        if (set==1){
          setSpeed(3, velocidad);
        }
        distancia=ultrasonido.Ranging(CM);
        if (distancia<60){ // estas líneas de código hacen que el robot
          set=2;           // vaya disminuyendo automáticamente la
          velocidad=distancia; // velocidad Según la distancia
          setSpeed(3, velocidad); //al obstáculo.
        }
        control = Serial.read();
      }
      if (control!=3&&control!=2&&distancia>30){
        motorStart(1, COCHE_STOP);
        velocidad=0;
        setSpeed(3, velocidad);
      }
```

```
break;
case 2: // orden más despacio (voz)
      motorStart(3, COCHE_ADELANTE);
      setSpeed(3, velocidad);
      y=1;
     while (y==1){
      if(Serial.available()>0){
       byte matlab=Serial.read();
          if (matlab==8){
            Serial.println(distancia); // manda dist. y vel.
            Serial.println(velocidad); // por el puerto serie.
          }
          else if (matlab==9){
            int x=1;
            while (x==1){
              if (Serial.available()>0){ // Lee en el puerto serie
                fuzzy=Serial.read();  // el resultado
                velocidad=velocidad-fuzzy; // del programa fuzzy.
                if (velocidad>0){
                  motorStart(3, COCHE_ADELANTE);
                  setSpeed(3, velocidad);
                }
                  x=0;
                  matlab=0;
                  y=0;
              }
            }
          }
      }
     }
break;
case 3: // orden más deprisa (voz)
      motorStart(3, COCHE_ADELANTE);
      setSpeed(3, velocidad);
      y=1;
     while (y==1){
      if(Serial.available()>0){
       byte matlab=Serial.read();
          if (matlab==8){
            Serial.println(distancia);
            Serial.println(velocidad);
          }
          else if (matlab==9){
            int x=1;
            while (x==1){
              if (Serial.available()>0){
                fuzzy=Serial.read();
                velocidad=velocidad+fuzzy;
                if (velocidad<250){
                  motorStart(3, COCHE_ADELANTE);
                  setSpeed(3, velocidad);
                }
                  x=0;
                  matlab=0;
                  y=0;
              }
            }
          }
      }
     }
break;
case 6: // orden retrocede (voz)
motorStart(3, COCHE_ATRAS);
setSpeed(3, 60);
delay(800);
motorStart(3, COCHE_STOP);
velocidad=0;
```

```
setSpeed(3, velocidad);
break;
case 4: // orden gira izquierda (voz)
motorStart(1, COCHE_IZQUIERDA);
setSpeed(1, 65);
motorStart(2, MOTOR_STOP);
setSpeed(2, 0);
delay(600);
motorStart(1, MOTOR_STOP);
setSpeed(1, 0);
break;
case 5: // orden gira derecha (voz)
motorStart(1, MOTOR_STOP);
setSpeed(1, 0);
motorStart(2, COCHE_DERECHA);
setSpeed(2, 78);
delay(600);
motorStart(2, MOTOR_STOP);
setSpeed(2, 0);
break;
//FIN CONTROL POR VOZ
// INICIO CONTROL POR JOYSTICK
case 'e': // Orden encender (joystick)
Serial.println("Joystick ON");
izq=0;
der=0;
orden2=0;
velocidad=55;
while (control=='e'&&orden2!='p'){
  if (Serial.available()) {
  orden2 = Serial.read();
  switch (orden2){
  case '1': // con el botón 1 se acelera
  if (velocidad<250){ // impide que la velocidad sea >255
      velocidad=velocidad+1;
  }
  break;
  case '2':// con el botón 2 se desacelera
  if (velocidad>0){ // Impide que la velocidad nunca sea <0
      velocidad=velocidad-2;
  }
  break;
  case '3':// con el botón 3 se para y pone la velocidad en 55
  motorStart(3, COCHE_STOP);
  velocidad=55;
  setSpeed(3, 0);
  break;
  case '4': // con el botón 4 se pone a cero los parámetros que
  izq=0;    // incrementa la velocidad de las ruedas deseadas
  der=0;
  break;
  case '5': // con el botón 5 se incrementa la velocidad de la
  if (velocidad<250){ //rueda izquierda
  izq=izq+1;
  Serial.print("incremento rueda izquierda ");
  Serial.println(izq);
  }
  break;
  case '6':// con el botón 6 se incrementa la velocidad de la
  if (velocidad<250){ //rueda derecha
  der=der+1;
  Serial.print("incremento rueda derecha ");
  Serial.println(der);
  }
  break;
  case '7':// con el botón 7 disminuye la velocidad de la rueda
  if (velocidad>0){ //izquierda
```

```
        izq=izq-1;
        Serial.print("incremento rueda izquierda ");
        Serial.println(izq);
        }
        break;
        case '8':// con el botón 8 disminuye la velocidad de la rueda
        if (velocidad>0){ //derecha
        der=der-1;
        Serial.print("incremento rueda derecha ");
        Serial.println(der);
        }
        break;
        case 'w': //avanza
            caso=10;
            motorStart(3, COCHE_ADELANTE);
            setSpeed(3, velocidad);
            delay(20);
        break;
        case 's': //retrocede
            caso=11;
            motorStart(3, COCHE_ATRAS);
            setSpeed(3, velocidad);
            delay(20);
        break;
        case 'a'://giro izquierda
            motorStart(1, COCHE_IZQUIERDA);
            setSpeed(1, velocidad);
            motorStart(2, MOTOR_STOP);
            setSpeed(2, 0);
            delay(20);
        break;
        case 'd':// giro derecha
            motorStart(1, MOTOR_STOP);
            setSpeed(1, 0);
            motorStart(2, COCHE_DERECHA);
            setSpeed(2, velocidad+5);
            delay(20);
        break;
        }
             motorStart(3, COCHE_STOP);// cuando no se manda nada por el
             setSpeed(3, 0);           // puerto se para
          }
          // FIN CONTROL POR JOYSTICK.
      }
      Serial.println("Joystick OFF");
      break;
    }
  }
}
```

En la programación del robot apareció un problema donde el robot no seguía una línea recta. Tras buscar información del problema se encontró que es debido a la fabricación de la placa que controla los motores, ya que no es ideal y no da la misma señal de salida a un motor que a otro. Este problema se resolvió dando un incremento extra a la salida que da menos valor hasta equilibrar la velocidad de ambas ruedas y conseguir que el robot vaya en línea recta.

## 4.12 Comunicación MATLAB y ARDUINO.

La comunicación entre MATLAB y Arduino se hace por radiofrecuencia *[4]* (PC y Arduino).

La comunicación es bidireccional ya que PC tiene que mandar órdenes (avanza, gira… etc.) y valores (incrementos) a Arduino y Arduino tiene que mandar valores (distancia y parámetro velocidad) al PC.

Para que la comunicación sea posible tiene que ser via Puerto Serie. En MATLAB hay que crear este puerto mediante código de la siguiente manera:

```
%Se crea la comunicación serial en COM7
arduino = serial('COM7','BaudRate',9600,'Terminator','CR/LF'); %velocidad
% de transmisión de datos 9600 baudios por segundo.
warning('off','MATLAB:serial:fscanf:unsuccessfulRead'); %Elimina mensajes
% de error
%Se abre el puerto para poder enviar los datos
fopen(arduino);
fprintf(arduino,'%s',char(Ind)); %Se envía el DATO almacenado en Ind.
fclose(arduino); %Se cierra el puerto
```

Este fragmento de código anterior es el encargado de mandar la orden a Arduino. Esta orden es la que posteriormente es reconocida dicha por voz y se almacena en "*Ind*".

Para que MATLAB lea datos del puerto serie se hace mediante el siguiente fragmento de código.

```
Nvalores=2; %Cantidad de valores que queremos  leer
m1=zeros(1,Nvalores); %Crea un vector con el numero de valores a leer.
i=1;
k=0;
fprintf(arduino,'%s',char(8));
while k<Nvalores % se hace un bucle de dos al tener velocidad y distancia
      %Leer el puerto serie
      a = fscanf(arduino,'%f.%f'); %Con esta función se lee el puerto
      % serie
      m1(i)=a(1);
      %Incrementar el contador
      i=i+1;
      k=k+1;
end
```

Para poder leer los valores escritos en el puerto serie por Arduino, primero se debe mandar desde MATLAB una orden de inicio de escritura a Arduino. Arduino capta esa orden y enseguida escribe los valores en el puerto serie para que MATLAB lo lea. Si este proceso

no se hace de forma sincronizada MATLAB se queda a la espera de que le llegue algo por el puerto serie y al final da un error que no hay nada disponible, ya que Arduino mandaría el dato mucho antes de que MATLAB estuviera listo para poder leer el puerto serie.

La parte de Arduino para escribir en el puerto serie se hace mediante código de la siguiente manera.

```
if(Serial.available()>0){
    byte matlab=Serial.read();// lee el puerto serie
       if (matlab==8){
         Serial.println(distancia); // manda dist. y vel.
         Serial.println(velocidad); // por el puerto serie.
```

Este fragmento de código es el encargado de captar si MATLAB da la orden a Arduino de que escriba la distancia y la velocidad. Una vez captada la orden, Arduino manda los datos.

En el proceso de realización de esta librería se presentó un problema en la sincronización entre ambos. El problema era que cuando Arduino escribía los datos en el puerto serie Matlab no estaba preparado para leer el puerto, con el resultado que Matlab cuando estaba listo para leer esperaba un tiempo dando un error al no recibir nada. Esto fue solucionado sincronizando los dos, es decir, que MATLAB ordene a Arduino cuando tiene que mandar los datos, evitando así que MATLAB no esté preparado cuando Arduino escriba los datos en el puerto. Con la librería final se obtiene una buena comunicación en tiempo real entre ambos.

## 4.13 Control del robot por joystick.

La decisión de poder controlar el robot con un segundo método de control se ha hecho para resolver el problema que puede presentarse cuando haya mucho ruido, ya que el reconocedor de voz no funcionaría correctamente.

Para poder controlar el robot mediante un mando de videojuegos hace falta un programa que interactúe con Arduino. Se necesita un programa capaz de simular cada pulsación del mando como una pulsación de teclado. Estas pulsaciones de teclado serán leídas por el puerto serie del programa de Arduino transmitiendo las órdenes resultantes a través de radiofrecuencia al robot.

El programa utilizado para la simulación de pulsaciones de letras del teclado se llama **Xpadder (ilustración 139).**

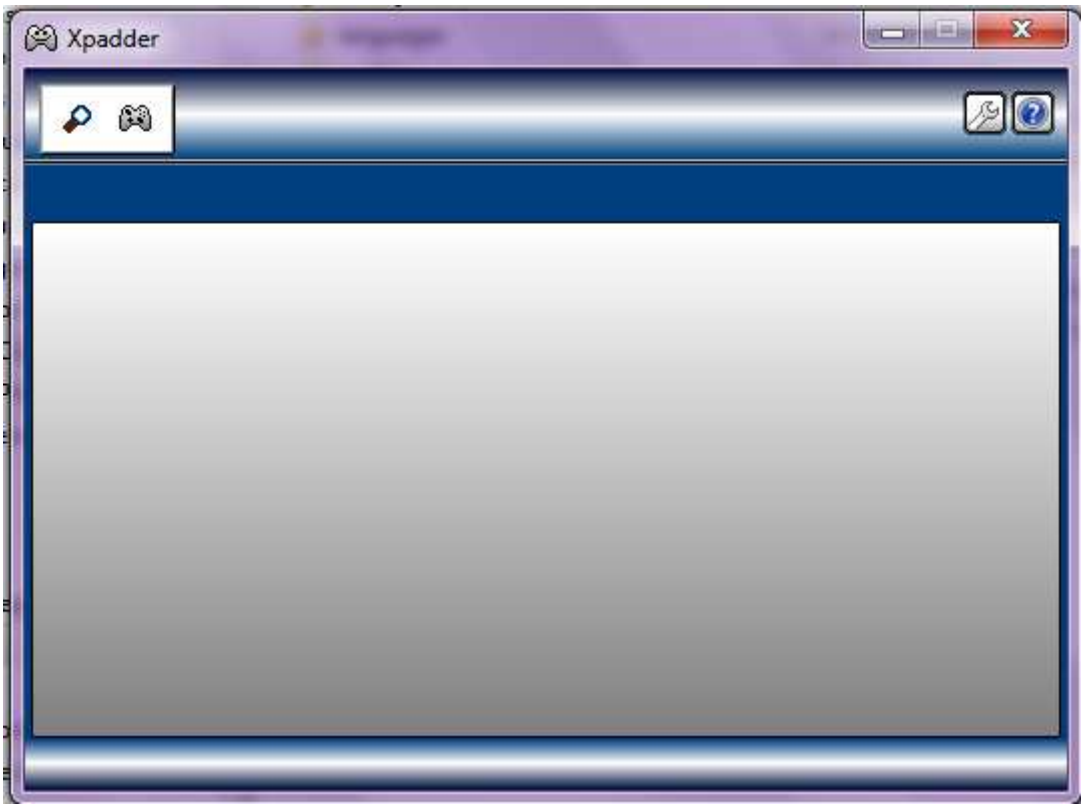



**Ilustración 139:** Programa Xpadder.

En este programa hay que construir nuestro mando poniendo los botones y crucetas del mando que se utilizará. El mando que se utilizará será ***Logitech Precision*** **(ilustración 140)**.

**Ilustración 140:** Mando Logitech Precision.

Una vez decidido el mando que se utilizará se pasará al diseño de él en el programa Xpadder. El primer paso será conectar el dispositivo en el puerto USB para que el programa lo reconozca y se puedan guardar las configuraciones hechas. Una vez reconocido el dispositivo se hace clic sobre el nombre del mando en el programa Xpadder donde aparecerá acto seguido una ventana de ajustes **(ilustración 141)**.

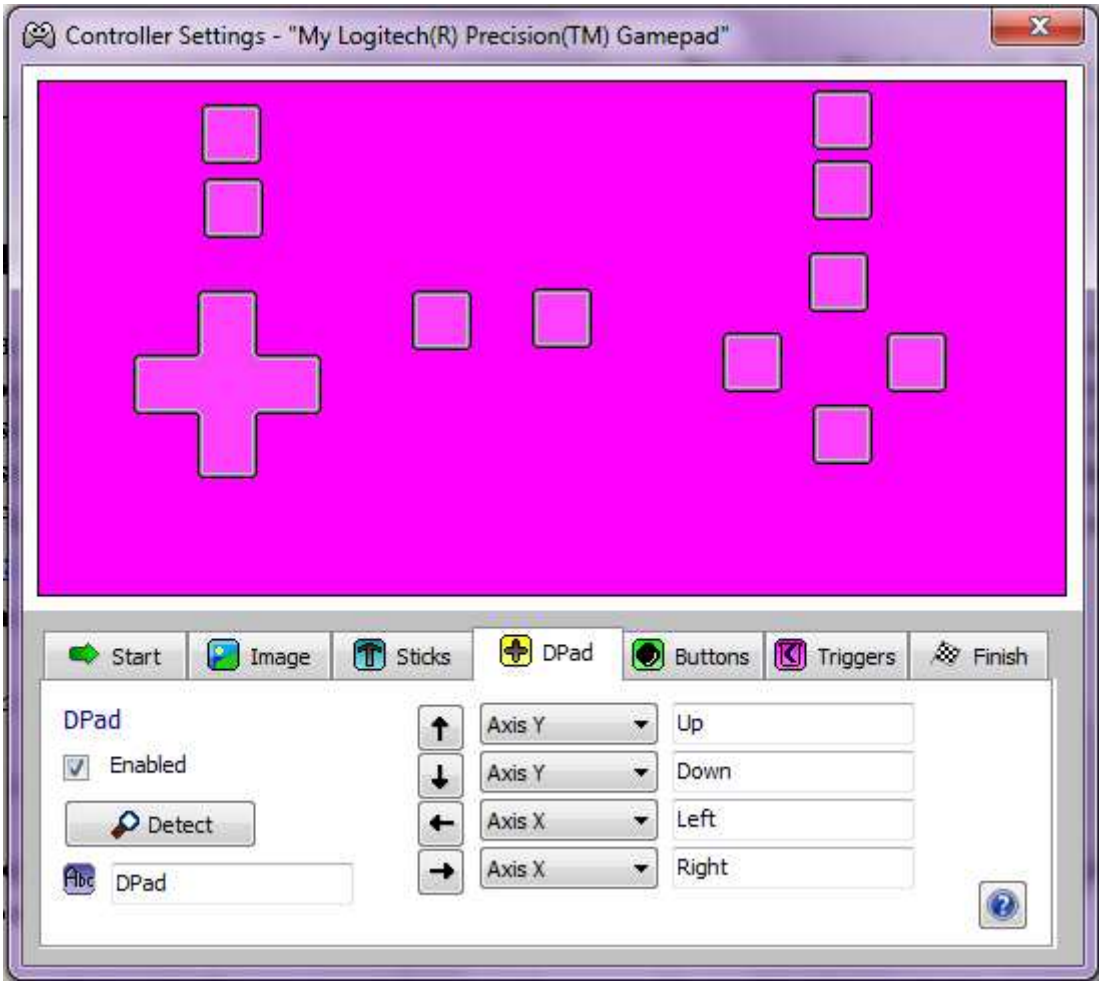



**Ilustración 141:** Configuración del joystick.

En la ventana ajustes **(ilustración 141)** aparece ***"DPad"*** que son las crucetas y ***"Buttons"*** el resto de botones. Los ***"Sticks"*** son las palancas analógicas, pero como el mando elegido no tiene no se pondrán. Para poner los botones basta con ir pulsando uno a uno todos los elementos del mando, e irán apareciendo en la ventana de ajustes. Una vez pulsados todos los botones se colocan en el mismo orden que están en el mando. Cuando se tienen todos los botones colocados se hace clic sobre *Finish→Close*.

Después de cerrar la ventana de configuración aparecerá otra ventana donde te salen los botones configurados anteriormente. Haciendo clic sobre cualquier botón aparecerá un teclado virtual **(ilustración 142)** donde habrá que elegir la tecla del teclado que se desea simular al pulsar ese botón.


**Ilustración 142:** Teclado virtual del programa Xpadder.

Para este caso se ha asignado en algunos botones dos teclas del teclado. Una de ellas es el *intro* y la otra tecla la que se quiera para dar una orden. Se le asigna el intro a algunos botones para cuando se pulsen se envié el dato a través del puerto serie del programa de Arduino. Si no se le asigna el intro, cada vez que se pulse el botón aparecerá una letra o un numero, pero nunca se enviará hasta pulsar *intro.*

También se puede configurar el tipo de pulsación al pulsar un botón. Hay dos tipos de configuración "*interruptor*" o "*metralleta*". La configuración interruptor al pulsar una vez el botón se activa y no se vuelve a desactivar hasta pulsar otra vez el mismo botón. La opción metralleta mientras se tenga un botón pulsado se estará escribiendo una letra o un número constantemente con la frecuencia que se le asigne. La configuración que se ha elegido para este caso es la de metralleta, que mientras se esté pulsando el botón el robot este activo hasta que se deje de pulsar el botón. A continuación en la siguiente ilustración 143 se podrán ver las teclas asignadas en cada botón.

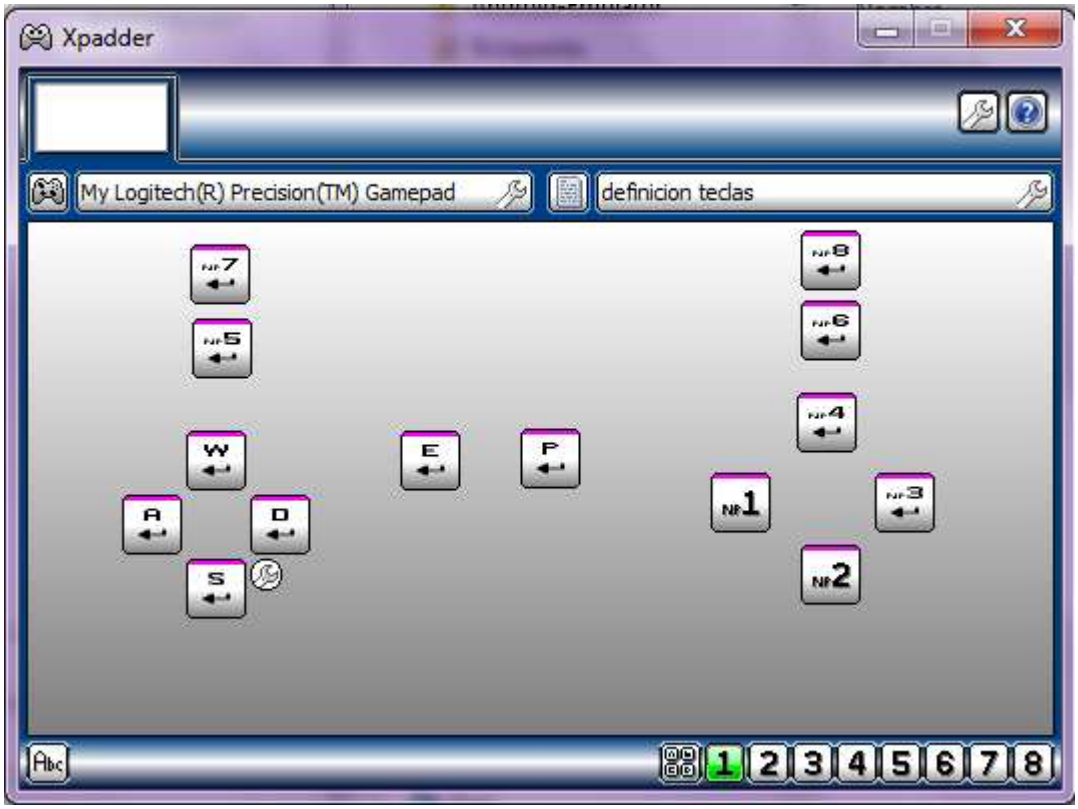



**Ilustración 143:** Asignación de botones Xpadder.

Una vez asignados los botones hay que asignar las funciones que tendrá cada botón. A continuación se explicará las funciones que se han dado para este caso.

- **Botón E:** El botón E (Encender) activa el control del joystick. Si se pulsa cualquier botón sin pulsar primero este botón el robot no hará nada. Para activar el robot se deberá pulsar primeramente el botón E.
- **Botón P:** Este botón P desactiva el robot, es el contrario que el botón E. Para desactivar el robot y dejarlo inactivo ante otras pulsaciones de otros botones se deberá pulsar el botón P. Para volver a activarlo se debe pulsar E.
- **Botón A:** El robot gira hacia la izquierda activando solo la rueda derecha en dirección hacia delante hasta que se deje de pulsar.
- **Botón B:** El robot gira hacia la derecha activando solo la rueda izquierda en dirección hacia delante hasta que se deje de pulsar.
- **Botón W:** Con este botón el robot avanza en línea recta hasta que se deje de pulsar.
- **Botón S:** Este botón hace retroceder al robot mientras se mantenga pulsado.
- **Botón 1:** El botón 1 actúa de acelerador. Mientras se tenga pulsado el botón, el robot irá cogiendo cada vez más velocidad hasta que se deje de pulsar.

- **Botón 2:** Con este botón el robot desacelera. Mientras se tenga pulsado el botón el robot irá perdiendo velocidad.
- **Botón 3:** El botón 3 detiene el coche y pone a 55 el valor de velocidad. Se ha programado así porque cuando el robot se acelera guarda su velocidad actual y cuando se pone en marcha de nuevo lo hace con la última velocidad, por lo tanto este botón resetea la velocidad a 55 para los casos que se quiera iniciar la marcha desde velocidades bajas.
- **Botón 4:** Este botón pone a 0 los parámetros que incrementan la velocidad de cada rueda individualmente *(izq=0, der=0)*.
- **Botón 5:** Con este botón se incrementa la velocidad de la rueda izquierda *(izq++)*.
- **Botón 6:** Incrementa la velocidad de la rueda derecha *(der++)*.
- **Botón 7:** Disminuye la velocidad de la rueda izquierda *(izq--)*.
- **Botón 8:** Disminuye la velocidad de la rueda derecha *(der--)*.

Los parámetros ***der*** e ***izq*** se suman a la velocidad total incrementando la velocidad de cada rueda de forma individual. Se ha programado así porque a veces el robot no avanza en línea recta, ya que una rueda gira más deprisa que otra. Con esto se consigue igualar la velocidad de giro de ambas ruedas consiguiendo que el robot avance en línea recta.

El control por Joystick construido en este apartado es muy satisfactorio. Con este control se puede controlar el robot en los casos en los que el reconocedor de voz no se pueda utilizar.

## 4.14 Puesta en marcha del robot.

Para poner en funcionamiento el robot se deberán seguir los siguientes pasos:

- Lo primero es cargar el programa de Arduino en el robot. Para cargar el programa del robot se deberá conectar la placa Arduino mediante el cable USB al ordenador. Una vez conectada la placa se elige el puerto donde se encuentra conectada la placa. Como se puede ver hay dos puertos disponibles (COM 3, COM 7) **(ilustración 144)**, el COM 3 es el puerto al que está conectado Arduino al PC mediante el cable y el puerto COM 7 es el puerto de conexión inalámbrica de Arduino y PC. Para pasar el programa al robot hay que elegir el puerto generado al

conectar la placa con el cable (COM 3 en este caso). Una vez elegido el puerto se carga el programa haciendo clic sobre  situado en la parte superior izquierda de la ventana. Es muy importante que el interruptor que desactiva el módulo APC 220 este en posición central, ya que de otra manera estaría el módulo conectado y no se podrá cargar el programa debido a un error de comunicación.


**Ilustración 144:** Puertos serie.

Una vez que se tiene el programa cargado hay que elegir el puerto COM 7 ya que la transmisión de órdenes y parámetros entre robot y PC se hará a través de la conexión inalámbrica. En el caso que se quiera hacer pruebas con el robot conectado mediante cable se dejaría el COM 3.

- Una vez hecho el procedimiento anterior hay que abrir el programa de reconocimiento de voz y grabar los patrones de las órdenes. Aunque se haya grabado anteriormente siempre es aconsejable que se vuelvan a grabar ya que al cambiar de ambiente cualquier eco o ruido puede interferir y no reconocer las grabaciones anteriores. Para grabar los patrones hay que hacer clic sobre el botón ***ingresar patrón*** y acto seguido decir la orden, una vez que salga el mensaje de patrón ingresado abrir la lista de patrones y guardar donde corresponda.

- Cuando se hayan grabado y guardado todos los patrones se pasará al encendido del robot. Primero hay que activar el módulo APC 220, para ello hay que poner en posición on (pulsar hacia atrás) el interruptor de palanca situado en la parte de atrás del robot **(ilustración 145)**.

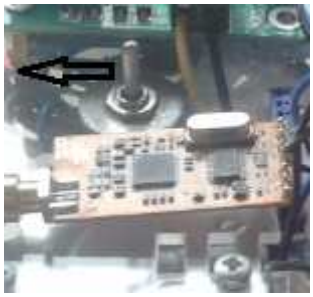


**Ilustración 145:** Interruptor activación APC220.

Después encender un segundo interruptor de color blanco, poniéndolo en la posición de encendido  . Cuando se encienda el interruptor se verán encenderse los Led de la placa Arduino. Si tras activar el interruptor no se enciende un Led de la placa de control de motores habrá que pulsar sobre el interruptor situado en ella llamado S1, una vez activo se deberá encender el Led que pone POWER.

- Si se desea controlar el robot con el joystick se deberá conectar el dispositivo y abrir el programa Xpadder.

- Para la captación y visualización de video hay que activar la aplicación **IP Webcam** instalada en el teléfono móvil y seguir los pasos descritos en el apartado 4.6.1. Cuando se hayan seguido esos pasos colocar el dispositivo móvil en el soporte del robot.

- Una vez llegado a este punto los programas abiertos quedarían de la siguiente manera en el PC **(ilustración 146)**.

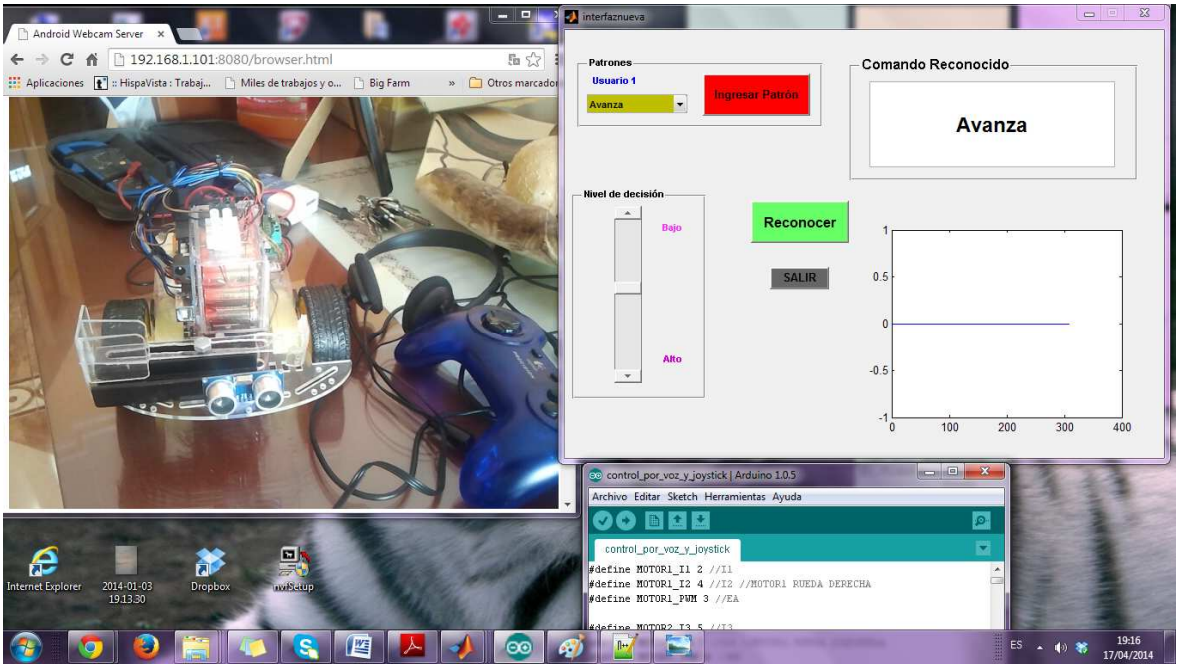



**Ilustración 146:** Programas necesarios para control del robot.

- Para comenzar a controlar el robot hay que elegir la manera con la que se desea manejar el robot (por voz o joystick). Si se desea controlar por voz hay que hacer clic sobre ***Reconocer*** y empezar a decir las órdenes que se quiera previamente definidas. Una vez terminado el control por voz pulsar sobre ***escape.*** Para controlar el robot con el joystick hay que abrir el puerto serie de Arduino  y poner el cursor y hacer clic sobre la línea de escritura del puerto **(ilustración 147)**. Una abierta la consola de Arduino y puesto para escribir, empezar a pulsar los botones del joystick para controlar el robot. **Muy importante pulsar primero el botón E,** ya que es el que activa el control del robot por joystick.

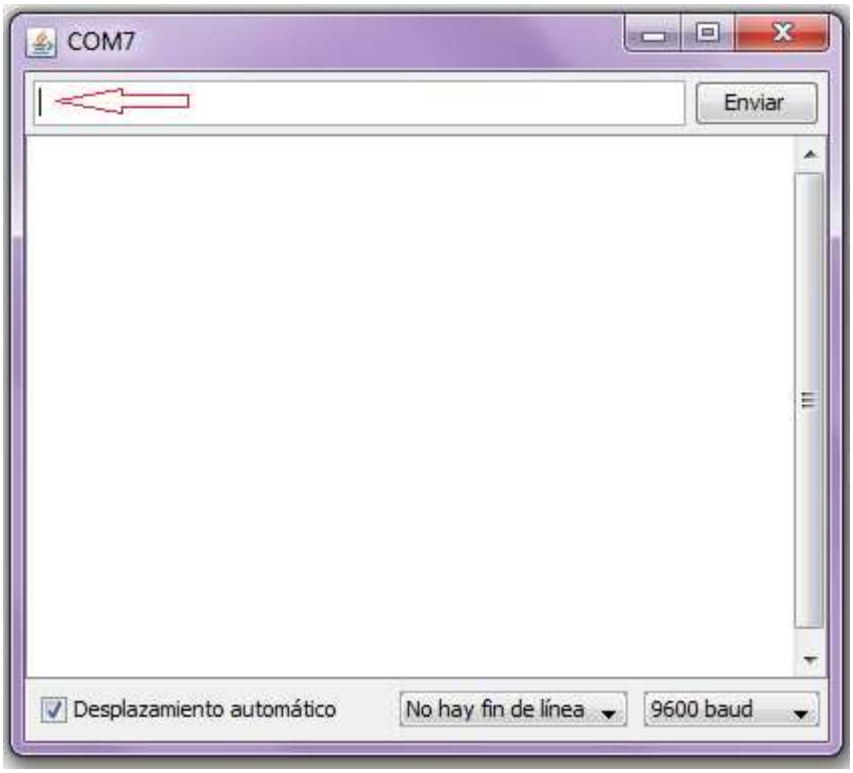



**Ilustración 147:** Consola puerto serie Arduino.

- Para la desconexión del robot hay que desactivar los interruptores encendidos anteriormente y desconectar los dispositivos del ordenador.

# CAPÍTULO 5: RESULTADOS.

# 5 RESULTADOS.

## 5.1 Resultados de la comunicación por radiofrecuencia.

Como se vio en el apartado 4.12, para establecer la comunicación entre Arduino y Matlab hicieron falta unos módulos de comunicación por radiofrecuencia. Estos módulos explicados en el apartado 4.2.5 tras unos ensayos de comunicación se obtuvieron buenos resultados, llegando a comunicarse con distancias aproximadas a 600 metros. También se probó la comunicación con el PC y robot en diferentes habitáculos y los resultados fueron satisfactorios; como por ejemplo, situando el PC en una habitación de la primera planta y el robot en la tercera planta. La comunicación es rápida y precisa aunque depende de la velocidad de procesamiento que tenga el PC utilizado.

## 5.2 Resultados de la velocidad de procesamiento.

### 5.2.1 Resultado comunicación control por voz.

La velocidad de procesamiento mediante las ordenes **avanza, para, gira izquierda, gira derecha** y **retrocede,** se ha obtenido *un retardo* entre *0.63~0.88 segundos* como se puede ver en la siguiente *Tabla 2.* Este tiempo se ha medido en el mismo instante en el que se acaba de decir la orden hasta que el robot se pone en funcionamiento.

| Orden | Tiempo de retardo, Primer intento (seg.) | Tiempo de retardo, Segundo intento (seg.) |
|---|---|---|
| **Avanza** | 0.65 | 0.69 |
| **Para** | 0.63 | 0.65 |
| **Gira Izquierda** | 0.75 | 0.8 |
| **Gira derecha** | 0.88 | 0.76 |
| **Retrocede** | 0.69 | 0.71 |

**Tabla 2:** Tiempos de retardos de comunicaciones órdenes básicas.

Para las ordenes **más deprisa, más despacio** se ha obtenido un tiempo mayor de retardo estando comprendido entre *1.33~1.65 segundos (Tabla 3),* esto es debido a la escritura y lectura en el puerto serie, ya que primeramente Matlab manda una orden a Arduino, este la recibe y manda dos valores a Matlab (distancia y velocidad) en el cual Matlab tiene que estar a la espera de que le lleguen esos datos para después procesarlos en el programa de lógica difusa para su posterior escritura en el puerto serie hacia Arduino. Estos tiempos se

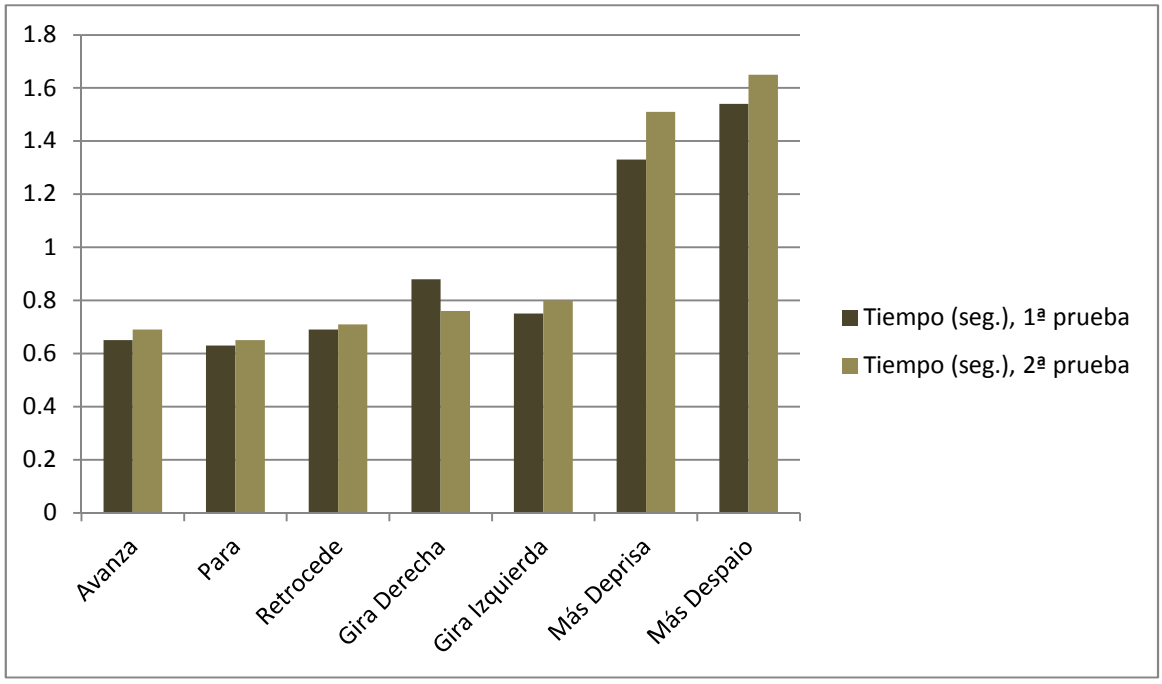
mejorarían con un ordenador mucho más potente, ya que el retardo no es debido a la comunicación, sino a la velocidad de procesamiento del ordenador.

| Orden | Tiempo de retardo, Primer intento (seg.) | Tiempo de retardo, Segundo intento (seg.) |
|---|---|---|
| **Más Deprisa** | 1.33 | 1.51 |
| **Más Despacio** | 1.54 | 1.65 |

**Tabla 3:** Tiempos de retardos de comunicaciones órdenes difusas.

Estos resultados de velocidad de procesamiento y comunicación son satisfactorios, con el inconveniente que para las ordenes *más deprisa* y *más despacio* no se puede ordenar en espacios pequeños porque para valores de velocidad mayores de 60 en *1.33~1.65* segundos el robot recorre una distancia considerable y se aproximaría mucho a los obstáculos y chocaría. Para evitar este inconveniente, como se ha descrito en apartados anteriores, se ha dotado al robot de "autonomía" para la reducción de velocidad y en el caso necesario de la detención del robot al aproximarse mucho al obstáculo.

En la siguiente ***gráfica de resultados 1*** se puede ver los tiempos de retardo para cada orden.




**Gráfica de resultados 1:** tiempos de retardo de las órdenes.

### 5.2.2 Resultado comunicación control por Joystick.

La velocidad de comunicación y procesamiento mediante el control por joystick es muy rápida, haciéndose casi instantánea. Si en algunos casos se aprecia un pequeño retardo es debido a la temporización interna del control, ya que se ha tenido que dar una pequeña temporización para evitar que el robot funcione a “trompicones” debido a la simulación (metralleta) de pulsación de teclas.

## 5.3 Resultados de movimientos.

Para los resultados de los movimientos se han hecho ensayos de todas las órdenes básicas (*avanza, para, retrocede, gira izquierda, gira derecha*). Se ordenará cinco veces la misma orden para ver la diferencia que se puede dar en cada prueba.

### 5.3.1 Resultado movimiento orden *avanza*.

Para la obtención de estos resultados se ha dicho cinco veces la orden avanza, tomando nota de si ha funcionado o no el robot, el movimiento que sigue (línea recta o desviación) y la distancia a la que se detiene el robot frente al obstáculo.

La posición inicial del robot es la siguiente **(ilustración 148)**.

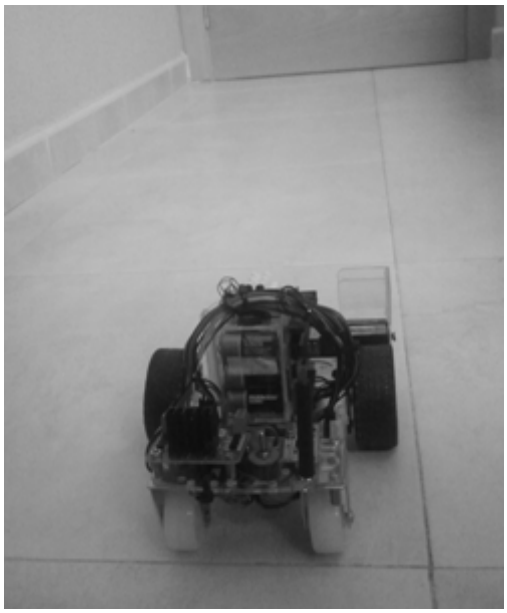
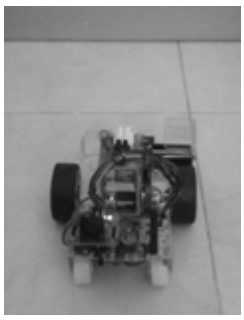
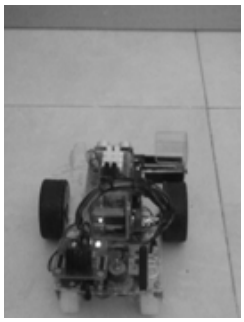


**Ilustración 148:** Posición inicial ensayo *avanza*.

Los resultados han sido los siguientes, *Tabla 4.*

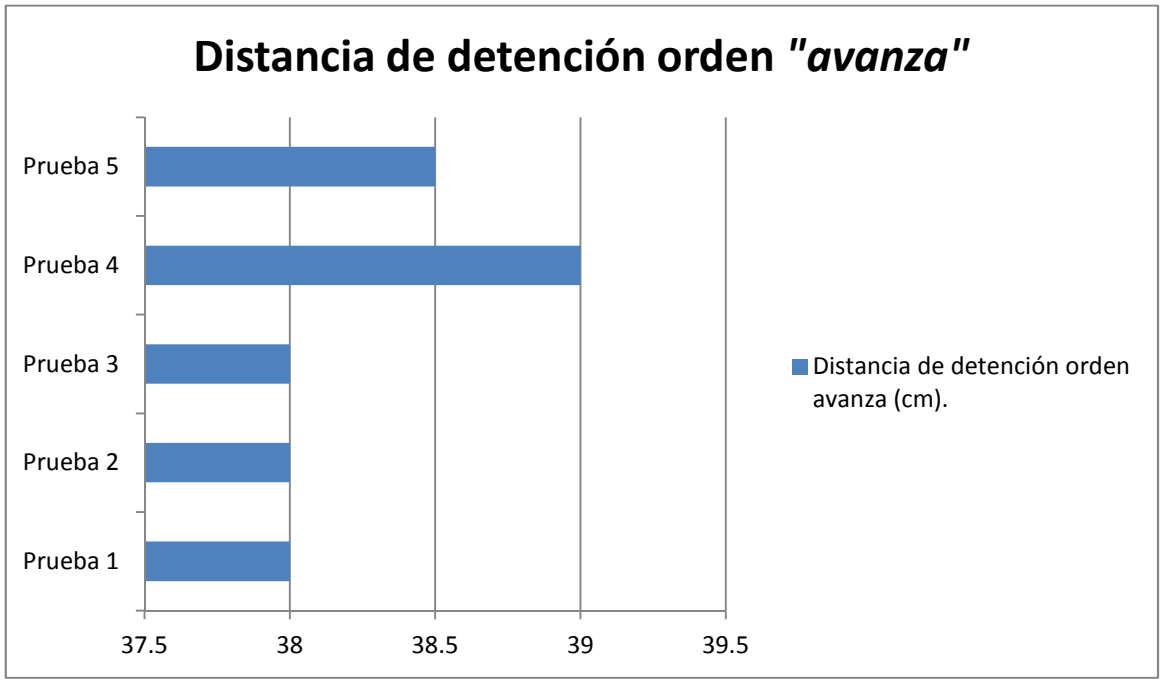
| Número de Prueba | Funcionamiento | Movimiento | Distancia al obstáculo (cm). |
|---|---|---|---|
| **Prueba 1** | Perfecto |  | 38 |
| **Prueba 2** | Perfecto |  | 38 |
| **Prueba 3, baldosa lisa.** | Perfecto |  | 38 |
| **Prueba 4** | Perfecto |  | 39 |

| | | | |
|---|---|---|---|
| **Prueba 5** | Perfecto |  | 38.5 |

**Tabla 4:** Ensayo movimiento orden *avanza.*

Como se puede ver en la *tabla 4* de los resultados obtenidos para la orden avanza se tratan de unos resultados muy buenos, ya que la distancia de detección siempre es aproximadamente igual. El funcionamiento del robot para esta orden no ha dado ningún fallo y los movimientos que ha tenido siguen prácticamente una trayectoria recta, aunque tiende a irse hacia la derecha, esto es debido, como ya se comento anteriormente, a que la salida del controlador de motores no es perfecta, entregando más potencia a una salida que a otra.

En la siguiente ***gráfica de resultados 2*** se puede ver la distancia de detención del robot al obstáculo.




**Gráfica de resultados 2:** Distancia de detención al obstáculo orden *avanza.*

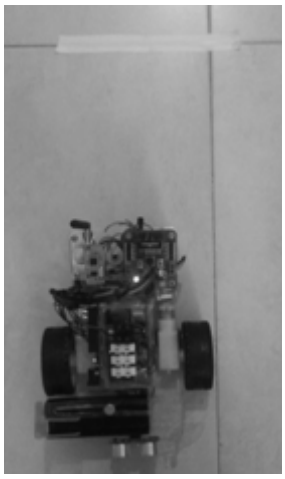
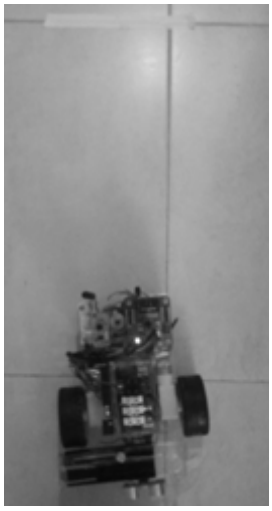
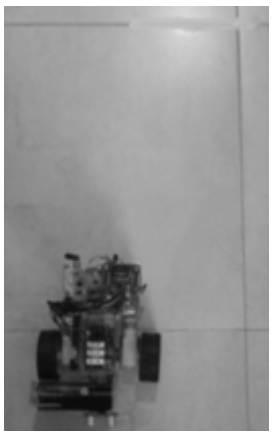
### 5.3.2 Resultado movimiento orden *Para*.

Para la obtención de estos resultados se harán cinco pruebas diciendo la orden *para* siempre desde la misma referencia, para después tomar nota de la distancia que recorre el robot hasta que se detiene.

Se colocará el robot a una distancia 1.5 metros de la marca de detención **(ilustración 149)**.

**Ilustración 149:** Posición inicial ensayo *Para*.

Los resultados obtenidos son los siguientes (*Tabla 5)*:

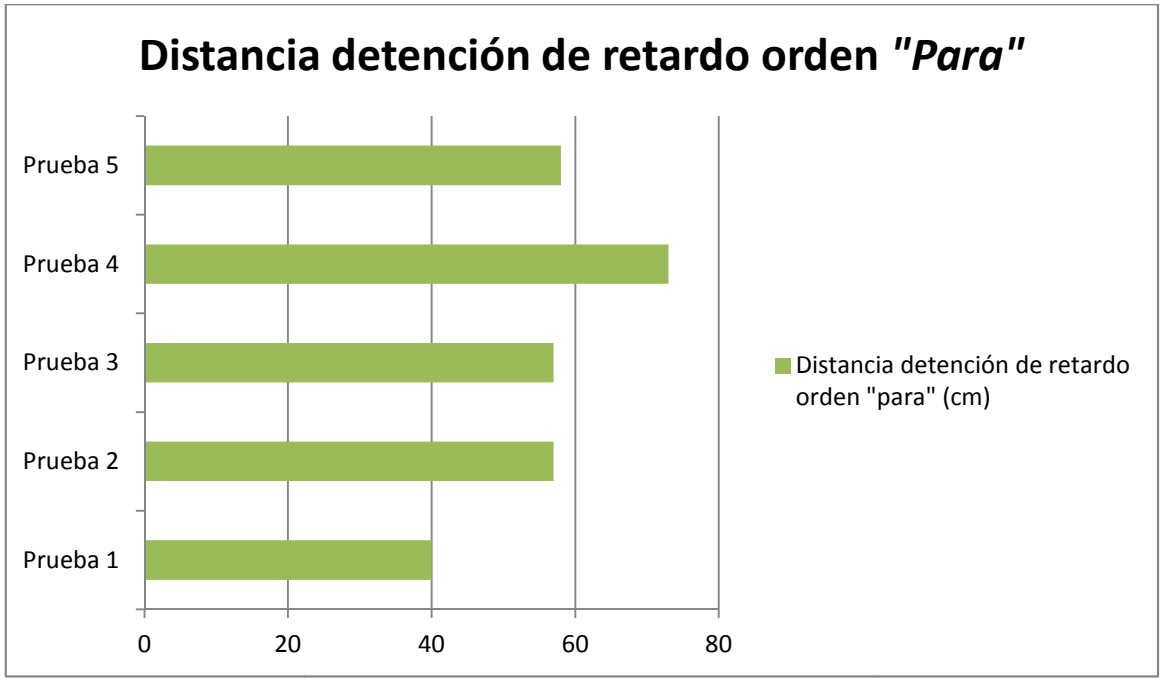
| Número de Prueba | Funcionamiento | Movimiento | Distancia recorrida (cm). |
|---|---|---|---|
| **Prueba 1** | Perfecto |  | 40 |
| **Prueba 2** | Perfecto |  | 57 |
| **Prueba 3** | Perfecto |  | 57 |

| | | | |
|---|---|---|---|
| **Prueba 4** | Perfecto |  | 73 |
| **Prueba 5** | Perfecto |  | 58 |

**Tabla 5:** Ensayo movimiento orden *Para*.

Como se puede apreciar en la *tabla 5* de resultados para la orden *"para"* desde que se ordena la orden hasta que el robot se detiene pasa una cierta distancia, esto es debido al tiempo de procesamiento y comunicación. Viendo estos resultados se ha llegado a la conclusión que se debe ordenar al robot que se pare antes de tiempo, o sea, hay que percibirse con la decisión de parar ya que esta orden tiene un cierto retardo y el robot recorre una cierta comprendida entre 40 y 75 cm aproximadamente.

En la siguiente ***gráfica de resultados 3*** se puede ver la distancia de detención de retardo del robot para la orden *"para"*.




**Gráfica de resultados 3:** Distancia de detención de retardo orden para.

### 5.3.3 Resultados movimiento orden *gira izquierda*.

Para la obtención de estos resultados se harán cinco ensayos para cada grado de giro. Habrá dos grados de giros, uno de 45º y otro de 90º. Primeramente se tomará la medida del resultado de 45º y posteriormente se ordenará otra vez la orden gira y se tomará nota de la respuesta para 90º.

La posición inicial será 0º **(ilustración 150)**.

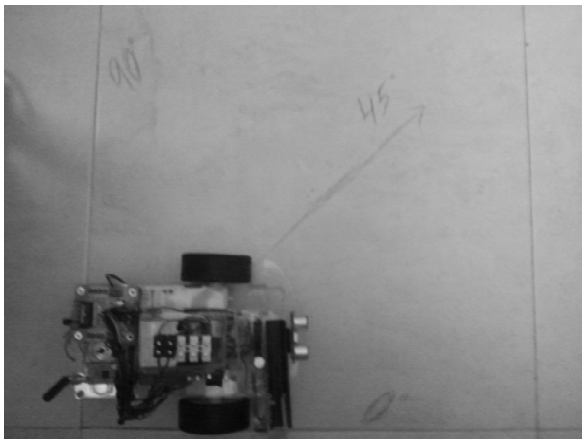
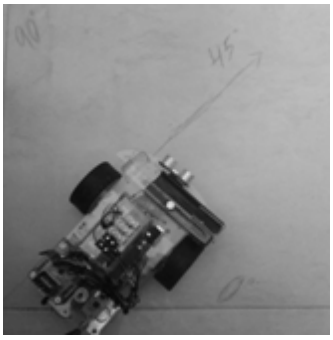
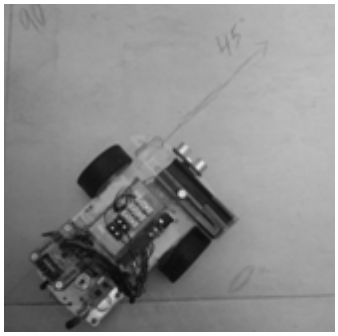
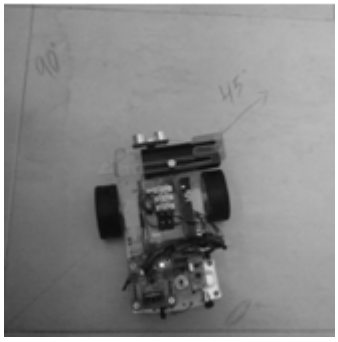
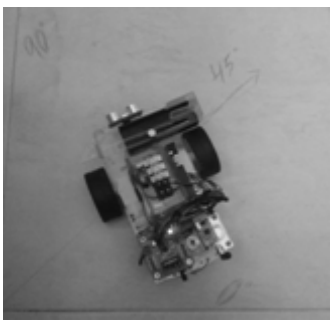



**Ilustración 150:** Posición inicial orden *gira izquierda*.

Los resultados obtenidos tras este ensayo han sido los siguientes:

| Número de Prueba | Funcionamiento | Movimiento para 45º ó 90º | 45º | 90º |
|---|---|---|---|---|
| **Prueba 1** | Perfecto |  | Aprox. 44º | Aprox. 88º |
| **Prueba 2** | Perfecto |  | Aprox. 45º | Aprox. 90º |
| **Prueba 3** | Perfecto |  | Aprox. 46º | Aprox. 92º |
| **Prueba 4** | Perfecto |  | Aprox. 48º | Aprox. 100º |

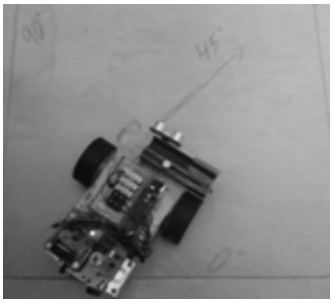
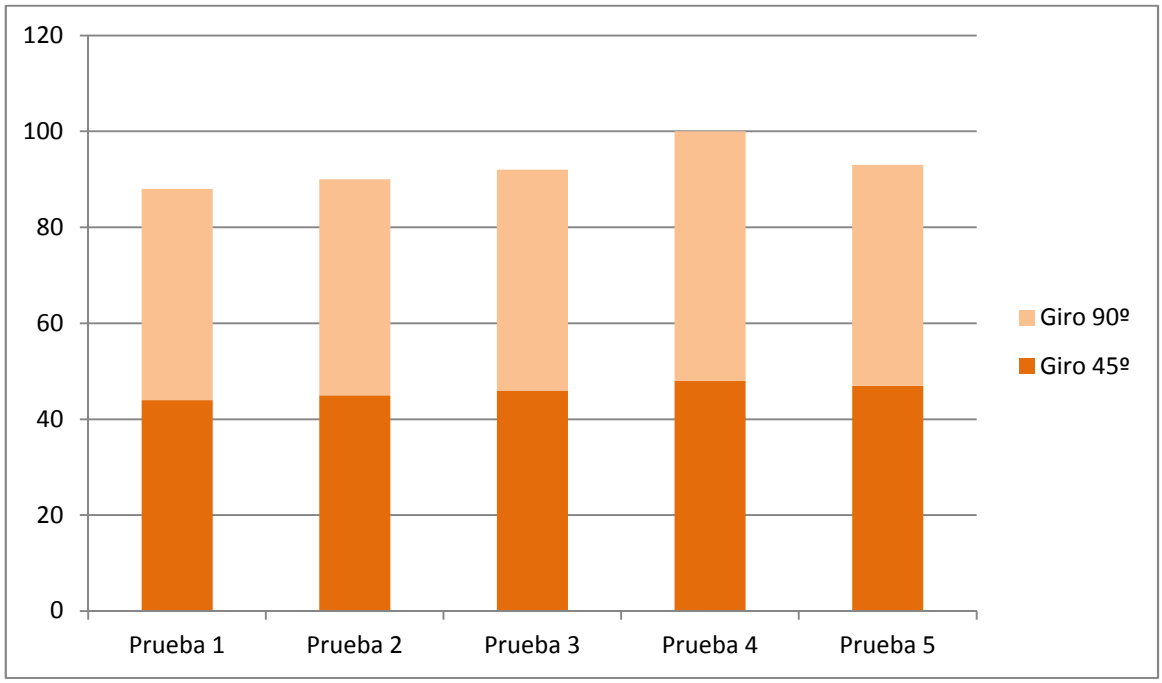
| | | | | |
|---|---|---|---|---|
| **Prueba 5** | Perfecto |  | Aprox. 47º | Aprox. 93º |

**Tabla 6:** Ensayo movimiento orden *gira izquierda*.

Como se puede ver en la *tabla 6* de resultados para la orden gira izquierda está próxima a 45º y si se ordena seguidamente la misma orden para conseguir 90º los resultados obtenidos son aproximadamente 90º. Por lo tanto esta orden se satisface.

En la siguiente ***gráfica de resultados 4*** se puede ver los giros realizados por el robot en grados para la orden *"gira izquierda"* para girar 45º y 90º.




**Gráfica de resultados 4:** Giros izquierda de 45 y 90 grados.

### 5.3.4 Resultados movimiento orden *gira derecha*.

Para la obtención de estos resultados se harán cinco ensayos para cada grado de giro. Habrá dos grados de giros, uno de 45º y otro de 90º. Primeramente se tomará la medida del

resultado de 45º y posteriormente se ordenará otra vez la orden gira y se tomará nota de la respuesta para 90º.

La posición inicial será 0º **(ilustración 151)**.

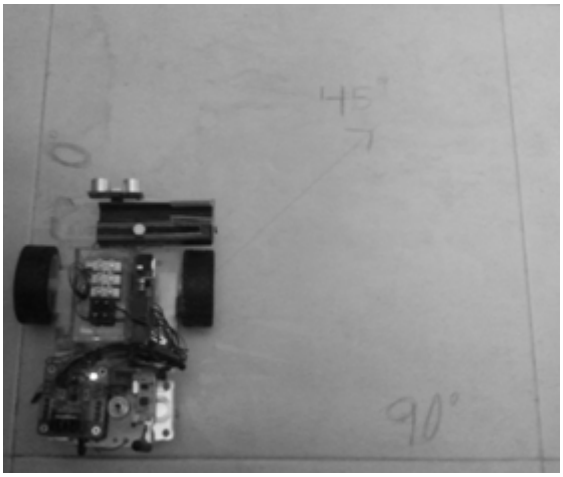
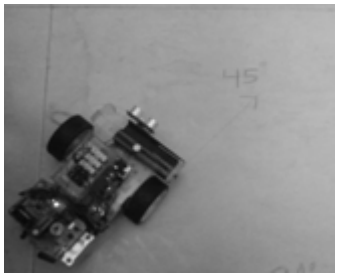
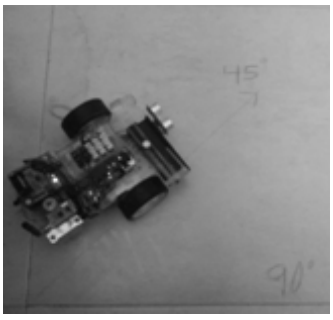



**Ilustración 151:** Posición inicial orden *gira derecha.*

Los resultados obtenidos tras este ensayo han sido los siguientes *(Tabla 7)*:

| **Número de Prueba** | **Funcionamiento** | **Movimiento para 45º** | **45º** | **90º** |
|---|---|---|---|---|
| **Prueba 1** | Perfecto |  | Aprox. 45º | Aprox. 90º |
| **Prueba 2** | Perfecto |  | Aprox. 46º | Aprox. 93º |

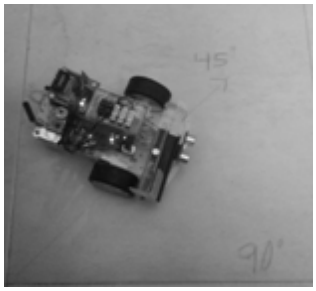
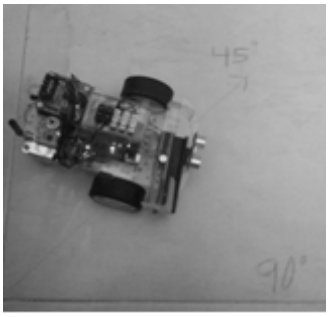
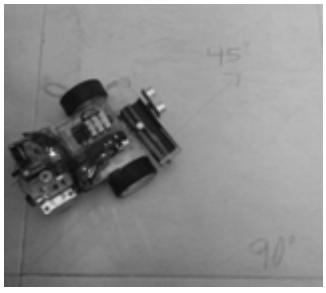
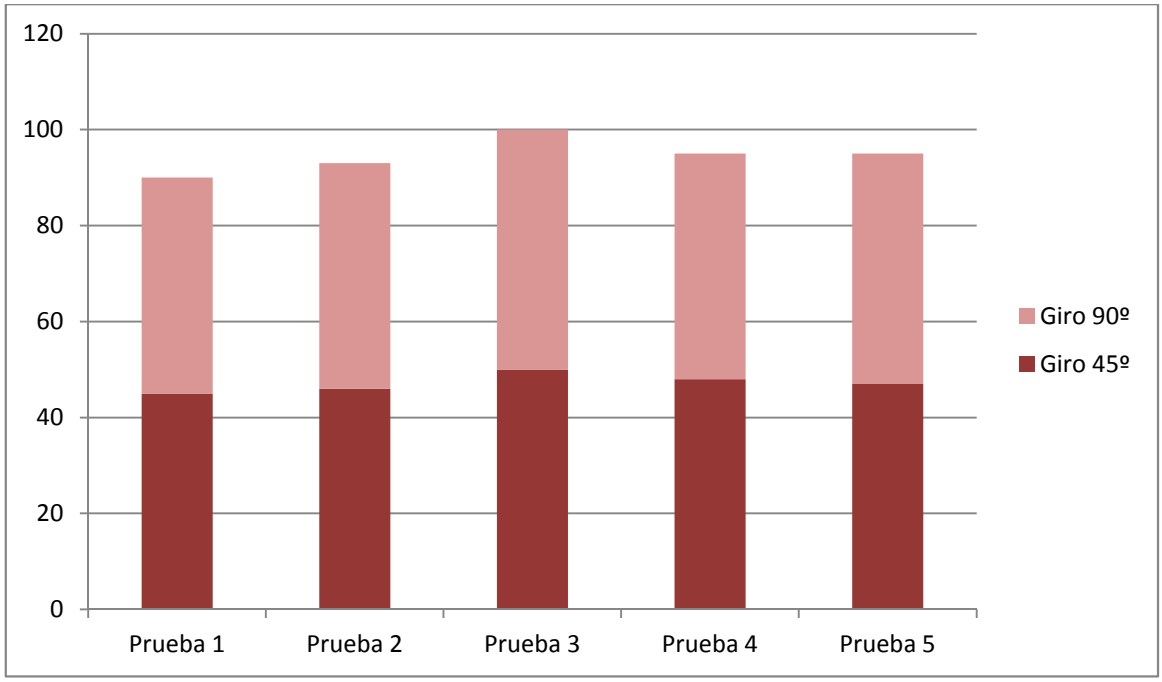
| | | | | |
|---|---|---|---|---|
| **Prueba 3** | Perfecto |  | Aprox. 50º | Aprox. 100º |
| **Prueba 4** | Perfecto |  | Aprox. 48º | Aprox. 95º |
| **Prueba 5** | Perfecto |  | Aprox. 47º | Aprox. 95º |

**Tabla 7:** Ensayo movimiento orden gira derecha.

Como se puede ver en la *tabla 7* de resultados para la orden gira derecha está próxima a 45º y si se ordena seguidamente la misma orden para conseguir 90º los resultados obtenidos son aproximadamente 90º. Por lo tanto esta orden se satisface.

En la siguiente ***gráfica de resultados 5*** se puede ver los giros realizados por el robot en grados para la orden "*gira derecha*" para girar 45º y 90º.




**Gráfica de resultados 5:** Giros *derecha para 45 y 90 grados.*

### 5.3.5 Resultado movimiento orden *retrocede.*

Para obtener los resultados de esta orden se procederá hacer cinco pruebas diciendo retrocede. Se deberá conseguir un desplazamiento aproximadamente de 20 cm.

La posición inicial del robot será **(ilustración 152)**:

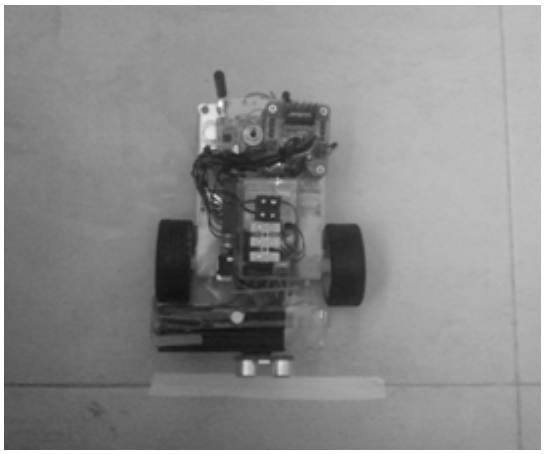


**Ilustración 152:** Posición inicial orden *retrocede.*

Los resultados obtenidos son los siguientes (*Tabla 8)*:

| **Número de Prueba** | **Funcionamiento** | **Movimiento** | **Distancia retrocedida (cm).** |
|---|---|---|---|
| **Prueba 1** | Perfecto |  | 23 |
| **Prueba 2** | Perfecto |  | 23.5 |
| **Prueba 3** | Perfecto |  | 25 |
| **Prueba 4** | Perfecto |  | 25 |

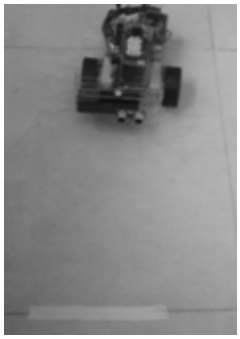
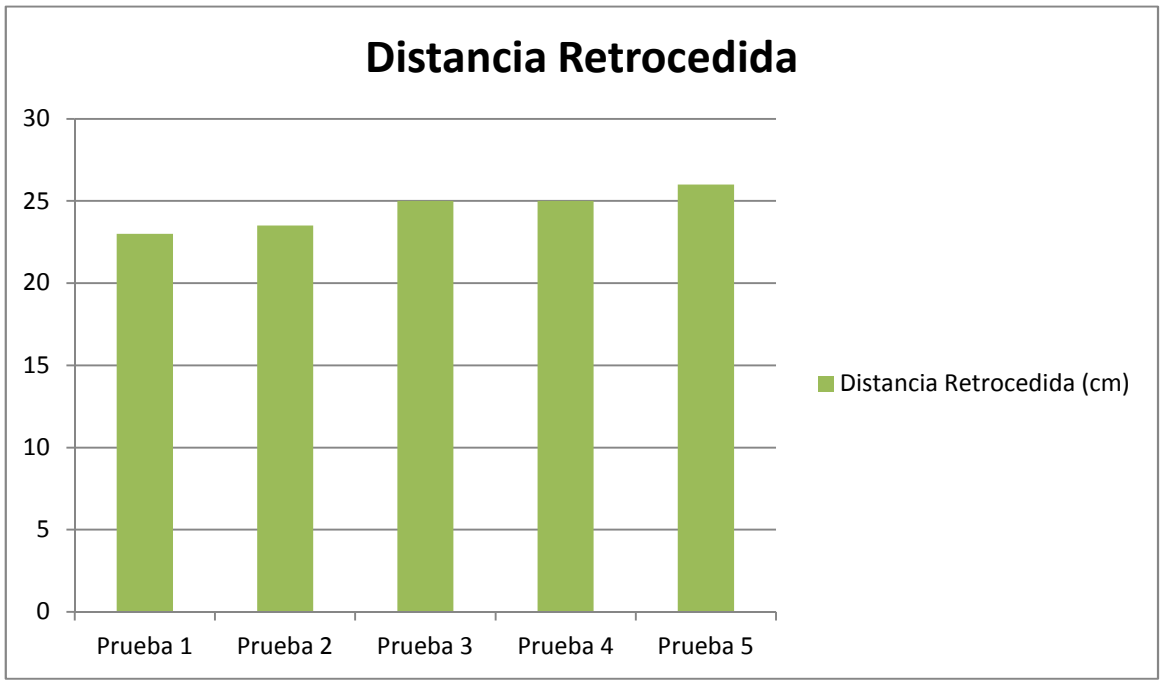
| | | | |
|---|---|---|---|
| **Prueba 5** | Perfecto |  | 26 |

**Tabla 8:** Ensayo movimiento orden *retrocede*.

Los resultados obtenidos *(Tabla 8)* han sido buenos, ya que la distancia que retrocede el robot está próxima a los 20 cm.

En la siguiente ***gráfica de resultados 6*** se puede ver la distancia retrocedida para la orden *"retrocede"*.




**Gráfica de resultados 6:** Distancia retrocedida orden *retrocede.*

## 5.4 Resultados del control difuso.

Para ver el funcionamiento del control difuso de las órdenes *más deprisa* y *más despacio* se harán de forma manual desde el *toolbox fuzzy* ya creado en el apartado 4.10, introduciendo los valores deseados para cada una de las variables.

Para acceder a la ventana de *rule viewer* ***(ilustración 153)***, sobre la ventana principal de la herramienta *toolbox fuzzy* hacer clic sobre **View→Rules** y a continuación aparecerá la siguiente ventana.

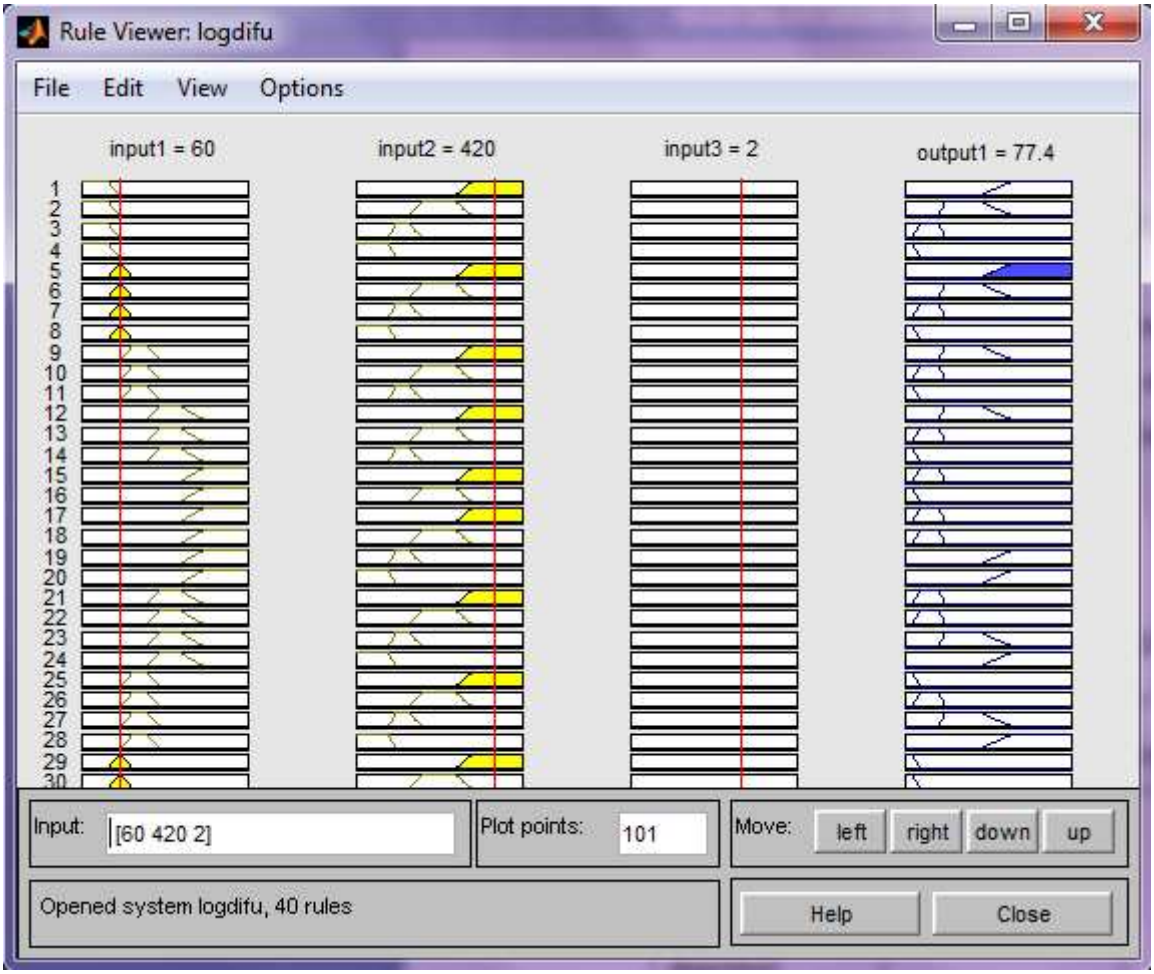



**Ilustración 153:** Programa toolbox fuzzy *rule viewer.*

Para simular de forma manual el funcionamiento, en el programa hay que poner los valores deseados sobre el recuadro *input* de la ilustración anterior. El resultado aparecerá en la parte derecha superior (*output1).*

### 5.4.1 Resultado control difuso de la orden *más deprisa.*

Se harán once pruebas con la orden *más deprisa,* cada una con diferentes distancias partiendo de "velocidad" igual a 60. Cada distancia del robot al obstáculo se pondrá de dicha forma que englobe todos los casos en la que se definió la variable *distancia* en el apartado 4.10.

| Número de Prueba (más deprisa) | Distancia (cm) | Velocidad Antigua | Incremento (+) | Velocidad Antigua + Incremento |
|---|---|---|---|---|
| 1 | 50 (Cerca) | 60 | 3.63 | 63.63 |
| 2 | 105 (Cerca-Medio) | 60 | 8.62 | 68.62 |
| 3 | 115 (Cerca-Medio) | 60 | 13.6 | 73.6 |
| 4 | 140 (Medio) | 60 | 15 | 75 |
| 5 | 170 (Medio-Lejos) | 60 | 27.5 | 87.5 |
| 6 | 190 (Medio-Lejos) | 60 | 36.7 | 96.7 |
| 7 | 250 (Lejos) | 60 | 39.2 | 99.2 |
| 8 | 315 (Lejos-MuyLejos) | 60 | 53 | 113 |
| 9 | 335 (Lejos-MuyLejos) | 60 | 68 | 128 |
| 10 | 420 (MuyLejos) | 60 | 77.4 | 137.4 |

**Tabla 9:** Ensayo control difuso orden *más deprisa.*

Como se puede apreciar en la ***tabla 9***, para una misma velocidad, cuanto mayor sea la distancia entre el obstáculo y el robot, mayor será el incremento de velocidad resultante, para posteriormente sumárselo a la velocidad antigua.

Un segundo ensayo *(tabla 10)* para esta misma orden (*más deprisa)* será tener el robot a una distancia siempre igual pero variando su velocidad. Se harán dos pruebas para cada caso de distancia con diferentes velocidades cada una. A continuación se muestra los resultados obtenidos:

| **Número de Prueba (más deprisa)** | **Distancia (cm)** | **Velocidad Antigua** | **Incremento (+)** | **Velocidad Antigua + Incremento** |
|---|---|---|---|---|
| 1 | 50 (Cerca) | 40 (Muy Lento) | 3.63 | 43.63 |
| 2 | 50 (Cerca) | 140 (Rápido) | 1 | 141 |
| 3 | 140 (Medio) | 40 | 15 | 55 |
| 4 | 140 (Medio) | 140 | 3.63 | 143.63 |
| 5 | 250 (Lejos) | 40 | 39.2 | 79.2 |
| 6 | 250 (Lejos) | 110 (Medio-Rápido) | 15 | 125 |
| 7 | 420 (MuyLejos) | 40 | 77.4 | 117.4 |
| 8 | 420 (MuyLejos) | 117.4 (Medio-Rápido) | 39.6 | 157 |

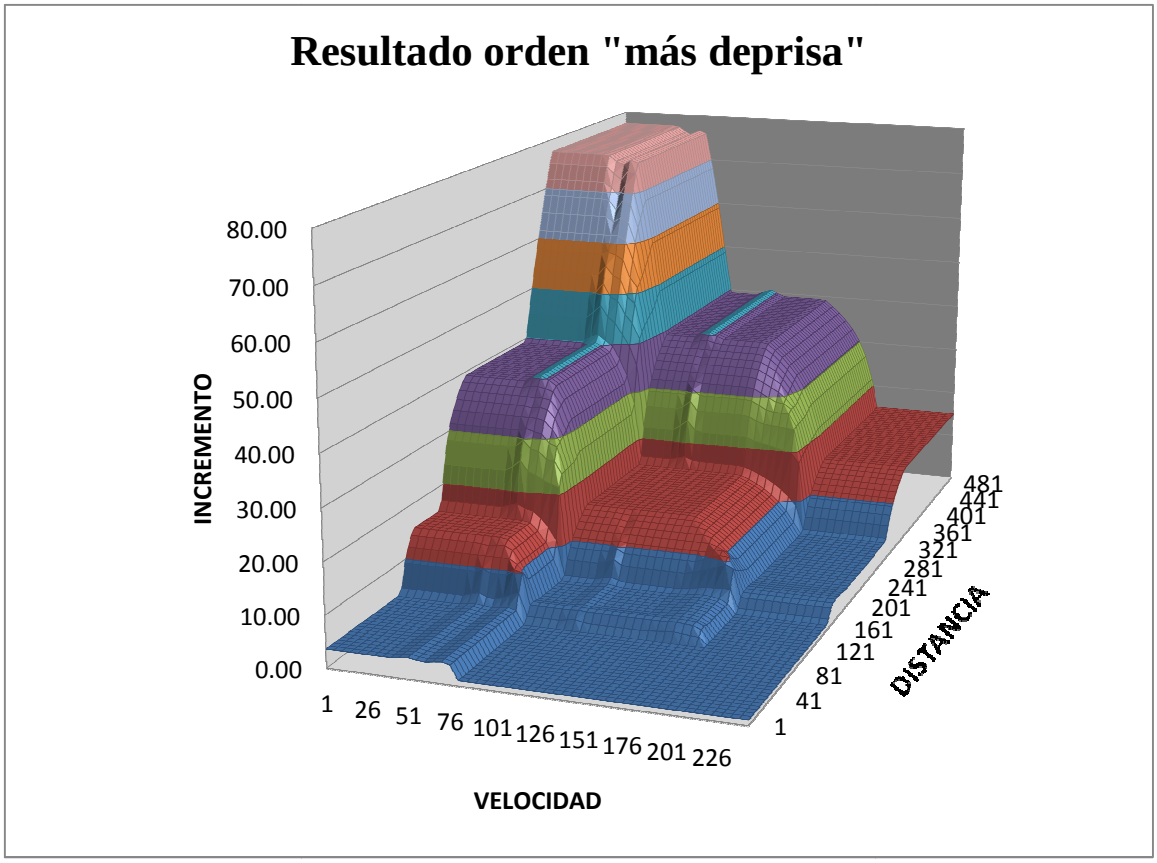
| | | | | |
|---|---|---|---|---|
| 9 | 420 (MuyLejos) | 157 Rápido-MuyRápido | 37.5 | 194.5 |
| 10 | 250 (MuyLejos) | 200 (Rápido) | 3.63 | 203.63 |

**Tabla 10:** Segundo ensayo lógica difusa orden *más deprisa*.

Como se puede apreciar en los resultados de la ***tabla 10***, aunque la distancia sea la misma, la respuesta (incremento) será función también de la velocidad del robot. Para la orden *más deprisa* cuanto mayor sea la velocidad el incremento será menor. En la prueba número diez se puede ver que aunque el robot se encuentre a 250 cm del obstáculo a una "velocidad" de 200 el resultado del incremento será muy poco, ya que se encuentra muy cerca del límite de la "velocidad máxima" (255).

Los resultados obtenidos (*tabla 9 y 10)* para la orden *más deprisa* han sido muy satisfactorios.

En la siguiente ***gráfica de resultados 7*** se puede ver la respuesta de incremento que se pueden dar para la orden *"más deprisa"* para todos los casos que se pueden dar.




**Gráfica de resultados 7:** Resultados orden *"más deprisa"*.

### 5.4.2 Resultado control difuso de la orden *más despacio*.

Se harán once pruebas con la orden *más despacio*, cada una con diferentes distancias partiendo de "velocidad" igual a 125. Cada distancia del robot al obstáculo se pondrá de dicha forma que englobe todos los casos en la que se definió la variable *distancia* en el apartado 4.10.

| **Número de Prueba (más despacio)** | **Distancia (cm)** | **Velocidad Antigua** | **Incremento (-)** | **Velocidad Antigua + Incremento** |
|---|---|---|---|---|
| 1 | 80 (Cerca) | 125 (Rápido) | 77.4 | 47.6 |

| | | | | |
|---|---|---|---|---|
| 2 | 105 (Cerca-Medio) | 125 | 69.7 | 55.3 |
| 3 | 115 (Cerca-Medio) | 125 | 50.8 | 74.2 |
| 4 | 140 (Medio) | 125 | 39.2 | 85.8 |
| 5 | 170 (Medio-Lejos) | 125 | 36.7 | 88.3 |
| 6 | 190 (Medio-Lejos) | 125 | 27.5 | 97.5 |
| 7 | 250 (Lejos) | 125 | 15 | 110 |
| 8 | 315 (Lejos-MuyLejos) | 125 | 15 | 110 |
| 9 | 335 (Lejos-MuyLejos) | 125 | 15 | 110 |
| 10 | 420 (MuyLejos) | 125 | 15 | 110 |

**Tabla 11:** Ensayo control difuso orden *más despacio.*

Como se puede apreciar en la ***tabla 11***, para una misma velocidad, cuanto menor sea la distancia entre el obstáculo y el robot, mayor será el incremento de velocidad resultante para posteriormente restárselo a la velocidad antigua. En los cuatro últimos casos se puede ver que el incremento es siempre 15, esto es así ya que se quiere como resultado que cuando se ordene la orden *más despacio* el robot disminuya su velocidad, por tanto para estos cuatro casos se ha querido que lo haga de la misma forma, aunque a veces no será la respuesta igual debido a la velocidad que lleve el robot.

Un segundo ensayo *(tabla 12)* para esta misma orden (*más despacio)* será tener el robot a una distancia siempre igual pero variando su velocidad. Se harán dos pruebas para cada caso de distancias, con diferentes velocidades cada una. A continuación se muestra los resultados obtenidos:

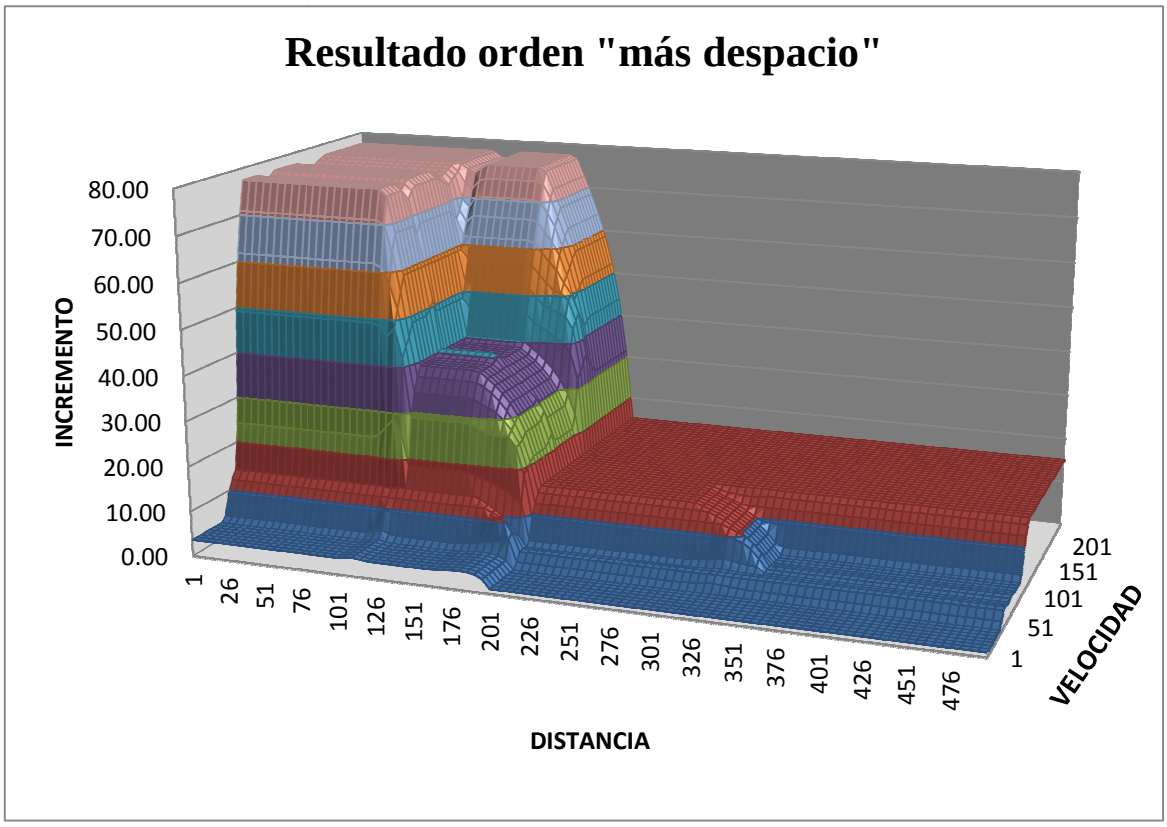
| **Número de Prueba (más despacio)** | **Distancia (cm)** | **Velocidad Antigua** | **Incremento (-)** | **Velocidad Antigua + Incremento** |
|---|---|---|---|---|
| 1 | 80 (Cerca) | 40 (Muy Lento) | 3.63 | 36.37 |
| 2 | 80 (Cerca) | 160 (Rápido-MuyRápido) | 76.3 | 83.7 |
| 3 | 140 (Medio) | 160 | 50.8 | 109.2 |
| 4 | 100 (Medio) | 109.2 | 75.4 | 33.8 |
| 5 | 250 (Lejos) | 40 | 1 | 39 |
| 6 | 250 (Lejos) | 200 (MuyRápido) | 15 | 185 |
| 7 | 200 (Lejos) | 100 | 15 | 85 |
| 8 | 300 (MuyLejos) | 200 | 15 | 185 |
| 9 | 420 (MuyLejos) | 40 | 1 | 39 |
| 10 | 420 (MuyLejos) | 200 | 15 | 185 |

**Tabla 12:** Segundo ensayo lógica difusa orden *más despacio.*

Como se puede apreciar en los resultados de la ***tabla 12***, aunque la distancia sea la misma, la respuesta (incremento) será función también de la velocidad del robot. Para la orden *más despacio* cuanto mayor sea la velocidad el incremento será mayor. En la prueba número cinco y nueve se puede ver que aunque el robot se encuentre a 250 ó 420 cm del obstáculo a una "velocidad" de 40, el resultado de incremento será muy poco, ya que se encuentra muy cerca del límite de velocidad mínimo (0).

Los resultados obtenidos *(tabla 11 y 12)* para la orden *más despacio* han sido muy satisfactorios.

En la siguiente ***gráfica de resultados 8*** se puede ver la respuesta de incremento que se pueden dar para la orden *"más despacio"* para todos los casos que se pueden dar.




**Gráfica de resultados 8:** Resultados orden *"más despacio".*

## 5.5 Resultados del reconocimiento de voz.

Para ver los resultados obtenidos por el programa de reconocimiento de voz se llevará a cabo diez veces cada orden, para sacar el porcentaje de error del sistema. Se harán tres pruebas en distintos lugares (silencioso, televisión encendida con volumen normal y al

máximo y por último lugar en la calle). Para cada uno de los lugares se grabarán de nuevo los patrones para un mejor reconocimiento. Para el ensayo del exterior, al principio no se obtuvieron buenos resultados debido a la distorsión que produce el aire, para evitar esto se puso la mano delante de la boca y el micrófono, así el aire no choca contra el micrófono.

### 5.5.1 Resultado reconocimiento orden *avanza*.

Para ver los resultados de reconocimiento sobre la orden *avanza* se dirá diez veces y se anotarán las veces que ha sido reconocida. Así con cada uno de los lugares anteriormente citados.

Primero se guardará el patrón de la orden avanza obteniéndose la siguiente gráfica de sonido **(ilustración 154)**:

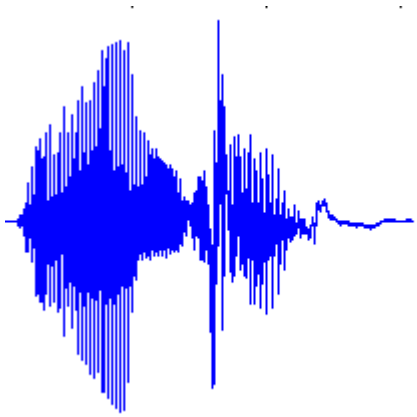


**Ilustración 154:** Gráfica patrón *avanza.*

La gráfica de la orden *avanza* reconocida tiene la siguiente forma **(ilustración 155)**:

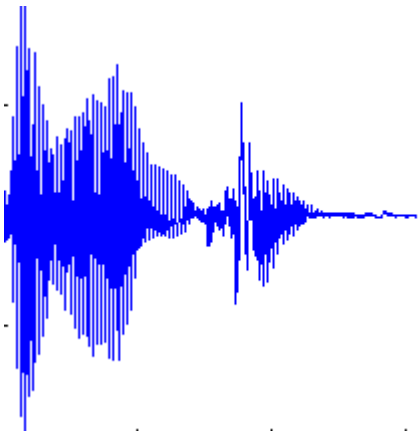


**Ilustración 155:** Gráfica orden *avanza* reconocida.

Como se puede ver en las dos gráficas tienen una cierta similitud. Una vez vista la forma que tiene la señal *avanza* se pasa a las pruebas obteniendo los siguientes resultados ***(tabla 13)***:

| **Número de veces orden *avanza*** | **Lugar silencioso** | **Con TV encendida** | **Volumen TV al máximo.** | **Exterior con aire** |
|---|---|---|---|---|
| **1** | Reconocido | Reconocido | Reconocido | Reconocido |
| **2** | Reconocido | Reconocido | Reconocido | Reconocido |
| **3** | Reconocido | Reconocido | Reconocido | Reconocido |
| **4** | Reconocido | Reconocido | Reconocido | Reconocido |
| **5** | Reconocido | Reconocido | Reconocido | Reconocido |
| **6** | Reconocido | Reconocido | Reconocido | Reconocido |
| **7** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **8** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **9** | Reconocido | Reconocido | Reconocido | Reconocido |
| **10** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **Resultado** | **100%** | **100%** | **70%** | **100%** |

**Tabla 13:** Resultados reconocimiento orden *avanza.*

Para la prueba de reconocimiento de la orden *avanza* en un lugar silencioso y en el mismo lugar con la TV encendida con volumen normal se ha obtenido el 100%. Con la televisión encendida y el volumen al máximo se ha obtenido un 70% de reconocimiento de la orden, ya que al estar el volumen tan alto supera el umbral de disparo del reconocedor y el ruido

de la TV activa el reconocedor, pero no reconoce ningún patrón, pero sale patrón desconocido. Para el ensayo en el exterior se obtuvieron buenos resultados (100%).

### 5.5.2 Resultado reconocimiento orden *para*.

Para ver los resultados de reconocimiento sobre la orden *para* se dirá diez veces y se anotarán las veces que ha sido reconocida. Así con cada uno de los lugares anteriormente citados.

Primero se guardará el patrón de la orden *para*, obteniéndose la siguiente gráfica de sonido **(ilustración 156)**:

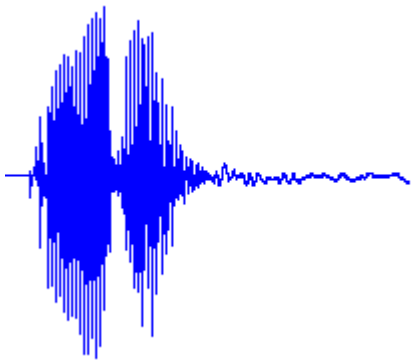


**Ilustración 156:** Gráfica patrón *para.*

La gráfica de la orden *para* reconocida tiene la siguiente forma **(ilustración 157)**:

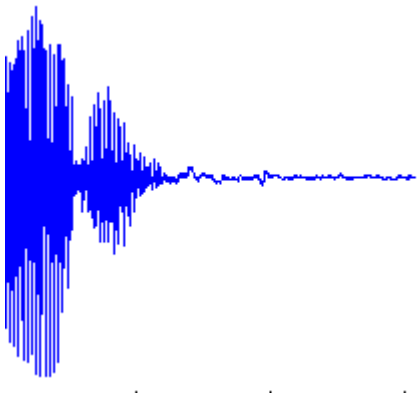


**Ilustración 157:** Gráfica orden *para* reconocida.

Como se puede ver en las dos gráficas tienen una cierta similitud. Una vez vista la forma que tiene la señal *Para* se pasa a las pruebas obteniendo los siguientes resultados ***(Tabla 14)***:

| **Número de veces orden *para*** | **Lugar silencioso** | **Con TV encendida** | **Volumen TV al máximo** | **Exterior** |
|---|---|---|---|---|
| **1** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **2** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **3** | Reconocido | Reconocido | No Reconocido | No Reconocido |
| **4** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **5** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **6** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **7** | Reconocido | Reconocido | Reconocido | Reconocido |
| **8** | Reconocido | Reconocido | Reconocido | No Reconocido |

| | | | | |
|---|---|---|---|---|
| **9** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **10** | Reconocido | Reconocido | Reconocido | Reconocido |
| **Resultado** | **100%** | **100%** | **30%** | **80%** |

**Tabla 14:** Resultado reconocimiento orden *Para.*

Para la prueba de reconocimiento de la orden *para*, en un lugar silencioso y en el mismo lugar con la TV encendida con volumen normal se ha obtenido el 100%. Con la televisión encendida y el volumen al máximo se ha obtenido un 30% de reconocimiento de la orden, ya que al estar el volumen tan alto supera el umbral de disparo del reconocedor y el ruido de la TV activa el reconocedor, pero no reconoce ningún patrón, pero sale patrón desconocido. Para el ensayo en el exterior se obtuvieron buenos resultados (80%).

### 5.5.3 Resultado reconocimiento orden *retrocede.*

Para ver los resultados de reconocimiento sobre la orden *retrocede* se dirá diez veces y se anotarán las veces que ha sido reconocida. Así con cada uno de los lugares anteriormente citados.

Primero se guardará el patrón de la orden *retrocede* obteniéndose la siguiente gráfica de sonido **(ilustración 158)**:

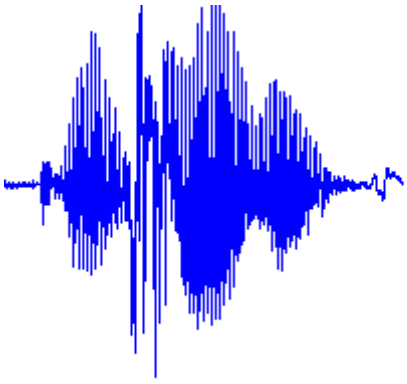


**Ilustración 158:** Gráfica patrón *retrocede.*

La gráfica de la orden *retrocede* reconocida tiene la siguiente forma **(ilustración 159)**:

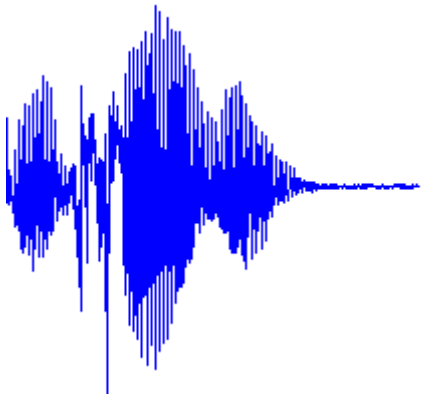


**Ilustración 159:** Gráfica orden *retrocede* reconocida.

Como se puede ver en las dos gráficas tienen una cierta similitud. Una vez vista la forma que tiene la señal *retrocede* se pasa a las pruebas obteniendo los siguientes resultados ***(Tabla 15)***:

| **Número de veces orden *retrocede*** | **Lugar silencioso** | **Con TV encendida** | **Volumen TV al máximo** | **Exterior** |
|---|---|---|---|---|
| **1** | Reconocido | Reconocido | Reconocido | Reconocido |
| **2** | Reconocido | Reconocido | Reconocido | Reconocido |
| **3** | Reconocido | Reconocido | Reconocido | Reconocido |
| **4** | Reconocido | Reconocido | Reconocido | Reconocido |
| **5** | Reconocido | Reconocido | Reconocido | Reconocido |
| **6** | Reconocido | Reconocido | Reconocido | Reconocido |
| **7** | Reconocido | Reconocido | Reconocido | Reconocido |
| **8** | Reconocido | Reconocido | Reconocido | Reconocido |

| | | | | |
|---|---|---|---|---|
| **9** | Reconocido | Reconocido | Reconocido | Reconocido |
| **10** | Reconocido | Reconocido | Reconocido | Reconocido |
| **Resultado** | **100%** | **100%** | **100%** | **100%** |

**Tabla 15:** Resultado reconocimiento orden *retrocede.*

Para la prueba de reconocimiento de la orden *retrocede* en un lugar silencioso y en el mismo lugar con la TV encendida con volumen normal se ha obtenido el 100%. Con la televisión encendida y el volumen al máximo se ha obtenido un 100% de reconocimiento de la orden, pero el ruido de la TV activa el reconocedor, pero no reconoce ningún patrón, pero sale patrón desconocido. Para el ensayo en el exterior se ha obtenido buenos resultados 100%.

### 5.5.4 Resultado reconocimiento orden *gira derecha.*

Para ver los resultados de reconocimiento sobre la orden *gira derecha* se dirá diez veces y se anotarán las veces que ha sido reconocida. Así con cada uno de los lugares anteriormente citados.

Primero se guardará el patrón de la orden *gira derecha*, obteniéndose la siguiente gráfica de sonido **(ilustración 160)**:

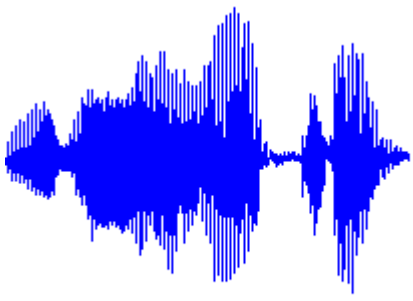


**Ilustración 160:** Gráfica patrón *gira derecha*

La gráfica de la orden *gira derecha* reconocida tiene la siguiente forma **(ilustración 161)**:

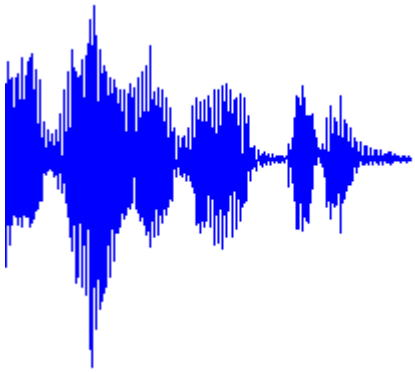


**Ilustración 161:** Gráfica orden *gira derecha* reconocida.

Como se puede ver en las dos gráficas tienen una cierta similitud. Una vez vista la forma que tiene la señal *gira derecha* se pasa a las pruebas obteniendo los siguientes resultados ***(Tabla 16)***:

| **Número de veces orden *gira derecha*** | **Lugar silencioso** | **Con TV encendida** | **Volumen TV al máximo** | **Exterior** |
|---|---|---|---|---|
| **1** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **2** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **3** | Reconocido | Reconocido | Reconocido | Reconocido |
| **4** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **5** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **6** | Reconocido | Reconocido | Reconocido | Reconocido |
| **7** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **8** | Reconocido | Reconocido | No Reconocido | Reconocido |

| | | | | |
|---|---|---|---|---|
| **9** | Reconocido | Reconocido | Reconocido | Reconocido |
| **10** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **Resultado** | **100%** | **100%** | **30%** | **100%** |

**Tabla 16:** Resultados reconocimiento orden *gira derecha.*

Para la prueba de reconocimiento de la orden *gira derecha* en un lugar silencioso y en el mismo lugar con la TV encendida con volumen normal se ha obtenido el 100%. Con la televisión encendida y el volumen al máximo se ha obtenido un 30% de reconocimiento de la orden, ya que con tanto ruido no se reconoce bien esta orden, al ser el umbral del ruido mayor que el umbral mínimo de disparo se activa el reconocedor, pero mientras no se habla no reconoce ningún patrón y sale patrón desconocido. Para el ensayo en el exterior se ha obtenido buenos resultados 100%.

### 5.5.5 Resultado reconocimiento orden *gira izquierda*.

Para ver los resultados de reconocimiento sobre la orden *gira izquierda* se dirá diez veces y se anotará las veces que ha sido reconocida. Así con cada uno de los lugares anteriormente citados.

Primero se guardará el patrón de la orden *gira izquierda* obteniéndose la siguiente gráfica de sonido **(ilustración 162)**:

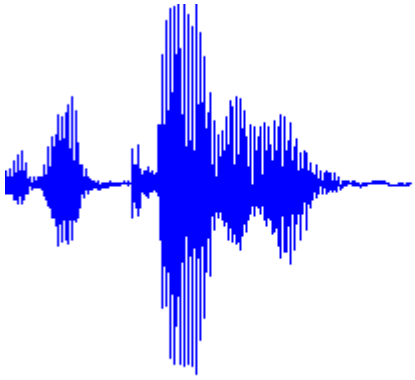


**Ilustración 162:** Gráfica patrón *gira izquierda.*

La gráfica de la orden *gira izquierda* reconocida tiene la siguiente forma **(ilustración 163)**:

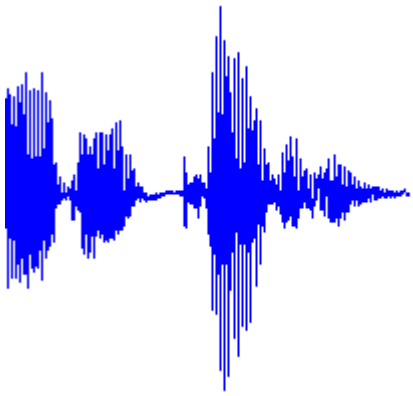


**Ilustración 163:** Gráfica orden *gira izquierda* reconocida.

Como se puede ver en las dos gráficas tienen una cierta similitud. Una vez visto la forma que tiene la señal *gira izquierda* se pasa a las pruebas obteniendo los siguientes resultados ***(Tabla 17)***:

| **Número de veces orden *gira izquierda*** | **Lugar silencioso** | **Con TV encendida** | **Volumen TV al máximo** | **Exterior** |
|---|---|---|---|---|
| **1** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **2** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **3** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **4** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **5** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **6** | Reconocido | Reconocido | No Reconocido | No Reconocido |
| **7** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **8** | Reconocido | Reconocido | No Reconocido | Reconocido |

| | | | | |
|---|---|---|---|---|
| **9** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **10** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **Resultado** | **100%** | **100%** | **0%** | **90%** |

**Tabla 17:** Resultados reconocimiento orden *gira izquierda.*

Para la prueba de reconocimiento de la orden *gira izquierda* en un lugar silencioso y en el mismo lugar con la TV encendida con volumen normal se ha obtenido el 100%. Con la televisión encendida y el volumen al máximo se ha obtenido un 0% de reconocimiento de la orden, ya que con tanto ruido no se reconoce. Al ser el umbral del ruido mayor que el umbral mínimo de disparo se activa el reconocedor, pero mientras no se habla no reconoce ningún patrón y sale patrón desconocido. Para el ensayo en el exterior se ha obtenido buenos resultados 90%.

### 5.5.6 Resultado reconocimiento orden *más deprisa.*

Para ver los resultados de reconocimiento sobre la orden *más deprisa* se dirá diez veces y se anotarán las veces que ha sido reconocida. Así con cada uno de los lugares anteriormente citados.

Primero se guardará el patrón de la orden *más deprisa* obteniéndose la siguiente gráfica de sonido **(ilustración 164)**:

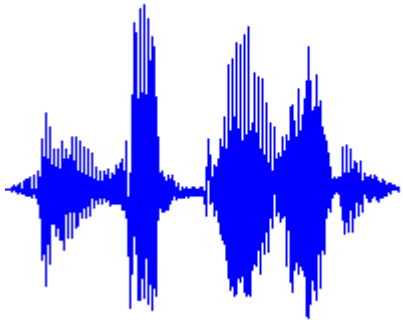


**Ilustración 164:** Gráfica patrón *más deprisa.*

La gráfica de la orden *más deprisa* reconocida tiene la siguiente forma **(ilustración 165)**:

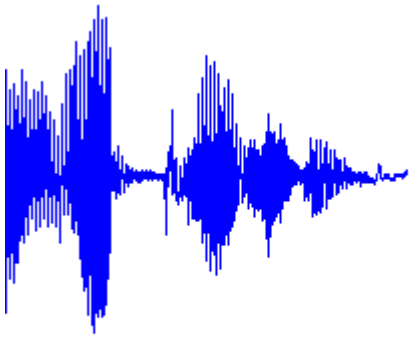


**Ilustración 165:** Gráfica orden *más deprisa* reconocida.

Como se puede ver en las dos gráficas tienen una cierta similitud. Una vez vista la forma que tiene la señal *más deprisa* se pasa a las pruebas obteniendo los siguientes resultados ***(Tabla 18)***:

| **Número de veces orden *más deprisa*** | **Lugar silencioso** | **Con TV encendida** | **Volumen TV al máximo** | **Exterior** |
|---|---|---|---|---|
| **1** | Reconocido | Reconocido | Reconocido | Reconocido |
| **2** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **3** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **4** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **5** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **6** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **7** | Reconocido | Reconocido | No Reconocido | No Reconocido |
| **8** | Reconocido | Reconocido | No Reconocido | Reconocido |

| | | | | |
|---|---|---|---|---|
| **9** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **10** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **Resultado** | **100%** | **100%** | **10%** | **90%** |

**Tabla 18:** Resultados reconocimiento orden *más deprisa.*

Para la prueba de reconocimiento de la orden *más deprisa* en un lugar silencioso y el mismo lugar con la TV encendida con volumen normal se ha obtenido el 100%. Con la televisión encendida y el volumen al máximo se ha obtenido un 10% de reconocimiento de la orden, ya que con tanto ruido no se reconoce bien. Al ser el umbral del ruido mayor que el umbral mínimo de disparo se activa el reconocedor, pero mientras no se habla no reconoce ningún patrón y sale patrón desconocido. Para el ensayo en el exterior se ha obtenido buenos resultados 90%.

### 5.5.7 Resultado reconocimiento orden *más despacio*.

Para ver los resultados de reconocimiento sobre la orden *más despacio* se dirá diez veces y se anotarán las veces que ha sido reconocida. Así con cada uno de los lugares anteriormente citados.

Primero se guardará el patrón de la orden *más despacio* obteniéndose la siguiente gráfica de sonido **(ilustración 166)**:

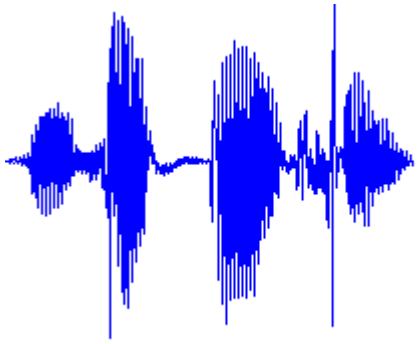


**Ilustración 166:** Gráfica patrón *más despacio.*

La gráfica de la orden *más despacio* reconocida tiene la siguiente forma **(ilustración 167)**:

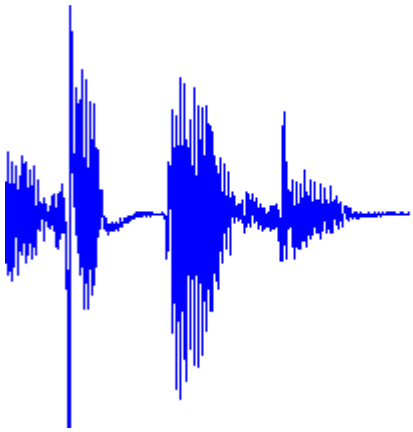


**Ilustración 167:** Gráfica orden *más despacio* reconocida.

Como se puede ver en las dos gráficas tienen una cierta similitud. Una vez vista la forma que tiene la señal *más despacio* se pasa a las pruebas obteniendo los siguientes resultados ***(Tabla 19)***:

| **Número de veces orden *más despacio*** | **Lugar silencioso** | **Con TV encendida** | **Volumen TV al máximo** | **Exterior** |
|---|---|---|---|---|
| **1** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **2** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **3** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **4** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **5** | Reconocido | Reconocido | Reconocido | Reconocido |
| **6** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **7** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **8** | Reconocido | Reconocido | No Reconocido | Reconocido |

| | | | | |
|---|---|---|---|---|
| **9** | Reconocido | Reconocido | Reconocido | Reconocido |
| **10** | Reconocido | Reconocido | No Reconocido | Reconocido |
| **Resultado** | **100%** | **100%** | **30%** | **100%** |

**Tabla 19:** Resultados reconocimiento orden *más despacio.*

Para la prueba de reconocimiento de la orden *más despacio* en un lugar silencioso y en el mismo lugar con la TV encendida con volumen normal se ha obtenido el 100%. Con la televisión encendida y el volumen al máximo se ha obtenido un 30% de reconocimiento de la orden, ya que con tanto ruido no se reconoce. Al ser el umbral del ruido mayor que el umbral mínimo de disparo se activa el reconocedor, pero mientras no se habla no reconoce ningún patrón y sale patrón desconocido. Para el ensayo en el exterior se ha obtenido buenos resultados 100%.

La siguiente ***tabla 20*** muestra los resultados de reconocimiento obtenidos por cada orden.

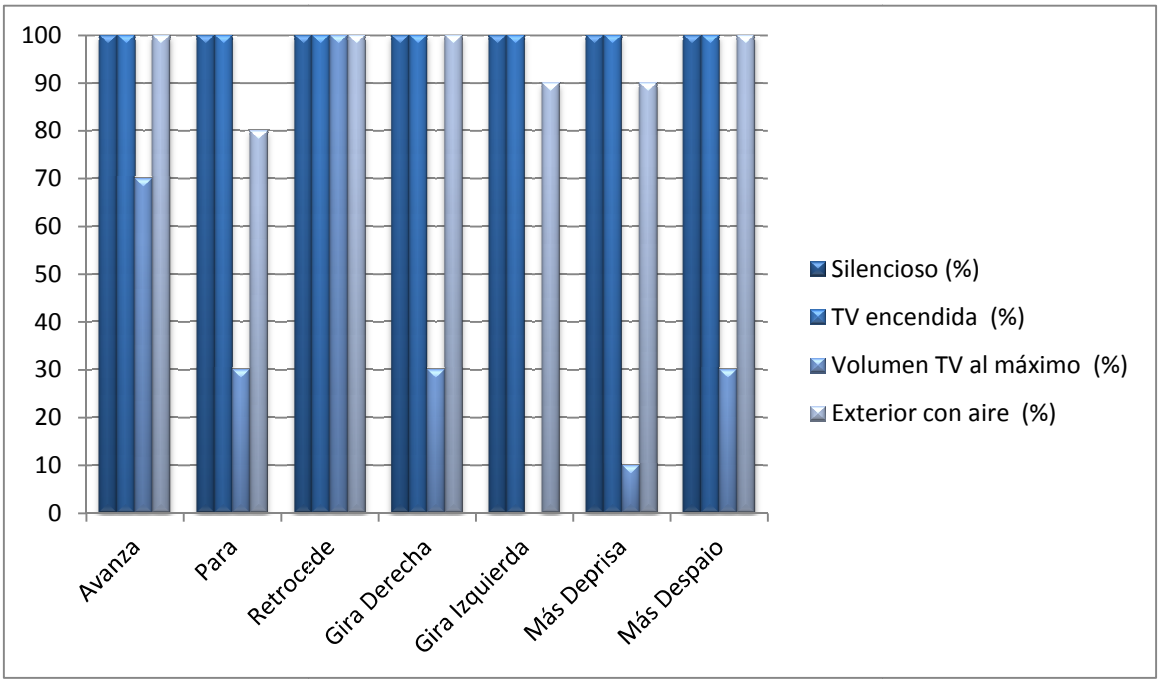
| **Orden a reconocer** | **Lugar silencioso (%)** | **Tv encendida volumen normal. (%)** | **TV encendida con volumen al máximo. (%)** | **Exterior con ráfagas de aire. (%)** |
|---|---|---|---|---|
| **Orden Avanza.** | 100 | 100 | 70 | 100 |
| **Orden Para** | 100 | 100 | 30 | 80 |
| **Orden Retrocede** | 100 | 100 | 100 | 100 |
| **Orden Gira Izquierda** | 100 | 100 | 30 | 100 |
| **Orden Gira Derecha** | 100 | 100 | 0 | 90 |

| | | | | |
|---|---|---|---|---|
| **Orden Más Deprisa** | 100 | 100 | 10 | 90 |
| **Orden Más Despacio** | 100 | 100 | 30 | 100 |
| **Resultado Total Del Reconocedor** | **100%** | **100%** | **38.57%** | **94.28%** |

**Tabla 20:** Resultados generales del reconocimiento de voz.

Como conclusión viendo la tabla anterior todas las pruebas han sido satisfactorias menos en la que hay mucho ruido (TV con volumen al máximo), por eso se ha optado a un segundo control mediante **Joystick** ya que en estas situaciones de mucho ruido el robot no se podría controlar por voz.

En la siguiente ***gráfica de resultados 9*** se puede ver los resultados obtenidos del reconocimiento de voz.




**Gráfica de resultados 9:** Reconocimiento de órdenes.

## 5.6 Resultados de visualización del entorno que rodea al robot.

La manera por la que se optó para ver el entorno que rodea al robot a través del PC fue un sistema muy sencillo y con buenos resultados, ya que nos permite ver el entorno en tiempo real, hasta incluso escuchar el audio. Con una simple aplicación y un teléfono móvil tenemos un sistema de captación de video bastante eficaz.

# CAPÍTULO 6: ESTUDIO ECONÓMICO.

# 6 ESTUDIO ECONÓMICO.

Para la elaboración de este proyecto se pretendió desde un primer momento hacerlo lo más económico posible. Se hizo un estudio de todos los componentes que lleva el robot para darle las características deseadas. Una vez que se hizo la lista de los componentes que hacían falta se buscaron en varias tiendas online los componentes para poder comparar precios y conseguirlos al menor precio posible. A continuación se detalla una lista donde aparecen los componentes buscados en diferentes tiendas online.

**Placa Arduino Uno:**

- Arduino uno 13.49€ sin gastos de envió o 1.23€ envió exprés; http://www.ebay.es/itm/Arduino-Uno-R3-Nuevo-USB-Cable-Incluido-/271233220194?pt=LH_DefaultDomain_186&hash=item3f26c27e62&_uhb=1
- Arduino uno 15.63€ sin gastos de envíos; http://www.miniinthebox.com/es/2012-arduino-uno-junta-r3-atmega328-con-el-cable-usb_p590432.html
- Arduino uno 12.98€ sin gastos de envíos; http://dx.com/es/p/uno-r3-development-board-microcontroller-mega328p-atmega16u2-compat-for-arduino-blue-black-215600

**Módulo APC220:**

- Wireless APC220 24.56€ sin gastos de envío http://dx.com/es/p/arduino-apc220-wireless-rf-modules-w-antennas-usb-converter-143011

**Tarjeta controladora motor:**

- Tarjeta controladora de motores L298N 9.19€ sin gastos de envíos http://www.miniinthebox.com/es/diy-motor-drive-junta-modulo-inteligente-trolley_p438441.html?pos=ultimately_buy_8

**Montaje inalámbrico Xbee:**

- Shield Xbee Arduino 11.95€ http://www.miniinthebox.com/es/xbee-arduino-escudo-compatible-v3-0-modulo_p381442.html
- 2 Módulos Xbee 59.29€ gastos de envío e IVA incluidos http://www.electan.com/xbee-2mw-con-antena-chip-serie-p-3123.html
- Xbee explorer USB 24.08€ más gastos de envío incluidos e IVA. http://www.electan.com/xbee-explorer-usb-p-3121.html

**Sensor Ultrasonido.**

- Sensor Ultrasonidos 4.13€ más gastos de envío http://www.miniinthebox.com/es/modulo-ultrasonico-hc-sr04-medir-la-distancia-del-transductor-del-sensor-para-arduino-201211270080054_p478889.html

**Chasis:**

- Chasis 2 ruedas 14.77€ sin gastos de envío. http://dx.com/p/37-in-1-smart-car-chassis-kit-for-arduino-black-yellow-151814
- Chasis 2 ruedas 22.07€ http://www.miniinthebox.com/es/intelligent-chassis-seguimiento-de-carritos-con-disco-codificado_p407335.html

**Cámara IP:**

- Cámara IP 57€ http://todoelectronica.com/camara-vigilancia-apexis-tipo-cubo-wifi-infrarrojos-color-negro-p-14603.html
- Cámara IP 32.89€ http://www.ebay.es/itm/IP-camara-videovigilancia-CCTV-vision-nocturna-WIFI-1-4-CMOS-wireless-blanco-/271288177858?pt=LH_DefaultDomain_186&hash=item3f2a0914c2&_uhb=1

La lista anterior corresponde a las tiendas más baratas. Se encontraron más tiendas online, pero no se han puesto por superar estos precios. A continuación se verá la diferencia de precio montando el robot con los módulos APC220 y con los módulos XBee.

- Precio robot con módulos inalámbricos APC220 (para el control remoto):
  Arduino uno 15.63€
  APC220 24.56
  L298N 9.19€
  Ultrasonidos 4.13€
  Chasis (tienda dx) 14.77€ o Chasis (tienda miniinthebox) 22.07€
  Cámara IP 32.89€
  **Total:**
  **Aprox. 101.17€ con chasis tienda dx**
  **Aprox. 108.47€ Con chasis tienda miniinthebox**

- Precio robot con módulos Xbee (Para el control remoto):
  Arduino uno 15.63€
  Shield Xbee Arduino 11.95€
  Módulo Xbee 59.29€
  Xbee Explorer USB 24.08€
  L298N 9.19€
  Ultrasonidos 4.13€
  Chasis (tienda dx) 14.77€ o Chasis (tienda miniinthebox) 22.07€
  Cámara IP 32.89€
  **Total:**
  **Aprox. 171.93€ con chasis tienda dx**
  **Aprox. 179.23€ Con chasis tienda miniinthebox**

Debido a la diferencia, casi 71 euros de un montaje a otro, se eligió por montar el robot con los módulos APC220, ya que estos módulos tienen características más que suficientes para nuestro proyecto. Otra opción fue incorporar un teléfono móvil al robot para hacer uso de la cámara de él, consiguiendo ahorrar 32.89 euros que valía la cámara IP más barata.

Al final se llegó a una lista definitiva de compra:

- Arduino Uno 15.63€ http://www.miniinthebox.com/es/2012-arduino-uno-junta-r3-atmega328-con-el-cable-usb_p590432.html
- Controlador motor L298N 9.19€ http://www.miniinthebox.com/es/diy-motor-drive-junta-modulo-inteligente-trolley_p438441.html?pos=ultimately_buy_8
- Sensor ultrasonidos 4.13€ http://www.miniinthebox.com/es/modulo-ultrasonico-hc-sr04-medir-la-distancia-del-transductor-del-sensor-para-arduino-201211270080054_p478889.html
- Chasis, 2 ruedas y 2 motores 22.07€ http://www.miniinthebox.com/es/intelligent-chassis-seguimiento-de-carritos-con-disco-codificado_p407335.html

**Total euros en esa pagina 51.02€**

En la página dx.com hay que pedir esto porque no la hay en las páginas anteriores (no tiene gastos de envío):

- APC220 24.56€ http://dx.com/es/p/arduino-apc220-wireless-rf-modules-w-antennas-usb-converter-143011

**El total a pagar es 75.58 euros.** Al final con estos componentes y utilizando la cámara del teléfono móvil se ha conseguido un gran ahorro.

Para la conexión de los componentes y construcción de soportes se han ido utilizando materiales reciclados como son cables de otros dispositivos en desuso. La madera y el metacrilato son retales que no han costado nada. Lo único que hay que sumar a los 75.58 euros son las dos ruedas fijas que costaron 2,60 euros, haciendo un total de **78.18 euros.**

Las horas invertidas en la realización del proyecto son:

- En el mes de **octubre** y **noviembre** se investigó y estudió sobre robótica, dedicando cuatro días a la semana y cuatro horas al día, haciendo un total de aproximadamente **144 horas** en los dos meses.
- En **diciembre** se siguió con el estudio e investigación sobre robótica mientras a la vez se empezó con el estudio de reconocimiento de voz y también a investigar sobre lógica difusa. En este mes se dedicaron seis días a la semana e incluso

algunas semanas los siete días con un total de 6 horas diarias aproximadamente. En total en este mes se dedicaron alrededor de **156 horas.**

- En el mes de **enero** y **febrero** se siguió con el reconocimiento de voz, se empezó a estudiar sobre lógica difusa y se empezó a programar en Arduino, dedicando seis días a la semana con cinco horas cada día. En estos dos meses se han hecho aproximadamente **180 horas.**
- **Marzo** fue un mes crucial ya que se termino de montar el robot, se acabó la lógica difusa y también se terminó de programar en Arduino, dedicando cinco días a la semana y cada día seis horas. En este mes se dedico aproximadamente **120 horas.**
- El mes de **abril** las dos primeras semanas no se dedicaron al proyecto. Las dos últimas semanas se han dedicado muchas horas de trabajo para completar el documento e ir dando los últimos retoques al proyecto. Las dos últimas semanas se dedico tiempo todos los días, llegando días en los que se echaron 11 horas. En estas dos semanas se han dedico aproximadamente **126 horas.**
- Las dos primeras semanas de **mayo** se han echado las mismas horas aproximadamente que las dos últimas semanas de abril, para tener completado y finalizado el trabajo. **126 horas** aproximadamente.

Sumando todas las horas dedicas al proyecto hacen un total de 852 horas, a estas se le añadiría unas 28 horas en tutorías con el profesor, sumando un total entre todo de **880 horas.** Estas horas si las tiene que dedicar un ingeniero en su puesto de trabajo equivaldría aproximadamente **5 meses y medio.** Un ingeniero recién egresado cobra en España alrededor de 1400 euros mensuales, por lo tanto, este proyecto hubiera costado desarrollarlo aproximadamente 7700 euros más los componentes hacen un total **aproximadamente de 7778.18 euros.**

Como conclusión se detallará todas las horas y el precio en la siguiente ***tabla 21***:

| **Mes** | **Nº Horas trabajas** | **Precio*hora (8.75€ la hora)** |
|---|---|---|
| **Octubre y Noviembre** | 144 | 1260 |
| **Diciembre** | 156 | 1365 |
| **Enero y Febrero** | 180 | 1575 |
| **Marzo** | 120 | 1050 |
| **Abril** | 126 | 1102.5 |
| **Mayo** | 126 | 1102.5 |
| **Tutorías** | 28 | 245 |
| **Total** | **852 horas** | **7700€** |
| | Componentes (78.18€) | **7778.18€** |

**Tabla 21:** Horas trabajadas y precio total.

Las horas dedicadas a cada parte del proyecto se verá en la siguiente ***tabla 22***:

| **Sección** | **Nº Horas trabajas** | **Precio*hora (8.75€ la hora)** |
|---|---|---|
| **Investigación sobre robótica** | 160 | 1400 |
| **Reconocimiento de voz** | 190 | 1662.5 |
| **Lógica difusa** | 130 | 1137.5 |
| **Montaje robot** | 40 | 350 |
| **Programar Arduino** | 80 | 700 |
| **Documento** | 252 | 2205 |
| **Total** | **852 horas** | **7455€** |

**Tabla 22:** Horas dedicadas a cada sección del proyecto y precio.

# CAPÍTULO 7: CONCLUSIONES.

# 7 CONCLUSIONES.

Para el desarrollo del presente trabajo fin de grado se han invertido muchas horas de investigación en búsqueda de información y desarrollo sobre robots móviles. Todas estas horas de trabajo me han servido para ampliar los conocimientos sobre robótica y conocer las competencias sobre su uso. También he visto la importancia de la programación en el diseño de los robots así como la aplicación de las matemáticas y leyes de electricidad/electrónica para la resolución de problemas de ingeniería. Todos estos conocimientos han sido un complemento muy satisfactorio para mi formación en el grado de Ingeniería Eléctrica.

Para este trabajo fin de grado se propuso como objetivo fundamental construir desde cero un robot móvil controlado por voz de la forma más económica posible. Una vez que se fue desarrollando el proyecto se pensó en aplicar técnicas de lógica difusa, lo que fue una decisión muy buena. Gracias a este motor de razonamiento difuso el robot no tiene un comportamiento tan prefijado. Dicho proyecto podría tener gran utilidad para acceder a zonas o áreas de difícil acceso para las personas o incluso en las que pudiera estar en peligro su integridad física. Otra gran utilidad sería la verificación de posibles anomalías de los lugares por donde el robot se mueva, ya que lleva incorporado una cámara para la visualización en un PC del entorno que rodea al robot. Dicho robot podrá ser controlado por personas con discapacidad ya que su principal control es por voz.

Para poder realizar todo lo anterior con garantías, en la primera fase del proyecto, se realizó un estudio del estado del conocimiento en robótica móvil durante un periodo de 3 meses. La construcción de un sistema de control por voz implica afrontar diversas áreas como la comunicación entre dispositivos diferentes (robot y PC), interpretación de la información de sensores, tratamiento y procesamiento de señales digitales, entre otros. Durante esta fase pudimos ver los posibles problemas que más adelante tendríamos que afrontar con algunas soluciones propuestas. Tras estas soluciones propuestas y el estudio propio realizado sirvió para plantear nuestras propias soluciones y cumplir con los objetivos marcados inicialmente.

Al final el proyecto se dividió en tres objetivos fundamentales:

- Diseño y construcción desde cero de un robot móvil lo más económico posible.
- Reconocimiento de órdenes mediante voz para el control del robot.
- Motor de inferencia difusa para definir el comportamiento del robot (Inteligencia Artificial).

Todos los objetivos citados se han cumplido. En primer lugar como se indica en el capítulo 4, se diseñó un robot móvil haciendo uso de Arduino y componentes compatibles con él. Dicho diseño es limitado a la disponibilidad económica.

En segundo lugar las librerías del programa de reconocimiento de voz, permiten la comunicación del usuario con el PC en tiempo real. La complejidad del problema requirió muchas horas de estudio e investigación sobre este tema, aproximadamente 2 meses. Los lenguajes de programación que permiten procesamiento de señales son limitados. Se eligió MATLAB ya que es un programa muy potente para el cálculo de matrices y por tener librerías ya construidas sobre el tratamiento de señales. Se hizo necesario también la construcción de librerías de comunicación entre MATLAB y Arduino, existiendo la posibilidad de controlar Arduino directamente desde MATLAB, pero se llegó a la conclusión de comunicar Arduino con MATLAB a través del puerto serie, ya que controlar Arduino desde MATLAB, los controles eran limitados y haciéndolo de la otra manera no había limitación y dichas librerías permiten comunicarse de forma inalámbrica. La comunicación inalámbrica fue otra solución propuesta, ya que permite la comunicación entre el robot y PC sin cables.

Una vez comunicado robot-computador, se procedió al diseño de los movimientos del robot y la creación de librerías de Arduino. Como en todo proyecto, las soluciones que se plantearon inicialmente presentaban problemas y ante estos problemas se proponían nuevas soluciones, llegando a una versión de movimientos satisfactorios.

En tercer lugar, la creación del motor de inferencia difusa fue otro gran reto al que nos hemos enfrentado. Hizo necesario investigar y estudiar sobre la utilización de lógica difusa. Tras ensayos de comportamiento del control difuso se dieron problemas que se fueron resolviendo realizando nuevos ensayos y estudios con diversos factores y variables,

llegando a una solución final donde el robot tiene un control difuso muy bueno cumpliéndose este objetivo.

Por último, una vez que se finalizó el motor de inferencia difusa se incorporó a la librería de reconocimiento de voz para su posterior comunicación con Arduino. En este caso hubo un problema que llevo varios días en resolverlo, ya que MATLAB y Arduino no lograban sincronizarse para la transferencia de datos entre ambos llegando a un resultado donde se tuvo que programar que MATLAB diera la orden a Arduino para iniciar la transferencia de datos y MATLAB estuviera a la espera de ellos. Se resolvió el problema satisfactoriamente.

La consecución de estos objetivos ha permitido obtener unos resultados positivos, al conseguir controlar el robot mediante voz con un control difuso sin necesidad de controlarlo por medio de botones. A partir de este proyecto pueden surgir otros muchos trabajos de fin de grado que presenten mejoras o satisfagan nuevos objetivos. Por ejemplo, la creación de un robot que vigile a una persona mayor, donde habría que añadir una serie de modificaciones no muy complejas al proyecto actual.

Para finalizar, me gustaría resaltar que la realización de este proyecto ha sido una de las mejores experiencias con un final muy satisfactorio durante el desarrollo formativo en el grado de ingeniería eléctrica, puesto que se puede ver como una máquina es capaz de obedecer órdenes comunicadas por voz. Para cualquier estudiante de electricidad es de gran satisfacción que un proyecto donde se inicia con ideas, suposiciones, colaboración del profesorado dándome sugerencias y pocas ideas sobre el tema a tratar, se logre el funcionamiento cumpliendo los objetivos marcados.

# 8. BIBLIOGRAFÍA.

# 8 BIBLIOGRAFÍA

[1] **Albertí Bertran Eduard** Procesado digital de señales: Fundamentos para comunicación y control [Libro]. - Catalunya : UPC, 2006.

[2] **Barragán Guerrero Diego Orlando** Manual de interfaz gráfica de usuario en Matlab. - 2008. - Vol. 1.

[3] **Comite Español de Automática** Libro blanco de la Róbotica en España: Investigación, tecnologias y formación [Libro]. - [s.l.] : CEA-GTRob con subvención del MEC, 2011. - Vol. 1ª edicción..

[4] **Crespo Cadenas Carlos** Radiocomunicación [Libro]. - [s.l.] : Prentince-Hall, 2008.

[5] **CTM Electrónica** Manual Módulo Transceptor (APC220ES) [Informe]. - [s.l.] : APPCON Technologies, 2008.

[6] **CTM Electrónica** Munual de Inicio APPCON [Informe]. - [s.l.] : APPCON Tenhnologies, 2008.

[7] **D. Guzmán V.M Castaño.** La Lógica difusa en ingeniería: Principios, aplicaciones y futuro. - [s.l.] : Ciencia y Tecnología, 2006.

[8] **Durán Vicente Mª Isable y Benido Matias Tamara** Lógica Borrosa // Universidad Carlos III. - Madird : Autoedición.

[9] **Faúndez Zanuy Marcos** Sistemas de comunicaciones [Libro]. - [s.l.] : Marcombo, 2001.

[10] **Faúndez Zanuy Marcos** Tratamiento digital de voz e imagen y aplicación a la multimedia [Libro]. - [s.l.] : Marcombo, 2000.

[11] **Fernández de Córdoba Martos Gonzalo** Creación de interfaces Gráficas de Usuario (GUI) con Matlab. - Salamanca. : Autoedición, 2007..

[12] http://arduino.cc/en/Main/ArduinoBoardUno [En línea].

[13] **L. Tokheim Roger** Fundamentos de los microprocesadores [Libro]. - Mexico : Mc Graw Hill, 1991.

[14] **Laster Clay** Guia de radioaficionado principiante. [Libro]. - [s.l.] : Marcombo, 1984.

[15] **Martín del Brio Bonifacio y Sanz Molina Alfredo** Redes neuronales y sistemas borrosos [Libro]. - [s.l.] : RA-MA, 2006. - Vol. 2º Edición.

[16] **Mayné Jordi** Estado actual de las comunicaciones por Radio Frecuencia. [Informe]. - [s.l.] : Silica, 2009.

[17] **Ortega García Javier y González Rodríguez Javier** Procesado de voz: Reconocimiento de mensaje y locutor. - [s.l.] : Autoedición.

[18] **Pajares Martin Sanz Gonzalo y Santos Peñas Matilde** Inteligencia Artificial e Ingeniería del Conocimiento [Libro]. - [s.l.] : RA-MA, 2005.

[19] **Panta Mártinez Javier** Control domótico por voz // Proyecto Final de Carrera Ing. en informática de Sistemas, Universidad Politecnica de Valencia. - [s.l.] : Autoedición, 2012. - págs. 1-64.

[20] **Ponce Cruz Pedro** Inteligencia Artificial con Aplicaciones a la Ingeniería [Libro]. - [s.l.] : Alfaomega, 2010. - Vol. Primera Edición.

[21] **Ruiz Gutiérrez José Manuel** Manual de Programación Arduino. [Libro]. - [s.l.] : Autoedición, 2008.

[22] **Téllez Barrera Juan Carlos** Todo sobre Mini Robotica [Libro]. - [s.l.] : Priale, 2007. - Vol. Nº 33.

[23] **Tienda de Robótica** Guía básica de Arduino. [Informe]. - 2012.

[24] **Toledo Castellanos Miguel Ángel** Introducción a la robótica [Libro]. - Mexico : Mc Graw Hill, 2008.

[25] **Trillas Enric, Terricabras Josep Maria y Alsina Claudia** Introducción a la Lógica Borrosa [Libro]. - [s.l.] : Ariel, 1995.

[26] **Velásquez Rámirez Genoveva** Sistema de reconocimiento de voz en Matlab // Trabajo fin de Grado, Guatemala. - Guatemala : Autoedición, 2008.

[27] www.mathworks.es.

[28] www.wikipedia.org

[29] **Zabala Gonzalo** Robótica. [Libro]. - [s.l.] : Users Express, 2007. - Vol. 1ª ed..