2014

Escuela Técnica Superior de Ingeniería
Departamento de Ingeniería de Sistemas y Automática

Jesús Vico Serrano

Tutor: Federico Cuesta Rojo

[CONTROL DE UN ROBOT MÓVIL BASADO EN RASPBERRY PI Y ARDUINO]

Agradecimientos:

Al final de este camino tan solo tengo palabras de agradecimiento a toda mi familia, especialmente mi madre que ha sufrido mucho estos años, y a mi pareja cuyo apoyo me ha mantenido en el camino para llegar a esta meta.
También agradecer a mi tutor Federico Cuesta su dedicación y atención prestada durante la elaboración de este proyecto.

TABLA DE ILUSTRACIONES

1. Objetivo del proyecto

En el presente Proyecto Fin de Carrera se detalla el estudio y diseño de un robot móvil para realizar diversas funciones de navegación autónoma así como de control remoto del mismo, utilizando para ello un sistema distribuido basado en plataformas programables de propósito general.

La programación de robots gracias a plataformas de propósito general ofrece la posibilidad de aprovechar las ventajas que diferentes SO ofrecen a través de aplicaciones y entornos de desarrollo amigables así como la programación de aplicaciones multilenguaje. Del mismo modo, es también muy valorable la existencia de comunidades de desarrollo a nivel global para aplicaciones y la prestación de soporte.

La existencia actualmente de plataformas de propósito general, bajo coste y dimensiones reducidas, alto rendimiento está ayudando a situar la programación de controladores de robots sobre éstas como una de las opciones más valoradas por desarrolladores en todo el mundo.

En concreto, para el desarrollo de robots, existen sistemas operativos dedicados, los cuales ofrece las librerías de programación e intérpretes multilenguaje, comandos Shell así como diversas funcionalidades de diagnóstico y troubleshooting los cuales puede ser embebidos dentro de la plataforma de propósito general.

Algunos de los problemas típicos asociados al desarrollo de software para el control de robots son los siguientes:

- La programación secuencial no se adapta a las necesidades de control derivadas de la interacción del robot con un mundo inherentemente asíncrono.
- Abstracción del hardware específico del robot: servos, encoders etc.
- Manejo de la complejidad en operaciones, comunicación y distribución de datos.

En el presente proyecto se recoge también como objetivo el análisis de las ventajas y desventajas en las distintas posibilidades en el desarrollo de software para el control de robots.

Finalmente se ha elaborado un modelo de robot sobre un sistema distribuido basado en hardware programable de propósito general

Se han seguido varias vertientes a la hora de llevar a cabo el diseño del mismo:

- Programación de robot mediante la ejecución de software customizado, haciendo uso de librerías para la lectura en puerto serie y gestión de las interfaces de propósito general.

- Programación de software bajo el Sistema Operativo ROS, integración mediante librerías dedicadas de plataformas de control de propósito general tales como Arduino y Raspberry Pi.

Después de una primera aproximación y estudio del estado del arte en cuanto a programación de robots, se ha optado por llevar a cabo un diseño de controlador software basado en ROS y distribuido en dos placas de propósito general:

- Arduino Mega ADK

- Raspberry Pi rev B.

2. Plataformas hardware

En el presente Proyecto se contempla el uso de las placas computadora Raspberry Pi así como de Arduino Mega ADK para la implantación del sistema controlador del robot a implementar.

A continuación se resumen las características técnicas de cada placa así como las ventajas que ofrece de cara a la programación de robots.

2.1. Raspberry Pi

Raspberry Pi es una placa computadora (SBC) de bajo costo desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

2.1.1. Historia

En 2006, los primeros diseños de Raspberry Pi se basaban en el micro controladorAtmel ATmega644. Sus esquemas y el diseño del circuito impreso están disponibles para su descarga pública.

En mayo de 2009, la Fundación Raspberry Pi fue fundada en Caldecote, South Cambridgeshire, Reino Unido como una asociación caritativa que es regulada por la Comisión de Caridad de Inglaterra y Gales.

El administrador de la fundación, Eben Upton, se puso en contacto con un grupo de profesores, académicos y entusiastas de la informática para crear un ordenador con la intención de animar a los niños a aprender informática como lo hizo en 1981 el ordenador Acorn BBC Micro.15 16 El primer prototipo basado en ARM se montó en un módulo del mismo tamaño que una memoria USB. Tenía un puerto USB en un extremo y un puerto HDMI en el otro.

En agosto de 2011, se fabricaron cincuenta placas Alpha, que tenían las mismas características que el modelo B, pero eran un poco más grandes para integrar bien unas interfaces para depuración. En algunas demostraciones se podía ver la placa ejecutando el escritorio LXDE en Debian, Quake 3 a 1080p y vídeo Full HD H.264 a través de la salida HDMI.

En diciembre de 2011, 25 placas Beta del modelo B fueron ensambladas y probadas de un total de 100 placas vacías. El diagrama de componentes de las placas finales sería el mismo que el de esas placas Beta. Se hizo una emostración de la placa beta arrancando Linux, reproduciendo un tráiler de una película a 1080p y ejecutando el benchmark Rightware Samurai OpenGL ES.

Las primeras ventas comenzaron el 29 de febrero de 2012. Premier Farnell vendió toda su existencia de inventario a los pocos minutos del momento de lanzamiento, mientras que RS Components tuvo 100.000 peticiones de interés el primer día. En los seis meses siguientes llegarían a vender 500.000 unidades.

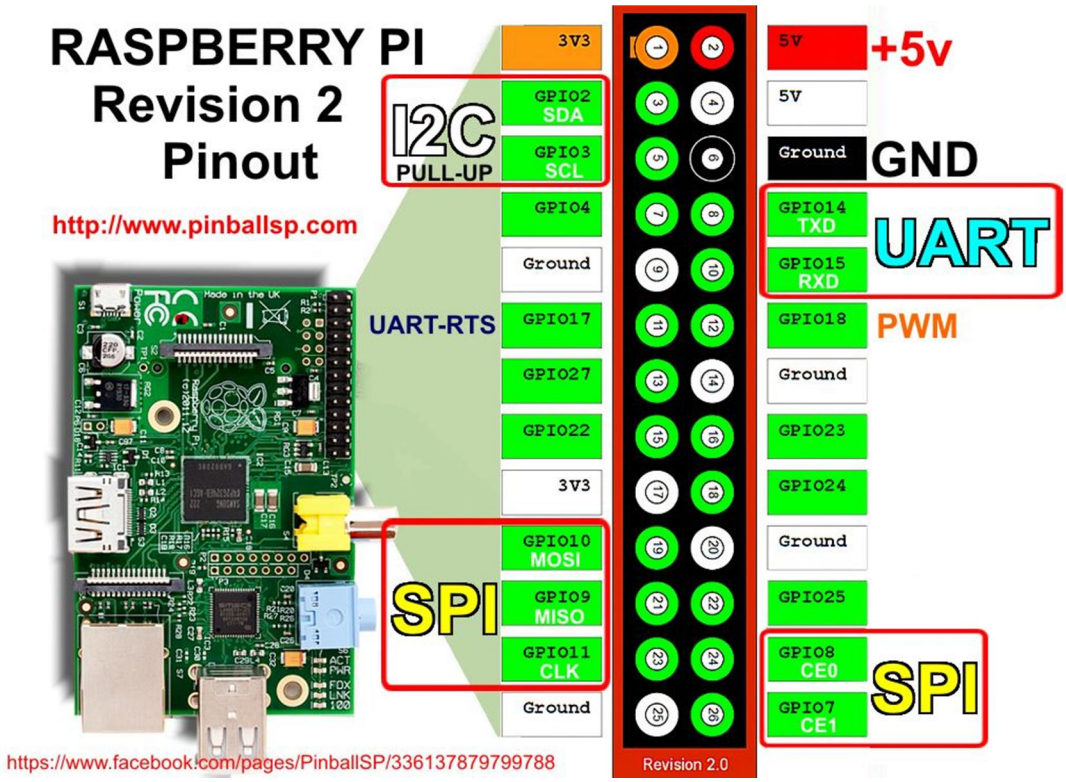
2.1.2. Características

El diseño incluye un System-on-a-chip (SoC) Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos "Turbo" para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía), un procesador gráfico (GPU) VideoCore IV, y 512 MB de memoria RAM aunque originalmente al ser lanzado eran 256 MB. El diseño no incluye un disco duro o una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa. El modelo B se vende a 35 $ y el modelo A a 25 $. La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, Raspbian (derivada de Debian), RISC OS, Arch Linux ARM (derivado de Arch Linux) y Pidora (derivado de Fedora); y promueve principalmente el aprendizaje del lenguaje de programación Python, y otros lenguajes como Tiny BASIC, C y Perl.

Las características de la placa Raspberry utilizada en este caso son las siguientes.

- Sistema Operativo: Linux ARM Raspbian wheezy (basado en Debian)
- Alimentación : 5V 3.5W
- CPU: ARM1176JZF-S 700Mhz
- GPU: Broadcom VideoCore IV
- Memoria RAM: 512 MB compartidos con GPU
- Almacenamiento: Tarjeta de memoria SD de 8 GB
- Interfaces (ver Figura 1)
    - 1 puerto Ethernet RJ-45 100Mbps.
    - 2 puertos USB 2.0
    - 1 micro USB
    - 1 puerto HDMI
    - 1 puerto 3.5MM Audio (Jack)
    - 1 interfaz RCA video
    - Conector CSI para cámara
    - 1 cabecera JTAG
    - 1 puerto DSI para display digital.
    - 8 pines de propósito general.
    - Interfaz SPI
    - Interfaz I2C
    - Interfaz UART

Figura 1 Raspberry Pi Pinout



Además de estas características, la placa Raspberry Pi cuenta con multitud de accesorios necesarios para el manejo diario y proporcionar un entorno de programación amigable. Entre ellos se ha utilizado un adaptador usb 802.11 b/g/n para la conexión wifi al router doméstico. Además de esto es aconsejable, aunque aún no se ha utilizado, la adquisición de un hub USB para ratón y teclado ya que al contar únicamente dos puertos será más cómodo a la hora de navegar por el entorno de ficheros gráfico.

El primer paso para la puesta en marcha de la placa en este caso es la instalación dentro de la tarjeta de memoria de una imagen de de la distribución de Linux Raspbian basada en Debian y especialmente aligerada y diseñada para su funcionamiento sobre esta placa.

Las ventajas que ofrece la utilización de esta placa para el diseño y control de robots son las siguientes:

- Arquitectura ARM,
- Posibilidad de implantación de SO, entornos de desarrollo y distribuciones específicas para el desarrollo de robots.
- Potencia de computación comparada con microcontroladores.
- Soporte multilenguaje (c, c++, python, java)

- Existencia de soluciones plug and play para mayor conectividad.
- Posibilidad de entorno de desarrollo embebido.
- Presencia de interfaces multimedia.
- Existencia de una amplia comunidad de desarrollo de aplicaciones sobre RPi.

Algunas de las desventajas encontradas durante el análisis realizado son las siguientes:

- Escasez de interfaces digitales.
- Escasez de interfaces analógicas
- Poca robustez frente a golpes
- Almacenamiento de datos basado en soporte físico SD card lo cual compromete la integridad de los datos y el sistema de ficheros ante posibles eventualidades relacionadas

2.2. Arduino Mega ADK

Arduino es una gama de placas microcontroladoras de bajo coste construidas por Arduino es una plataforma de electrónica abierta para la creación prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos. Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar.

En cuanto la placa utilizada en este proyecto (Mega ADK) se trata de una modificación de la placa Arduino Mega 2560 la cual facilita la interconexión con un dispositivo externo mediante un puerto USB. En el presente proyecto dicha interfaz será utilizada como puerto serie para la comunicación con Raspberry Pi.

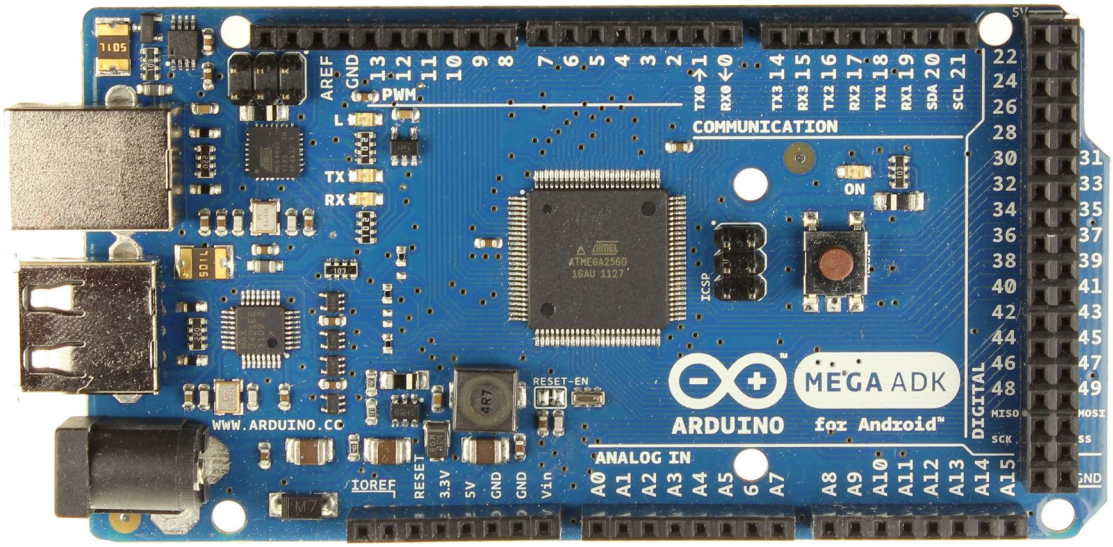
2.2.1. Características

- Microcontrolador: ATmega2560
- Alimentación: 5V
- Entrada: 7-12V
- Límites (max): 5.5-16V
- Pines digitales: 54 (14 con PWM)

- Pines analógicos: 16
- Corriente por pin: 40 mA
- Corriente sobre pin 3,3V: 50 mA
- Memoria Flash (programa): 256 KB (8 KB usados para el bootloader)
- SRAM: 8 KB
- EEPROM: 4 KB
- Reloj: 16 MHz

En la Figura 2 se puede observar la imagen y pinout de la placa Arduino utilizada.

Figura 2 Placa Arduino Mega ADK



2.2.2. Ventajas

- Asequible: una placa microcontrolador Arduino puede ser elaborada por uno mismo al encontrarse bajo una licencia de código abierto. Aún montada su precio es mucho más competitivo que el de otras plataformas microcontroladoras como PIC.

- Facilidad de programación: la programación se lleva a cabo mediante el desarrollo de sketches (scripts) los cuales serán compilados y cargados en la placa por el propio software de desarrollo Arduino IDE

- Versatilidad de pines tanto digitales como analógicos. Hasta un total de 54 pines digitales y 16 analógicos a utilizar tanto para el control del hardware del robot como para la lectura de información de los distintos sensores.

- Amplia comunidad de desarrollo.

- Posibilidad de agregación de nuevas librerías. Toda librería .h puede ser incluida para su ejecución en Arduino a través de Arduino IDE.

- Disponible puerto serie USB.

- Entorno de desarrollo sencillo y multi plataforma: Linux, Windows, Mac OS.

- Programación en lenguaje C++

- Plataforma de código abierto

- Hardware ampliable

Estas herramientas de desarrollo de aplicaciones con múltiples entradas y salidas suponen un gran potencial a aprovechar para la programación de un entorno mezcla de software y hardware como un robot.

La existencia de módulos PWM integrados en la placa, convertidores A/D, así como múltiples salidas digitales y puertos serie e i2c unido al rendimiento convierte a Arduino en una plataforma de bajo coste idónea para el control de servos, lectura de sensores, comunicación entre controladores de hardware lo que en definitiva supone el desarrollo de un robot.

Sin embargo Arduino también consta de limitaciones y entre ellas se encuentra:

- Imposibilidad de control de tiempo de ejecución de tareas, algo disponible en otras plataformas como PIC y requisito indispensable en aplicaciones de control avanzadas.

- Capacidad de almacenamiento de programa limitado: la potencia de cálculo viene limitada por la circunstancia de contar con una memoria EEPROM de únicamente 256KB así como un procesador de únicamente 8 bits.

- Frecuencia de reloj: ésta es únicamente de 16Mhz lo cual puede resultar limitante para el desarrollo de aplicaciones con altas necesidades en cuanto a velocidad de cálculo.

2.3.**Programación mediante microcontroaladores**

Tradicionalmente, la programación de robots se ha apoyado sobre plataformas microcontrodoras de bajo nivel de abstracción y mucho más especializadas que las descritas anteriormente. En concreto destacan la utilización de microcontroladores:

- PIC
- PLC
- Motorola HC8000.

Para el desarrollo industrial, grandes empresas especializadas en robótica tales como Kuka, iRobot, ABB hacen uso de plataformas bajo licencia propietaria a medida para las necesidades del prototipo en cuestión.

Las ventajas frente a estas plataformas tradicionales del uso de plataformas de propósito general como Arduino y Raspberry Pi son principalmente la de facilidad de programación frente a estas, facilidad de integración con periféricos para dotar de mayor conectividad y sobre todo y desde el punto de vista del desarrollo, la existencia de una comunidad a nivel mundial de programadores gracias a bajo coste y la licencia abierta sobre la que se basan estas placas.

Durante el desarrollo del proyecto se consideraron diversas alternativas para la ejecución del mismo haciendo uso de varias combinaciones en cuanto a lenguaje de programación y controlador. En primer lugar se consideró la idea de basar un robot con un controlador implantado sobre una placa Raspberry Pi corriendo Raspbian como se describe a continuación.

2.4. Opción de programación sobre Raspberry Pi

Tras una previa tarea de recopilación de información al respecto se analizaron los pros y los contras de la programación sobre Rpi teniendo en cuenta las limitaciones de la plataforma y la existencia de librerias que pudiesen resultar útiles.

2.4.1. Instalación de sistema operativo

El primer problema que nos encontramos es la necesidad de dotar a la placa Rpi de SO. Al tratarse la placa de una plataforma basada en arquitectura ARM se requiere de un SO sobre el que establecer el sistema de ficheros y las diversas aplicaciones necesarias.

La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, Raspbian (derivada de Debian), RISC OS 5, Arch Linux ARM (derivado de Arch Linux) y Pidora (derivado de Fedora).

Para la instalación del SO se opta por la opción más generalista y versátil la cual se corresponde con Raspbian.

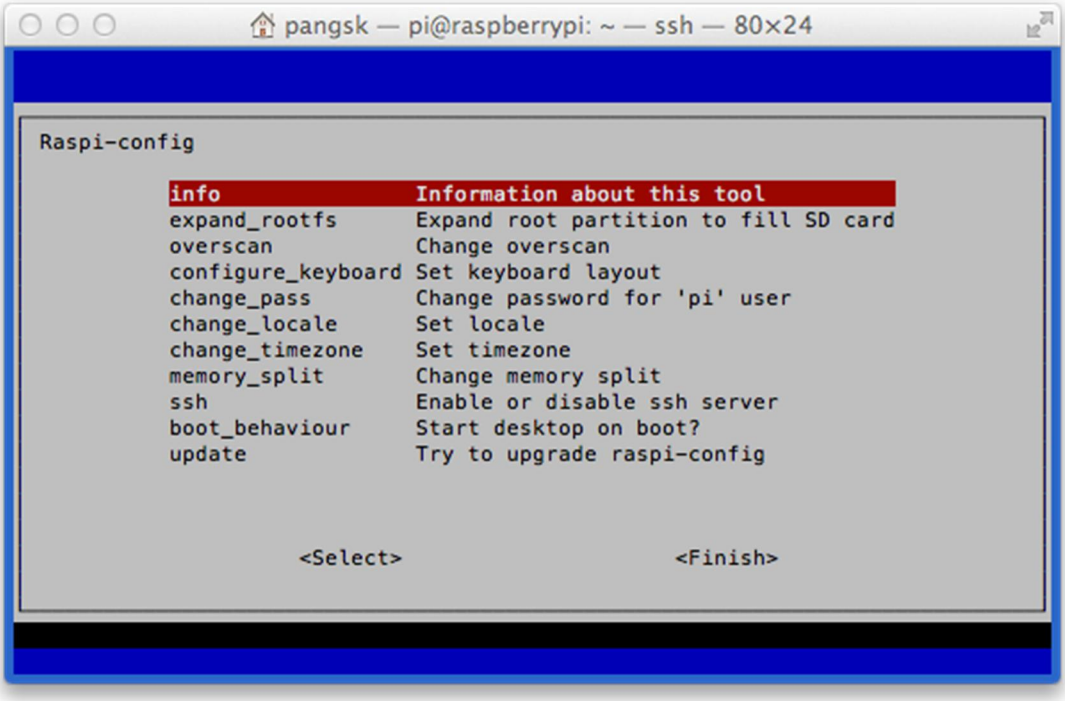
Para la instalación de la misma basta con la escritura sobre la tarjeta SD de al menos 4GB correctamente formateada de una imagen del SO disponible en la página web de la Raspberry Pi foundation. La escritura de la imagen se puede

realizar haciendo uso de varias herramientas. En el caso del presente proyecto se utilizó la aplicación Win32Imager sobre Windows 7.

Una vez establecida la imagen basta con su inserción en el Slot SD de la Rpi y conectar la alimentación. Enseguida se inicia el SO Raspbian Wheezy.

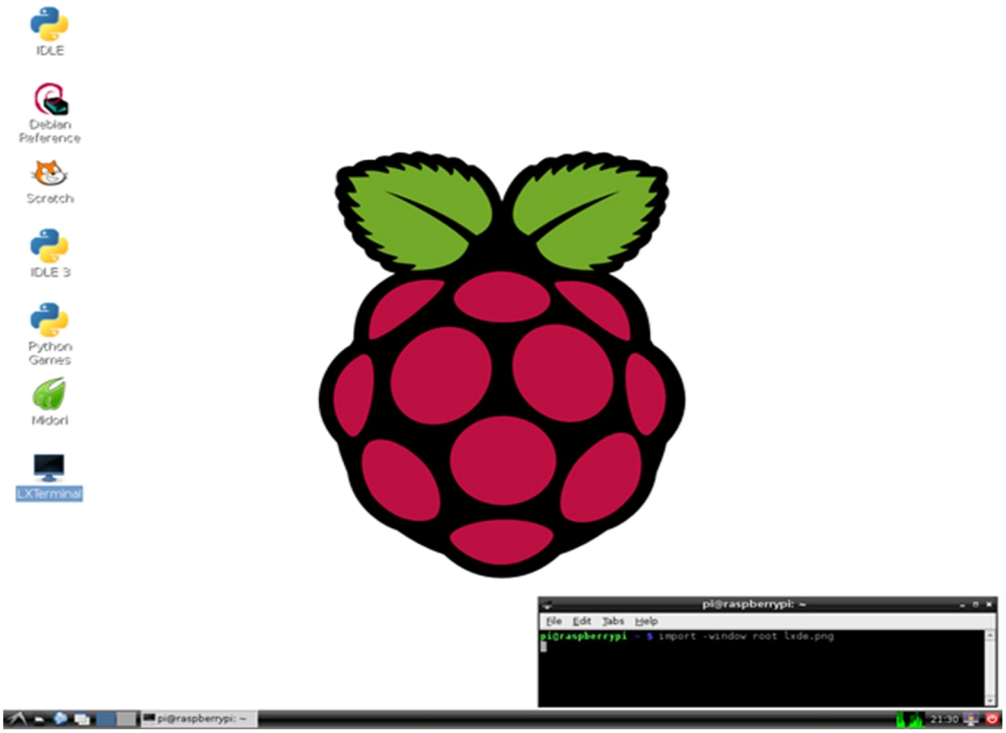
El primer paso consiste en la elección de parámetros de configuración básicos mediante la herramienta Raspbian-Config (ver Figura 3). En ella se expandirá en primer lugar la partición utilizada por el SO a todo el espacio disponible en la SD para posteriormente configurar las Locales del sistema instalando tanto la determinada por defecto EN-enUTF8 así como las locales de Español, estableciendo estas por defecto es_ES-UTF8. Posteriormente será necesesaria la configuración del teclado internacional de 105 teclas en idioma español. Tras esto, el SO estará listo para utilizar.

Figura 3 Menú Raspi Config



Una vez configurada la placa, se puede acceder al entorno gráfico LXDE mediante el comando startx (ver Figura 4)

Figura 4 Raspbian: entorno gráfico



2.4.2. Instalación de librerías

El segundo problema que se plantea es la necesidad de controlar mediante un programa ejecutable en la plataforma.

Para ello se recomienda el uso de la librería liberada por la comunidad Rpi que proporciona las funciones necesarias para el control de los pines de propósito general de la placa (detallados en el apartado anterior) mediante la ejecución de sccripts en lenguaje Python.

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

Al tratarse de un lenguaje de programación interpretado se requiere de la instalación de un intérprete de órdenes. Para ello, Raspbian cuenta con el intérprete Python 2.6 instalado por lo que no es necesario instalar nada más. Sin embargo, para hacer uso de funciones avanzadas o cuya release es reciente puede ser necesario la actualización de este paquete mediante el comando "apt-get update python-2.6"

El siguiente paso consiste en la instalación de las librerías de Python para el manejo de los pines de propósito general (en adelante GPIO). Es igualmente posible hacer uso de las librerías para el manejo de los GPIO mediante lenguaje C wiringPI, pero debido al menor estado de madurez de desarrollo y que ofrecen

grosso modo las mismas funcionalidades que las librerías dedicadas de Python se optó por el uso de éstas de manera que fuese posible aprovechar la orientación a objetos del lenguaje y su simplicidad.

```
$ wget
http://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-
0.1.0.tar.gz
$ tar zxf RPi.GPIO-0.1.0.tar.gz
$ cd RPi.GPIO-0.1.0
$ sudo python setup.py install
```

Tras la descarga del paquete, su descompresión e instalado la plataforma ya está lista para la ejecución de programas Python haciendo uso del los GPIO.

Una vez instaladas estas librerías, el primer paso fue programar sobre la herramienta IDLE un pequeño ejemplo de programación de un script Python para verificar el correcto funcionamiento de las librerías. Se basaba en un programa que haría parpadear un led cada segundo.

Para ello se programó el siguiente script denominado led.py :

```
import RPi.GPIO as GPIO
import time
# blinking function
def parpadeo(pin):
        GPIO.output(pin,GPIO.HIGH)
        time.sleep(1)
        GPIO.output(pin,GPIO.LOW)
        time.sleep(1)
        return
# to use Raspberry Pi board pin numbers
GPIO.setmode(GPIO.BOARD)
# set up GPIO output channel
GPIO.setup(11, GPIO.OUT)
# blink GPIO17 50 times
for i in range(0,50):
        parpadeo(11)
GPIO.cleanup()
```

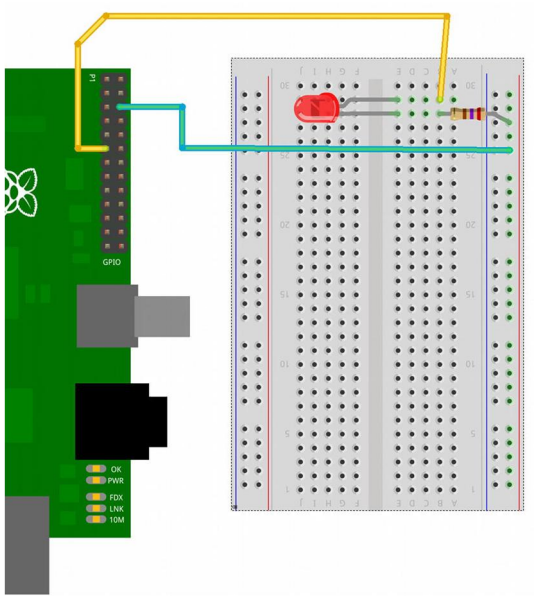
Para su ejecución sobre Raspbian bastaba con utilizar el comando:

```
$ sudo python led.py
```

Haciendo uso del pin 11 e identificando el mismo en el pinout de la placa Raspberry y conectándolo a una placa de pruebas con el siguiente circuito se podía observar como pretendía el funcionamiento del LED.

El montaje ideado y probado corresponde al siguiente esquema (ver Figura 5):

Figura 5 Conexionado Ejemplo LED



2.4.3. Ventajas y desventajas de programación sobre Raspberry PI

Tras analizar la programación en Python sobre Raspberry Pi quedan claras las posibilidades de programación ofreciendo como ventajas:

- Control total a bajo nivel de cada pin de entrada y salida
- Lenguaje de programación sencillo y orientado a objetos.
- Simplicidad de desarrollo.
- Entorno amigable.

Tras analizar los requerimientos en cuanto interfaces por parte de una aplicación robótica (múltiples pines de entrada y salida, control digital de 5V, necesidad de programación múltiples pines de entrada/salida analógicos) se ponen de manifiesto también los siguientes hándicaps de la plataforma:

- La Rpi posiblemente no cuente con los suficientes pines de entrada y salida.

- El nivel digital es de únicamente 3.3 voltios cuando la mayor parte de los sensores comerciales requieren una tensión nominal de 5V.

- Alto requerimiento de alimentación por parte del robot.

- Robustez de la solución en entredicho. Ante una falta de alimentación, el reinicio de la plataforma Raspberry Pi requiere de asistencia.

Uso de wiringPI:

Al igual que mediante Python, existe la posibilidad de llevar a cabo un control sobre los pines de entrada y salida GPIO mediante un programa en lenguaje C++ haciendo uso de la librería wiringPI. Dicha librería asimilada a las utilizadas para el control de las interfaces en Arduino, está publicada bajo licencia GNU LGPLv3. Dicha librería proporciona mediante la función gpio() el control del las interfaces tanto analógicas como digitales y PWM de las que consta la placa RPi.

En este punto se llega a las siguientes conclusiones:

- Existe una limitación evidente en cuanto a disponibilidad de interfaces de entrada y salida utilizando únicamente una placa Raspberry Pi para controlar un robot.

Esta limitación física es fácilmente resoluble. Analizando los puntos fuertes tantos de la plataforma Arduino como de Raspberry Pi se considera ejecutar un control de robot distribuido, utilizando la potencia de cálculo, versatilidad y facilidad de uso de las Raspberry Pi y la gran escalabilidad y diversidad en cuanto a disponibilidad de pines de entrada y salida del Arduino Mega ADK.

2.5. Integración de Arduino y Raspberry Pi.

El primer planteamiento realizado para poder llegar a implementar un control entre ambas plataformas es la necesidad de comunicación entre las mismas. Para ello se consideró implantar una comunicación haciendo uso de los puertos serie de ambas plataformas.

En el caso de la placa Rpi se dispone dentro del pool de pines de propósito general de dos pines TX y RX a 3V. Dichos pines son los GPIO 13 y 14 de la Raspberry pi. En Arduino por otro lado los dos pines del puerto serie dedicado TX0/RX0 y TX1/RX1 funcionan a nivel lógico de 5v.

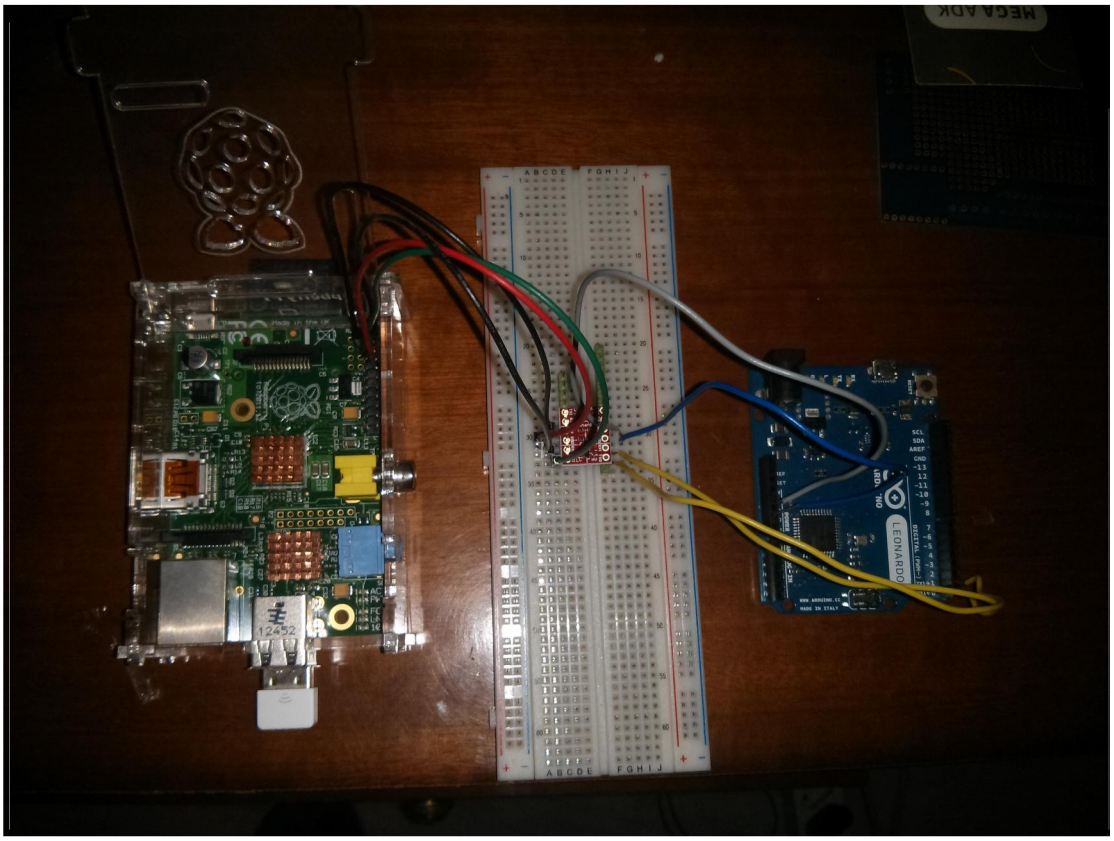
Con estas especificaciones se encuentra un nuevo problema. La diferencia nivel lógico de funcionamiento para los puertos serie entre ambas placas provoca lo siguiente:

- Es posible la comunicación entre Arduino y Raspbery pi. La diferencia de tensión es fácilmente adaptable de 5v a 3v mediante un divisor de tensión haciendo uso de dos resistencias de de 1,6 y 3.3 KΩ.

- Existe un riesgo manifiesto de rotura de puertos de Raspberry Pi conectados a Arduino directamente para los que no se haga uso de un divisor de tensión que reduzca el voltaje de entrada en el mismo.

El montaje a realizar para la comunicación en ambos sentidos es la siguiente (ver Figura 6).

Figura 6 Conexionado intercambio por puerto Serie



Donde en el centro se puede encontrar el convertidor de niveles lógicos. Es necesaria la conexión a cada lado a cada nivel lógico, LOW (en este caso 3.3v) y HIGH (5v). Una vez llevado a cabo este montaje ambas placas estarán preparadas para su comunicación mediante puerto serie.

El siguiente paso ha sido la realización de pruebas para controlar el funcionamiento del paso de mensajes mediante puerto serie.

2.5.1. Ejemplo de programación

Se ha programado tanto en Arduino como en Raspberry Pi mediante lenguaje Python, y la librería GPIO descrita anteriormente el siguiente programa el cual pretende el envío de caracteres y datos mediante puerto serie desde el Arduino y la Raspberry y el posterior respuesta de manera que una vez recibido el carácter H sea encendido un Led mientras que si se recibe un carácter L éste se apague.

Se programa el siguiente script a cargar en Arduino haciendo uso de la herramienta Arduino IDE (entorno de desarrollo):

```
void setup () {
Serial.begin(9600); //Inicializo el puerto serial a 9600
baudios
pinMode(13, OUTPUT)
}

void loop () {
if (Serial.available()) { //Si está disponible
char c = Serial.read(); //Guardamos la lectura en una variable
char
if (c == 'H') { //Si es una 'H', enciendo el LED
Serial.println("'H' pulsada");
digitalWrite(13, HIGH);
} else if (c == 'L') { //Si es una 'L', apago el LED
Serial.println("'L' pulsada");
digitalWrite(13,LOW);
} else {
Serial.println("Caracter no reconocido");
}
}
valor=random(255);//asignamos valor aleatorio
valor = map(valor, min, max, 0, 255);
Serial.println(valor);
}
```

La función Map adapta el valor argumento a un tamaño byte.

Para la transmisión de valores de datos de otro tipo tales como int, float, long etc se debe de programar un algoritmo de transmisión de mensajes basado en incluir el valor en un array de bytes de manera que en el otro extremo el valor pueda ser reconstruido de igual manera.

No existe una función en las librerías disponibles tanto para Arduino como para C++ ni Python para la transmisión/recepción de un dato de tipo distinto a byte.

La compleja programación de un protocolo para el intercambio de un dato además de la necesidad de envío de cabeceras de manera que los datos puedan ser perfectamente reconstruidos en el extremo remoto supone un problema a la hora de programar un algoritmo de control de robot.

Por otro lado, en el control de robot se requerirá de la programación de interrupciones, debido a la naturaleza de tiempo real de la aplicación de robot por lo que un control asíncrono de la transmisión de datos no resulta una solución fácilmente integrable.

La escritura de software para robots al incrementarse el alcance, funcionalidades y escalabilidad de los mismos se puede convertir en una tarea

muy laboriosa y compleja. Además existen una gran cantidad de tipos de robots, con un hardware muy diverso de manera que la adaptabilidad de las comunicaciones, control no es un asunto en absoluto trivial.

En este punto, se analiza en el presente proyecto la posibilidad de utilizar una plataforma integrable sobre la placa Raspberry Pi, instalable sobre la distribución Raspbian que permita proporcionar soluciones middleware que permitan una sencilla integración de hardware en el sistema de control de robot, un entorno de desarrollo sencillo y amigable, así como funcionalidades de abstracción hardware para ello. En este sentido se analiza el desarrollo del software para el robot objeto de este proyecto haciendo uso de la plataforma ROS (Robot Operating System).

3. ROS (Robot Operating System)

3.1. Desarrollo de software para robots mediante ROS.

En la plataforma inicialmente comentada basada en la implantación de un controlador de robot distribuido integrando y comunicando las placas Raspberry Pi y Arduino se plantea la posibilidad de utilizar ROS como marco de desarrollo y herramienta middleware para ello. Desde este sistema operativo instalable sobre Raspbian se establecería la abstracción hardware necesaria para sobre todo implantar las comunicaciones asíncronas requeridas por este tipo de aplicaciones así como posibilidades de depurado de software y pruebas.

3.1.1. ROS : Descripción

Ros es un sistema operativo orientado a la programación de robots desarrollado en 2007 por parte del Laboratorio de Inteligencia Artificial de la Universidad de Standford como soporte al proyecto AI Robot STAIR.

3.1.1.1. Historia

El sistema operativo ROS tiene como plataforma de soporte y desarrollo primaria Ubuntu. Sin embargo, otras plataformas basadas en arquitecturas de procesamiento genéricas tales como ix86 ARM etc son soportadas igualmente aunque su periodo de mantenimiento suele ser limitado.

La aparición de nuevas distribuciones de ROS ha venido marcada por la integración de nuevos componentes, la corrección de bugs y la interacción plug-and-play de diferentes robots comerciales de distintos fabricantes.

Una distribución de ROS se compone de una serie de stacks versionadas de la plataforma. Esto incluye las nuevas versiones de paquetes desarrolladas por al comunidad de developers global. Por ejemplo, podemos decir que uan distribución puede estar formada por:

- ROS stack de core
- Nueva release x.x de common_msgs (paquete)
- Nueva release de x.x de visualization_msgs (paquete)
- Etc…

El historial de publicación de distribuciones es el siguiente:

- May 2014 (expected), Indigo Igloo
- September 4, 2013 - Hydro Medusa

- December 31, 2012 - Groovy Galapagos
- April 23, 2012 - Fuerte
- Aug 30, 2011 - Electric Emys
- March 2, 2011 - Diamondback
- August 3, 2010 - C Turtle
- March 1, 2010 - Box Turtle
- January 22, 2010 - ROS 1.0

3.1.1.2. Utilidad

El diseño de software para robots es complejo, particularmente si el alcance, funciones y la escala de robots siguen expandiéndose. Además cada robot puede tener muy diverso hardware haciendo que la reutilización de código fuente no sea para nada trivial. A parte, para ciertas funciones auxiliares del robot necesarias dentro de un control, tal como algoritmos de navegación comunicaciones con sensores o cualquier otra función a nivel de driver etc puede requerir de un software demasiado pesado para su ejecución en unos tiempos de control específicos.

Desde el punto en que los tiempos de desarrollo necesarios son cada vez más pequeños y los requerimientos en cuanto a complejidad puede estar por encima del nivel de conocimiento de los investigadores que llevan a cabo el proyecto se hace necesario un esfuerzo para el desarrollo dentro del marco de la programación de robots de un sistema soporte para la integración.

Dicho esfuerzo fue llevado a la práctica por los investigadores de la universidad de Standford desarrollando un variado conjunto de prototipos de desarrollo software para abstraer de la complejidad y facilitar la integración de hardware y software. Dichos sistemas supusieron la creación de gran cantidad de aplicaciones robóticas experimentales y académicas utilizadas en la enseñanza y en la industria.

ROS surge como el resultado del énfasis en conseguir un marco de trabajo cuyo objetivo primordial fuese el de la optimización de la complejidad y abstracción, así como la integración de nuevos desarrollos y software en un ámbito de evolución de las aplicaciones robóticas cada vez más amplio.

3.1.2. Objetivos de diseño

En este sentido ROS fue diseñado para cumplir una serie de requisitos demandados en el diseño de aplicaciones robóticas de gran escala. Estos requisitos fueron definidos por los diseñadores de la aplicación de la siguiente manera:

- Facilidades peer-to-peer.
- Multi-lenguaje: se trata de una plataforma que soporta software Python y C++
- Ligero
- Gratis y de código abierto.

En cuanto a las facilidades peer-to-peer un sistema construido y basado en ROS consiste en un conjunto de procesos integrando diferentes host, conectados en tiempo de ejecución en una topología peer-to-peer.

En un principio, las comunicaciones basadas en un servidor central son también posibles y se trata de una topología igualmente válida, aunque puede presentar problemas en un entorno de redes heterogéneo.

En el proyecto descrito en el presente documento se considera la implementación de servicios ejecutados en un PC embarcado (en este caso Raspberry Pi) y otros conectados via puerto serie en Arduino, además se considerará la ejecución de un programa lector de sensores alojado en un dispositivo móvil enlazado con Raspberry mediante un puente WLAN por lo que en caso de utilizar un servidor central se generaría un tráfico a través de esta interfaces innecesario. Dicho problema es solucionado por el entorno distribuido de control que proporciona la topología peer to peer al diseño final.

En cuanto al entorno de desarrollo multilenguaje, al existir varios lenguajes de programación los cuales están comprometidos por diversos requerimientos en cuanto a eficiencia de ejecución, facilidad de escritura, facilidad de debugging así como sintaxis ROS ofrece soporte para los lenguajes más extendidos en el desarrollo de software para robot como son Python, C++, LISP y Octave.

La negociación de la interconexión peer-to-peer se realiza mediante XML-RPC lo que hace dicha funcionalidad accesible para la mayoría de lenguajes.

Además, en lugar de implementar ROS en código C para luego incluir subinterfaces al resto de lenguajes, se implementan todas las funcionalidades de manera nativa para cada uno de los lenguajes soportados. Sin embargo, y según requerimientos de los usuarios y comunidad de desarrollo se han diseñado envolventes de librerías C++ para comprometer la compatibilidad con otros lenguajes, en este caso Octave.

Para el caso de programación cruzada, es decir, programación de varios servicios en varias plataformas en diferentes lenguajes se define un lenguaje neutral de definición de interfaz (IDL) para la descripción de los mensajes a intercambiar entre diferentes instancias o procesos dentro de nuestro sistema. Dicho lenguaje es estándar y las funciones de ejecución nativa son capaces de interpretar esta descripción basada en texto corto descriptivo del tipo de dato dentro del mensaje.

Los generadores de código para cada lenguaje entonces generan una implementación nativa de manera que los datos son automáticamente serializados y deserializado por ROS y las funciones que componen sus librerias al utilizar funciones para el envío y la recepción de los mismos. Esto genera un ahorro considerable en complejidad de programación. Donde en el apartado anterior se refería a la necesidad de dividir los datos en arrays de bytes para su reconstrucción en destino, el uso de las funciones que el IDL y ROS pone a nuestro alcance supone un ahorro de de 137 líneas de código en C++ y 96 en Python.

Por otro lado, el paquete ROS dispone de una serie de herramientas las cuales permiten llevar a cabo las siguientes tareas entre otras:

- Navegar en el árbol de código fuente.
- Obtener y configurar parámetros.
- Visualizar la topología peer-to-peer establecida dentro de un proyecto.
- Medida de ancho de banda utilizado.
- Dibujar gráficamente datos medidos
- Etc.

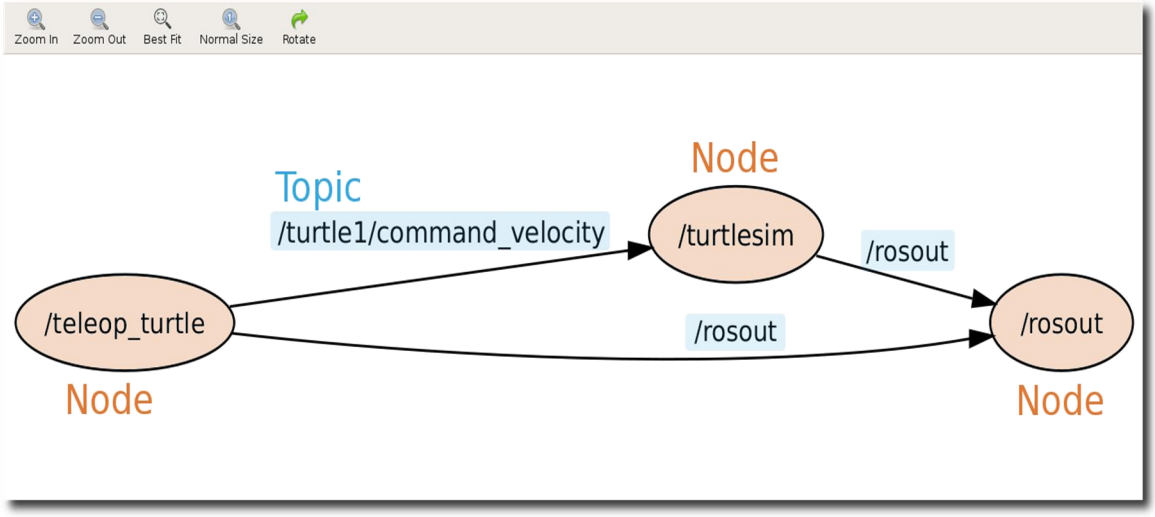
En cuanto a la operación y estructura de funcionamiento de ROS se definen una serie de conceptos básicos necesarios para comprender el funcionamiento del sistema. Éstos son nodos, mensajes, topics y servicios.

3.1.3. Nodos en ROS:

Los nodos son procesos que llevan a cabo para esta aplicación el tratamiento de datos. Por ejemplo, el algoritmo de control del robot. Un sistema está formado por varios nodos. Cada nodo realiza una función específica de control por lo que existe intercambio de datos entre los mismos haciendo uso de los mensajes definidos y disponibles en ROS.

En sistemas formados por varios nodos es conveniente establecer un gráfico mediante la herramienta RXgraph de ROS la cual muestra la estructura de nodos activa en el sistema así como los mensajes intercambiados entre ellos (ver Figura 7)

Figura 7 Ejemplo de establecimiento de Nodos ROS



Los nodos se comunican entre ellos mediante el paso de mensajes. Como ya se ha descrito antes, los mensajes son estructuras de datos estrictas definidas mediante IDL. Los tipo de mensaje estándar son (integer, floating point, boolean etc) así como sus respectivos arrays. También es posible la definición de un tipo de mensaje customizado. Se trata de un tipo de estructura incluyendo mediane lenguaje IDL los tipos de datos necesitados. Por ejemplo, y para el robot que ocupa este proyecto, se ha creado un tipo de mensaje específico en el que se introducen los datos de varios sensores ultrasónicos así como sensores LDR. Estos datos son recibidos por el nodo controlador que los procesa y ejecuta el algoritmo de control del robot.

La difusión de los mensajes se realiza mediante el paradigma publicador-subscriptor. Cada nodo envía un mensaje publicando éste en un topic determinado. Al mismo tiempo, cada nodo capta los mensajes de los topic en los que está interesado según su programación. En cada nodo y en cada sistema en general pueden existir una gran variedad de publicadores y subscriptores concurrentes.

El paradigma publicador-suscriptor supone un esquema de comunicación basado en difusión o "broadcasting" por lo que se trata de un método muy flexible pero que no es adaptable en un entorno puramente síncrono. Sin embargo es idóneo para aplicaciones de control puramente asíncronas y distribuidas como en el caso de un robot.

Para funciones que requieren transacciones síncronas entre las mismas como por ejemplo, un servidor web, se define en ROS lo que se denomina servicios. En el proyecto actual no hay necesidad de uso de ningún servicio determinado por lo que no se entrará en profundidad en su definición.

En el caso del proyecto en curso se definirán varios nodos, situados en un dispositivo móvil Smartphone con sistema operativo Android, en el controlador

alojado en la placa Raspberry Pi y en el sistema Arduino para la lectura de datos de los sensores y establecimiento de señales para los actuadores del robot.

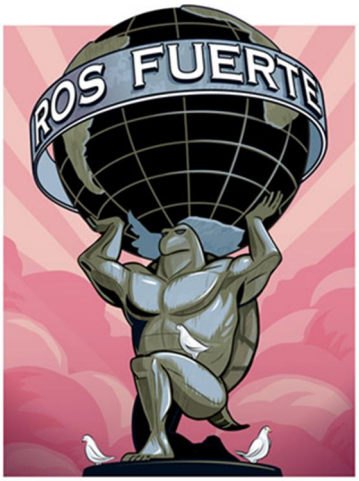
El primer paso dado para establecer el escenario completo ha sido la instalación de ROS sobre Raspbian. El core del sistema se encontrará configurado sobre la RPi. El debugging, establecimiento de mensajes, nodos y comunicación entre los mismos se configurará sobre ROS. Para los nodos periféricos tan solo se hará uso de las bibliotecas de ROS necesarias para el establecimiento de la comunicación (publicadores y suscriptores).

3.2.**Preparación del entorno.**

La instalación de ROS se lleva a cabo por recomendación de la comunidad de desarrollo desde SVN (el nuevo sistema de control de versiones que sustituye a CVS). Esto implica la instalación mediante descarga del código y su posterior compilación en nuestra placa de manera nativa.

En este caso la versión de ROS utilizada ha sido Fuerte. Esta es la quinta distribución publicada por ROS el 23 de abril de 2012. Fuerte incluyó grandes mejoras que hacen más fácil la integración con varios entornos software y herramientas. Las mejoras incluyeron una reescritura del sistema de compilación y linkado así como la migración al entorno QT y una continuación a la transición a librerías "standalone" alguna de ellas utilizadas sobre Arduino en el presente proyecto.

Figura 8 ROS Fuerte-Turtle



El primer paso para la instalación es la configuración de red sobre la placa Rpi.

Tras la configuración de la conectividad de la placa a través de Raspbian, se procederá desde un ordenador a iniciar una sesión SSH que nos permitirá mediante código realizar todas las tareas incluidas en el proceso de instalación

de la placa.

```
ssh -X pi@192.168.1.XX
```

3.2.1. Instalación de librerías, Python y demás utilidades necesarias:

En primer lugar se acomete la instalación de los siguientes paquetes y librerías necesarios:

```
sudo apt-get install build-essential python-yaml cmake
subversion wget python-setuptools mercurial
```

A continuación las librerías de core y las dependencias para el arranque de ROS:

```
sudo apt-get install build-essential python-yaml cmake
subversion wget python-setuptools mercurial git-core

sudo apt-get install python-yaml libapr1-dev libaprutil1-dev
libbz2-dev python-dev libgtest-dev python-paramiko libboost-
all-dev liblog4cxx10-dev pkg-config python-empy swig
```

Por último se debe instalar el paquete Python-nose. Al disponer de la versión Wheezy de Raspbian se debe instalar desde el paquete python directamente mediante apt-get.

```
sudo apt-get install python-nose
```

El siguiente paso sería la instalación de dependencias y librerias gráficas, esto es opcional y necesario únicamente si se planea utilizar las herramientas gráficas de ROS.

```
sudo apt-get install python-wxgtk2.8 python-gtk2 python-
matplotlib libwxgtk2.8-dev python-imaging libqt4-dev graphviz
qt4-qmake python-numpy
```

Por último se debe compilar y enlazar el paquete swig-wx desde la fuente si se requiere usar herramientas gráficas. Swig –wx requiere bison y las herramientas de automake GNU.

```
sudo apt-get install bison++ automake autoconf
git clone https://github.com/ros/swig-wx.git
cd swig-wx
./autogen.sh && ./configure && make && sudo make install
```

Una vez completado este paso estamos en disposición de todas las

herramientas necesarias para la compilación y linkado de ROS en nuestra máquina.

3.3.Instalación de librerías CORE

Los siguientes pasos de instalación requieren la compilación del código ROS en dos capas diferentes:

- Descarga y compilación de la capa de core de las librerías ROS y herramientas en /opt/ros/fuerte.

- Descarga y compilado de las librerías de alto nivel haciendo uso de la herramienta rosmake ya instalada en el paso anterior.

Antes de poder realizar estas tareas se requiere de una herramienta ROS para SVN que permite la instalación o actualización de archivos o documentos. Se trata de una API Python para interactuar con el código fuente establecido en un repositorio de manera que llama al paquete vcstools para el control y el almacenamiento de los mismos en un paquete .rosinstall. Esta herramienta fue desarrollada como complemento de ayuda a la instalación de ROS pero no tiene ninguna dependencia con el mismo.

Una vez instalada mediante pip al no existir versión en los repositorios consultados por apt-get para Debian estamos en condiciones de recoger el código fuente de las librerías de Core de ROS y almacenarlo en /ros-underlay para su posterior compilado.

```
sudo pip install -U rosinstall
```

Este paso crea un sistema de instalación de las librerías CORE y en este caso al tratarse de la versión FULL se agregan elementos adicionales para el debugging y visualización 3D:

```
rosinstall --catkin ~/ros-underlay
http://ros.org/rosinstalls/fuerte-ros-full.rosinstall
```

Desde octubre de 2013, no existe soporte por parte de la comunidad para ROS Fuerte por lo que el repositorio anterior no está disponible. En ese caso, no queda más remedio que la descarga manual del mismo y compilación e instalación manual mediante rosinstall fichero origen –directorio destino.

3.4.Instalación de paquetes de ROS

A continuación se crea el directorio build dentro de ROS underlay: por lo que la compilación e instalación del sistema de instalación creado en el directorio ros-underlay se realiza mediante herramientas CMAKE y make según las listas copiadas en el mismo.

```
cd ~/ros-underlay
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/opt/ros/fuerte
```

Finalmente se compila y se instala el paquete ROS en el directorio /opt/ros/fuerte

```
make -j8
sudo make install
```

3.5.**Configuración de ROS: setup.**

La creación de proyectos y ejecutables se lleva a cabo mediante la creación de paquetes. Para el manejo de paquetes ROS se hace necesario el uso de ciertos comandos de navegación para evitar el tedioso tramite de escritura y búsqueda de carpetas de paquetes continuamente.

Algunos de los comandos más utilizados en el presente proyecto son los siguientes:

```
$ rospack find [package_name]
$ rosstack find [stack_name]
```

Estos comandos permiten obtener información sobre paquetes y stacks. En concreto estos comandos devuelven el directorio donde se encuentra instalado el paquete invocado.

```
$ roscd [packet name]
```

Este comando cambia el directorio de trabajo por aquel en el que se encuentre el paquete de ROS invocado. Dicho comando integrado en el SHELL del sistema Raspbian permite la funcionalidad de autocompletado.

Para el manejo de estos comandos el sistema ROS dispone de un amplio juego de variables de entorno las cuales son utilizadas por los comandos anteriores para realizar sus funcoines. Entre ellas, y tras la instalación de ROS es imprescindible que la ruta del directorio donde se encuentren todos nuestros paquetes y stacks se encuentre dentro de la variable de entorno definida como ROS_PACKAGE_PATH.

Los paquetes creados o instalados mediante gestores de paquetes o haciendo uso de las herramientas rosw rosbuild contienen un archivo setup.sh generado por los mismos.

Mediante el comando source será posible usar los comandos de ROS en el SHELL de Linux. En este caso:

```
$source /opt/ros/<distro>/setup.bash
```

Añadiendo está línea al archivo .bashsrc se tendrá acceso siempre a dichos comandos sin necesidad de introducirlos cada vez que se abre un terminal.

Este comando asegura que el directorio de raíz de ROS se encontrara dentro de la variable de entorno por lo que todo paquete incluido en este podrá ser manejado por los comandos ROS.

Para comprobar el contenido de la variable basta con ejecutar la orden:

```
$Echo $ROS_PACKAGE_PATH
```

3.6.Configuración del entorno para programación

3.6.1. Configuración de interfaz de red WLAN.

El siguiente paso seguido, implica para mayor comodidad a la hora de llevar a cabo la gestión de la placa Raspberry y la programación de código y gestión del entorno de programación la dotación de conectividad wifi para la misma. Para ello se ha utilizado el módulo Wifi EDUP Ultra-Mini Nano USB 2.0 cuya imagen se observa en la Figura 9.

Figura 9 Microadaptador Wifi compatible

Este adaptador conectado a uno de los puertos USB de Raspberry pi proporcionará conectividad wifi de manera que sea posible la apertura de sesiones SSH para el manejo de la placa sin necesidad de conectividad a través de

cable Ethernet. Este adaptador además dotará de movilidad al sistema una vez integrado en el robot en cuanto a la recepción de datos en un entorno WLAN.

Para la configuración de red haciendo uso de este dispositivo se editó el archivo /etc/network/interfaces:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet dhcp
        wpa-ssid "Red WLAN que proceda"
        wpa-psk "contraseña que proceda"
```

Tras guardar será necesario reiniciar la interfaz Wlan0 mediante:

```
$ ifdown Wlan0
$ ifup Wlan0
```

Mediante el comando ifconfig podremos consultar todos los parámetros de configuración incluida la dirección IP obtenida mediante DHCP a la que deberemos conectarnos.

3.6.2. Establecimiento de workspace y paquete ROS

Para el desarrollo del software controlador del que se verá comprendido el sistema controlador del robot es necesario establecer para el proyecto a llevar a cabo el establecimiento de un espacio de trabajo. Será este entorno donde se trabajará de manera más cómoda sobre código ROS.

La herramienta de ROS que provee una interfaz para comandos de control de código fuente como SVN, git y Mercurial y que permite la gestión de estos para la instalación de paquetes ROS se denomina ROSWS. Dicha herramienta es parte del paquete rosinstall el cual fue implantado en Raspbian e integrado con ROS durante la fase de instalación anteriormente descrita.

Para la creación de un entorno de trabajo se utiliza la herramienta rosws inicializando un nuevo directorio (se utilizará de nombre fuerte_workspace) como entorno de trabajo.

```
rosws init ~/fuerte_workspace /opt/ros/fuerte
```

Este comando inicializa en el directorio fuerte_workspace alojado directamente en /root un espacio de trabajo referenciado al directorio de instalación de ROS (en este caso fue /opt/ros/fuerte)

Es recomendable una vez se inicializa el entorno de trabajo, reservar una carpeta donde crearemos todos los paquetes de los que se componga el proyecto. Dicha carpeta tendrá el nombre de Sandbox

Una vez que inicializamos nuestro workspace se generan los archivos setup.sh y setup .bash. Mediante el comando source hacemos que este directorio creado y gestionado por rosws sea incluido en la variable de entorno descrita anteriormente ROS_PACKAGE PATH para poder ser reconocidos todos los paquetes contenidos en el mismo por los comandos de navegación de ROS.

```
mkdir ~/fuerte_workspace/sandbox
rosws set ~/fuerte_workspace/sandbox
source ~/fuerte_workspace/setup.bash
```

De esta manera podemos ir añadiendo paquetes mediante rosws dentro de la carpeta /fuerte_workspace/sandbox si necesidad de tener que utilizar el comando source de nuevo.

3.6.3. Creación de paquete objetivo del proyecto

El siguiente paso es la creación del paquete objetivo de nuestro proyecto. Los paquetes de ROS consisten en una serie de archivos

- Manifest
- CMakeLists.txt
- Mainpage.dox
- Makefiles.

Así como la las carpetas /src donde definiremos nuestro programa a construir así como /msg donde definiremos mensajes customizados a intercambiar entre los diferentes nodos del sistema.

La herramienta roscreate-pkg permite mediante una simple instrucción generar todos los archivos y las estructuras necesarias para la definición de un paquete sin el tedioso trámite de generación a mano. Además ofrece la posibilidad de establecer dependencias en los archivos Cmake y Makefile con otros paquetes ROS para la utilización de funciones específicas.

Una vez en el directorio /sandbox crearmos nuestro paquete denominado “proyecto” con el siguiente comando:

```
# roscreate-pkg proyecto std_msgs rospy roscpp
```

La inclusión de rospy y roscpp así como std_msgs nos permite tener acceso para todos los ficheros fuente de nuestro fichero a funciones core de ros

definidas tanto para python (rospy) como para c++ (ros cpp) así como la definición de mensajes estándar y mapeables con tipos de variables de C++ (int8, float32, byte, double, etc).

Se obtiene el siguiente resultado:

```
Creating package directory ~/fuerte_workspace/sandbox/proyecto

Creating include directory
~/fuerte_workspace/sandbox/proyecto/include/proyecto

Creating cpp source directory ~/ros/ros_tutorials/proyecto/src

Creating python source directory
~/fuerte_workspace/sandbox/proyecto/src/proyecto

Creating package file
~/fuerte_workspace/sandbox/proyecto/Makefile

Creating package file
~/fuerte_workspace/sandbox/proyecto/manifest.xml

Creating package file
~/fuerte_workspace/sandbox/proyecto/CMakeLists.txt

Creating package file
~/fuerte_workspace/sandbox/proyecto/mainpage.dox

Please edit proyecto/manifest.xml and mainpage.dox to finish
creating your package
```

El archivo manifest juega un importante papel. En él se define como los paquetes son construidos, ejecutados y documentados.

Para asegurar que ROS reconoce nuestro paquete, es decir, todas las variables de entorno contienen su ruta por lo tanto podemos hacer uso de todas las funciones de ROS para el manejo y la gestión del mismo, hacemos uso de la herramienta rospack a la que añadiéndole el atributo find y profile nos muestra información relevante del mismo como elementos ejecutables y directorio en el que se encuentra.

3.6.4. Ejecución de Roscore

El objetivo primordial y principal valor añadido por ROS en la programación de robots es el establecimiento del middleware que permite la comunicacion entre nodos.

Para este cometido ROS cuenta con el proceso roscore. Éste es una colección de nodos y programas requeridos para el establecimiento de nodos de comunicación en un sistema basado en ROS. Siempre que se requiera establecer una comunicación entre dispositivos o distintos nodos corriendo en la misma maquina es necesaria la ejecución de un proceso roscore paralelamente.

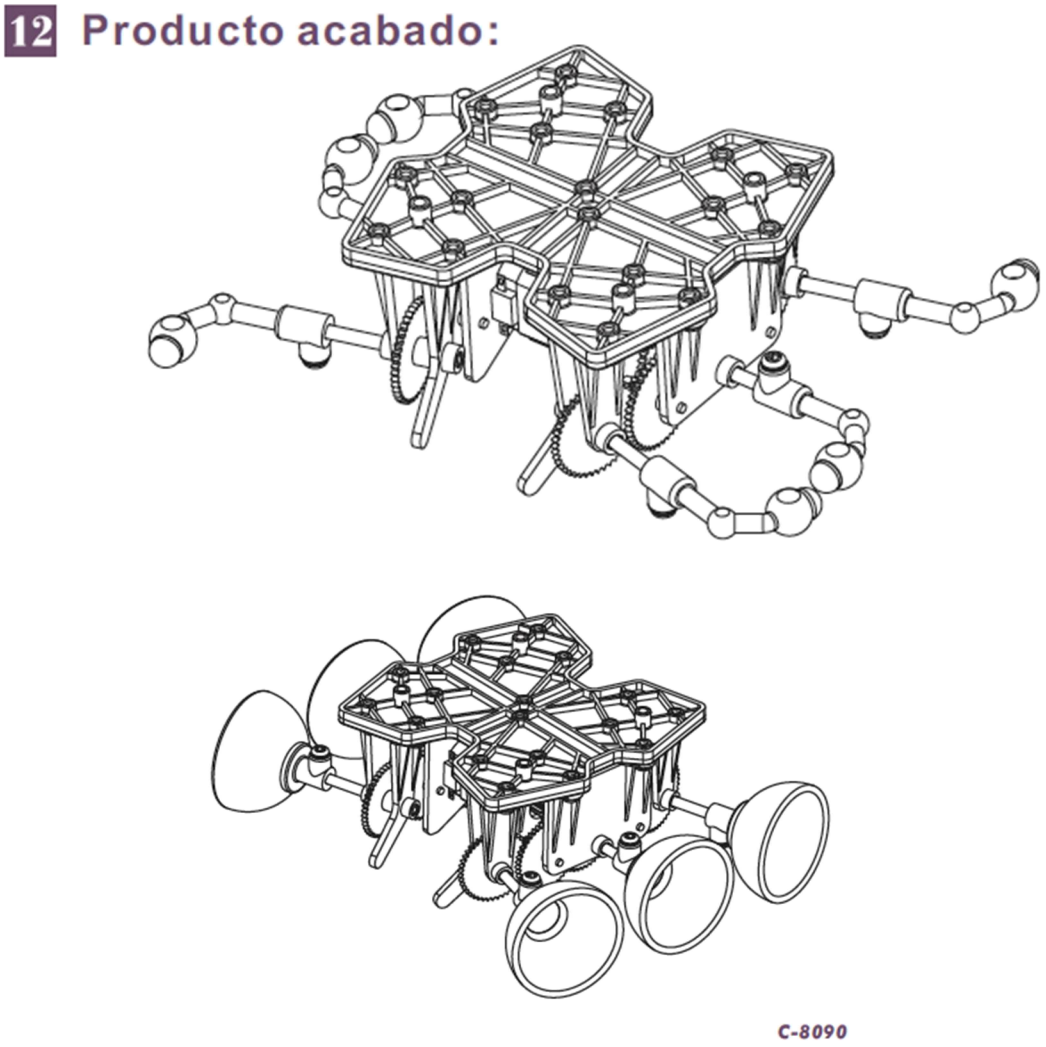
4. Robot objetivo

4.1.Descripción

El robot objetivo utilizado para las funciones a desarrollar en el presente proyecto será un robot móvil de 6 ruedas o patas al ser este configurable. Se trata de un grupo motriz fabricado por CIC Creative (CEBEKIT). Contiene un grupo de motor doble con engranaje reductor, controlados independientemente. Existe la posibilidad de configuración mediante motricidad basada en patas circulares o ruedas.

El modelo c-8090 objetivo del proyecto es el siguiente observable en la fi Figura 10.

Figura 10 Robot Objetivo



Este robot dispone de dos motores con una velocidad de salida de 170 rpm nominal y con voltaje de entrada de 5V. No se dispone de ruedas directrices por lo que todo giro deberá realizarse sobre su mismo eje aplicando fuerzas en sentido contrario en ambos ejes de tracción.

El robot en cuestión dispone de una superficie plana donde colocaremos toda la electrónica embarcada la cual llevará a cabo las diferentes funciones contempladas.

Los motores del robot son alimentados por 4 baterías alcalinas recargables de 1.5 voltios en serie cada uno. Dichas baterías se encuentran integradas mediante soportes portabaterías lo cual permite su integración sobre el robot de una manera compacta como se puede observar en la Figura 11.

Figura 11 Alimentación Robot

4.2.**Modelos de control implementados**

Las funciones que se han ensayado para el presente proyecto son las siguientes:

- Modelo 1 :

Control remoto mediante el intercambio de mensajes ROS. El control se realiza haciendo uso de los sensores giróscopos de cualquier teléfono Android. Controlando la inclinación en los ejes x e y de nuestro Smartphone y en función del grado de ésta se controlaría el movimiento del robot.

- Modelo 2:

Funcionamiento autónomo mediante algoritmo de evitación de obstáculos. El robot en función de la intensidad luminosa en el ambiente arrancaría una marcha hacia delante ejecutando un algoritmo que permita evitar la colisión con cualquier obstáculo que éste se encontrase en su camino

4.3.**Solución empleada**

Para el desarrollo de ambos modelos y funciones se hará uso de las facilidades que la plataforma ROS ofrece. Entre ellas, la utilizada será la facilidad de middleware de comunicaciones entre las dos plataformas Raspberry Pi y Arduino Mega ADK.

El sistema estará conformado por una serie de publicadores y suscriptores de varios topics para el intercambio de datos de sensores previa transformación en Arduino, su procesamiento en Raspberry Pi y publicación de comandos de control por parte de esta los cuales servirán para el control de los dos motores del robot.

El primer paso necesario por tanto para este cometido es el establecer una interfaz de comunicación mediante ambas plataformas. La interfaz más sencilla y a su vez aprovechable es la de conectar ambas plataformas a través de sus interfaces USB.

Uno de los problemas de la placa Arduino es la necesidad de alimentación. La placa requiere de una tensión de entrada recomendada de 7 a 12 voltios. Sin embargo, durante los experimentos realizados en el presente proyecto, se ha comprobado que los 5 voltios y 100mA provistos por la placa Raspberry PI a través del puerto USB utilizado para la comunicación serial con Arduino son suficientes para la operación de esta placa.

Alimentando de esta manera, podemos conseguir colocar ambas placas en funcionamiento sin necesidad de una demanda de fuente de tensión externa para ello.

4.4.**Comunicación entre placas (ROSSERIAL)**

Una vez definida la interfaz serie USB se requiere la instalación de la utilidad ROS que permite la activación de la interfaz USB así como la aplicación

que permite que todo dato se publicado no solo en el entorno ROS sino de igual modo a través de las interfaces activas para ello en este caso la interfaz serie.

Para este cometido y tras evaluar las diferentes opciones disponibles se opta por la utilización del paquete Rosserial.

4.4.1. Descripción

ROSserial es una aplicación que permite la comunicación punto a punto sobre puerto serie. Su aplicación primaria es la integración con plataformas microcontroladores de bajo coste (en este caso, Arduino Mega ADK). Consiste en el establecimiento de un protocolo p2p y generación de las librerías para el uso de funciones ROS en sketchs de Arduino tanto en Python como C++.

En este caso se generará una librería denominada rosserial_arduino el cual dará soporte de funciones ROS en esta plataforma.

4.4.2. Instalación en el entorno de trabajo

En primer lugar instalaremos ambos paquetes en nuestra plataforma Raspberry PI. En este caso se llevará a cabo la instalación de archivos binarios. Al tratarse la versión de ROS instalada (Fuerte) de una release no reciente (aunque estable sobre Raspberry Pi) no consta de soporte para su instalación mediante SVN y código fuente.

```
sudo apt-get install ros-fuerte-rosserial-arduino
sudo apt-get install ros-fuerte-rosserial
```

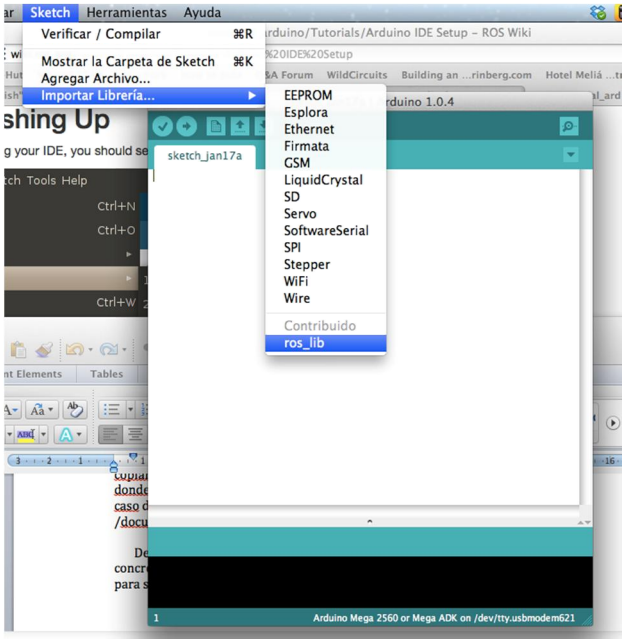
4.4.3. Generación de librerías para Arduino

Una vez instalados ambos paquetes, se generará una carpeta denominada ros_lib/. En la misma se contienen las librerías que deben ser copiadas en el directorio /Arduino/libraries para el reconocimiento de librerías por parte del entorno de desarrollo Arduino IDE cuya interfaz se muestra en la Figura 12.

```
roscd rosserial_arduino/src
```

De esta manera será posible exportar todas las funciones de ROS y en concreto aquellas que nos permiten crear publicadores y suscriptores de “topics” para su programación sobre Arduino.

Figura 12 Entorno de programación Arduino IDE



En este caso es necesario realizar varios cambios en la librería ArduinoHardware.h para su compilación y carga en Arduino Mega. Es necesario cambiar la interfaz sobre la que asignar el stream de datos de HardwareSerial a Serial.

Una vez realizado este paso, cualquier publicador o suscriptor ejecutado en Arduino podrá empezar a recibir/ enviar datos leyendo el stream transmitido por puerto serie USB.

4.4.4. Inicialización de comunicación serial

Para inicializar la interfaz USB como puerto serie y permitir que el servicio roscore establezca en ésta el puente entre el nodo Master (Raspberry Pi) y el nodo corriendo en Arduino es necesario ejecutar el siguiente proceso con los siguientes argumentos:

```
rosrun rosserial_python serial_node.py /dev/ttyUSB0
```

Queda abierto el proceso que establece un nodo virtual sobre el que se publicarán y se "escuchará" por parte el ROS_MASTER los topics pertinentes.

5. Modelo 1: Control remoto mediante Smartphone Android.

5.1. Descripción.

El objetivo de este control se basa en la publicación de los datos medidos por el sensor giróscopo de un Smartphone Android a través de una interfaz WLAN.

La solución se basa en un nuevo driver instalable en un sistema Android que conecta los sensores de un dispositivo Android a un entorno ROS. La aplicación publica en un topic denominado sensor_msgs/Imu y sensor_msgs/Image los datos leídos de los sensores acelerómetros, giróscopos, magnetómetros y cámara. La aplicación en cuestión tiene como nombre Android Sensor Driver y fue creada por Chad Rockey y se encuentra disponible en google play.

Esta aplicación ofrece en lenguaje java la capacidad de generar un publicador para el topic IMU el cual contendrá los datos del acelerómetro que necesitará el control implantado en Raspberry Pi para el control de los motores del robot.

Una vez instalada la aplicación en el Smartphone utilizado en el presente escenario (Google Nexus 4) se requiere la configuración del escenario.

5.2. Configuración del escenario

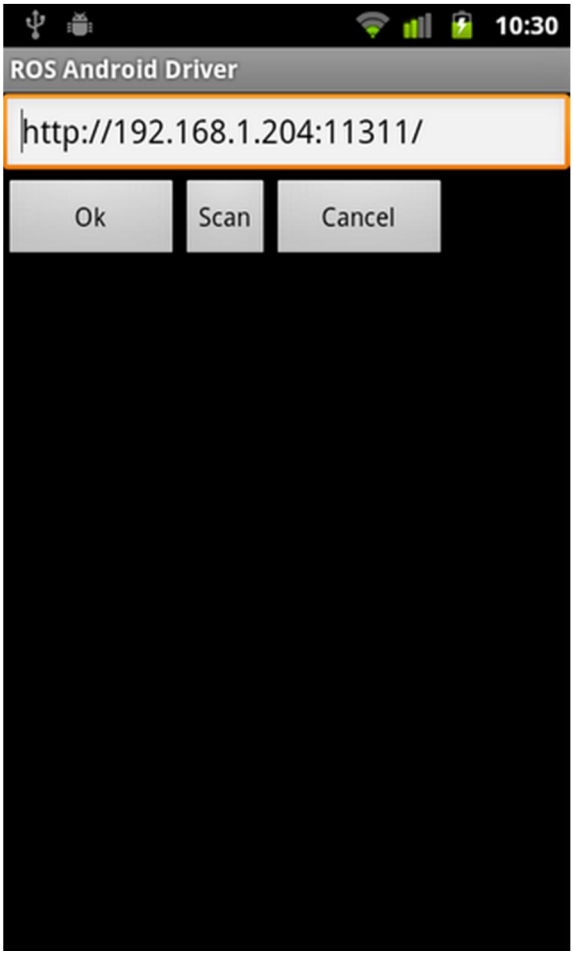
Para ello se deberán configurar correctamente la variable de entorno ROS_MASTER_URI. Esta variable contiene información sobre donde los distintos nodos del sistema (en este caso el la aplicación Android Sensor Driver corriendo en nuestro Smartphone) puede localizar el nodo master donde se encuentra corriendo el sistema CORE de ROS (en este caso el proceso roscore).

Se deberá exportar la IP obtenida por la interfaz WLAN de Raspberry pi mediante ifconfig.

```
$exportROS_MASTER_URI=http://192.168.1.204:11311/
```

La IP local WLAN de nuestra placa es la .204. Acto seguido se deberá ordenar a la aplicación en Android que comiencen a emitirse los datos al máster existente en la siguiente dirección en la interfaz mostrada en la Figura 13

Figura 13 Interfaz Ros Android Driver y Nexus 4



5.3.**Pruebas**

Para probar este escenario es posible realizar una prueba muy sencilla basada en la inicialización de la aplicación en android. Una vez nuestra Raspbery pi consta de una interfaz WLAN levantada en la misma red SSID que nuestro Smartphone, los datos se empezarán a publicar hacia la IP ROS Master. El escenario planteado es el reproducido en la Figura 14.

Para comprobar que dichos datos efectivamente se están publicando correctamente basta con iniciar el proceso roscore.

Una vez inicializado roscore será posible visualizar los topics publicados mediante la herramienta rostopics. Si ejecutamos:

```
$rostopic list
```

obtenemos el siguiente resultado:

```
Published topics:
* /android/Imu  1 publisher
* /rosout [roslib/Log] 2 publishers
* /rosout_agg [roslib/Log] 1 publisher

Subscribed topics:
* /rosout [roslib/Log] 1 subscriber
```

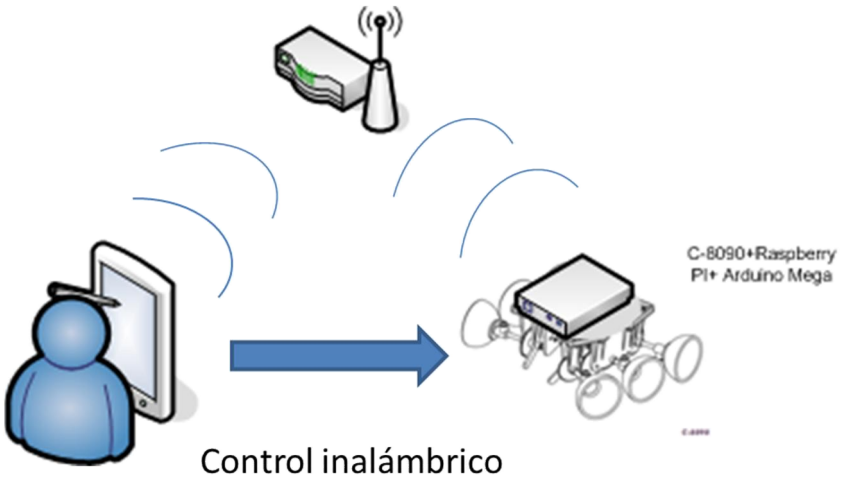
Se comprueba cómo se está publicando el topic android/Imu relacionado con los sensores del smarphone.

Para inspeccionar la correcta conexión del nodo Android desde donde se está publicando el topic basta con utilizar la herramienta rosnode de la siguiente forma:

```
$rosnode list
$rosnode ping /android_sensors_driver

rosnode: node is [/android_sensors_driver]
pinging /android_sensors_driver with a timeout of 3.0s

xmlrpc reply from http://192.168.1.223:48117
time=53.053141ms

xmlrpc reply from http://192.168.1.223:48117
time=115.639925ms

xmlrpc reply from http://192.168.1.223:48117
time=122.823000ms
```

Una vez configurado el extremo publicador del estado de cada uno de los sensores a utilizar se está en disposición de desarrollar el programa C++ controlador del robot.

Figura 14 Escenario planteado



5.4.**Solución implementada**

El objetivo cualitativamente es el de programar un suscriptor en C++ que recoja los datos obtenidos del topic /Android/Imu y que aplique en su función de callback la llamada a un publicador que transmita por el puerto serie haciendo uso del paquete rosserial el comando de velocidad adecuado para cada motor.

La funcionalidad final programada será la de controlar mediante la aceleración lineal detectada en el Smartphone el movimiento y velocidad en función el grado de inclinación de cada motor tal como se establece en el esquema introducido en la Figura 15.

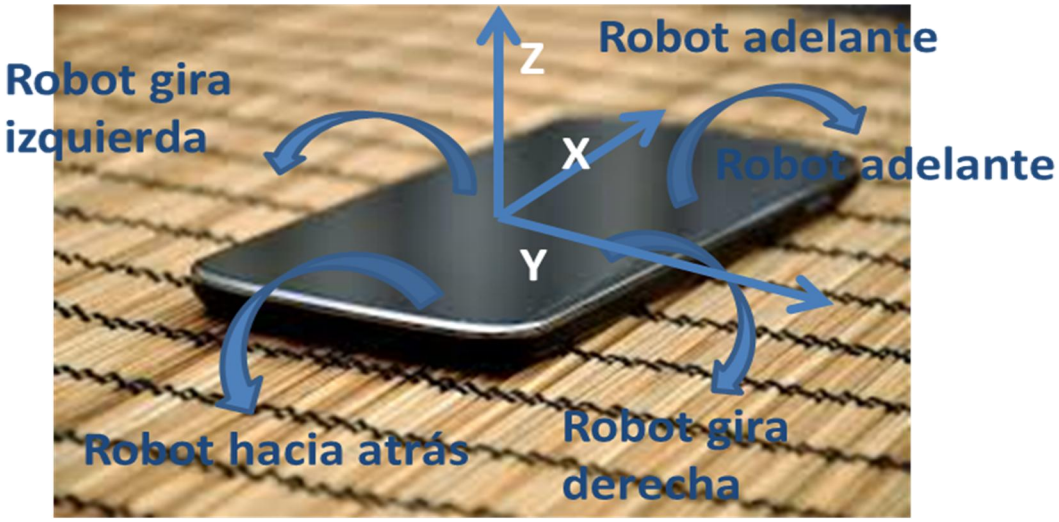
- Hacia delante: En ese caso inclinando el Smartphone hacia delante, giro entorno al eje y en sentido de las agujas del reloj, ambos motores entrarían en funcionamiento a partir de una cierta medida de aceleración lineal. De este modo el robot se movería con ambos motores girando a la misma velocidad hacia adelante.

- Hacia atrás: Inclinando el smarphone hacia atrás, giro entorno al eje y en sentido opuesto al de agujas del reloj, los motores entrarían a girar en sentido contrario a la misma velocidad en función de la inclinación detectada por los sensores.

- Giro a la izquierda: Inclinando el smarpthone sobre el eje x y en sentido opuesto a las agujas del reloj, giraría el motor derecho hacia adelante mientras que el izquierdo lo haría hacia atrás a la misma velocidad y en función de la inclinación detectada por los sensores.

- Giro a la derecha: Inclinando el smarpthone sobre el eje x y en sentido de las agujas del reloj, giraría el motor derecho hacia atrás mientras que el izquierdo lo haría en sentido opuesto a la misma velocidad y en función de la inclinación detectada por los sensores.

Figura 15 Algoritmo implementado

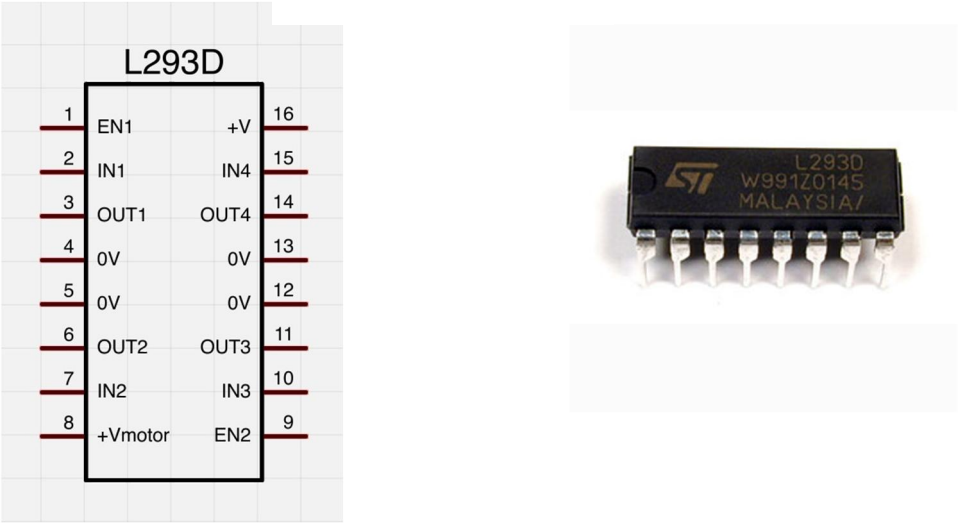


5.4.1. Driver para motores y puente H.

Como "driver" para ambos motores se hace uso en el presente proyecto del circuito L293D de en este caso ST (ver Figura 16).

Se trata de un chip puente H muy útil y que permite el control de dos motores independientemente. El pinout del circuito y el conexionado final del mismo se encuentran expuestos en la siguiente imagen:

Figura 16 Integrado L293D



El chip consta de un pinout el cual recoge:

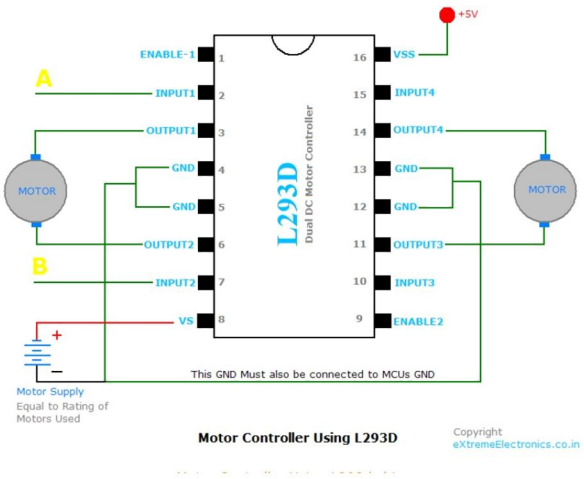
- 2 pines Enable, que habilitan el paso de potencia hacia los pines

output. Estos dos pines (1 para cada motor) estarán conectados a uno de los puertos PWM de Arduino. Será la señal PWM en función del duty-cicle marcado por Arduino mediante hardware, la encargada de trasladar

- 4 pines IN: que irán conectados a salidas digitales de Arduino y que marcan con un nivel lógico de +5V (HIGH) 0V (LOW) el sentido de giro de cada uno de los motores.
- Pin V+ debe marcar la referencia de nivel lógico alto por lo que debe ir conectado a unos de los pines de 5v de Arduino.
- El pin Vmotor provee de potencia al circuito para distribuir a los dos motores. Este pin por tanto debe ir conectado a los dos porta baterías de 5 4,5 V cada uno conectados en serie por lo que el voltaje total conectado es de 9V.

El conexionado a realizar es el establecido en la Figura 17.

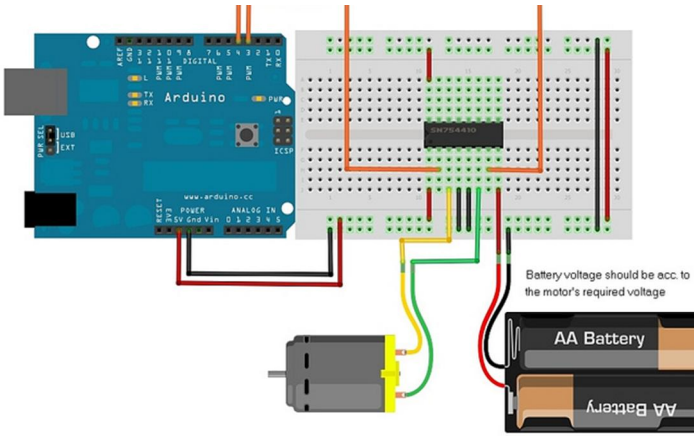
Figura 17 Conexionado al motor



5.5.**Conexionado**

El conexionado del escenario se realizará de la siguiente manera (ver Figura 18).

Figura 18 Conexionado físico



Una vez resuelto el conexionado se lleva a cabo el desarrollo del programa en C++ a ejecutar tanto en Raspberry Pi en el entorno ROS como en Arduino Mega para la recepción de comandos de control hacia los motores.

5.5.1. Programa

En primer lugar se define el programa ejecutado en el nodo Master ROS. Este programa tiene como cometido la lectura del topic Android/Imu del que se extraerá el dato de aceleración lineal.

Posteriormente este dato será interpretado por un algoritmo el cual establecerá la opción de movimiento en cada momento en función de los valores leídos. Por otro lado la velocidad se establecerá de igual manera teniendo en cuenta de igual modo el valor comprendido entre 0 y +/- 9,8 m/s medida y publicada por el sensor.

Para ello se define en el archivo Project.cpp una clase de C++ para hacer accesible los distintos publicadores dentro de las funciones de Callback de los suscriptores. Se definen los publicadores y suscriptores por tanto como instancias públicas dentro de la clase.

```
class miclase
{
public:

  miclase()
  {

mov=n.subscribe("android/imu",1,&miclase::movCallback,this) ;
    veloci=n.advertise<std_msgs::Float32>("velocidad",1);
    opt=n.advertise<std_msgs::Float32>("opcion",1);
  }
```

Se establece en este caso un suscriptor denominado “mov” el cual recibirá los datos publicados en el topic “Android/imu” y que llamará a la función de callback definida dentro como privada dentro de la clase “miclase”. Ésta es “movCallback”. Por otro lado, se tienen otros dos publicadores “veloci” y “opt” los cuales publicarán en sendos topics “velocidad” y “opción” los parámetros necesarios para la interpretación por parte de Arduino de cara al manejo en cada momento de los motores.

Una vez creados estos topics, se definen como variable privada (o sea, solo accesible desde objetos de la misma clase) la función de Callback para el suscriptor “mov”

```
void movCallback(const sensor_msgs::Imu& imu_msg)
  {
    std_msgs::Int8 option;
    std_msgs::Float32 velocid;
    float x,y,z;
    x=imu_msg.linear_acceleration.x;
    y=imu_msg.linear_acceleration.y;
    z=imu_msg.linear_acceleration.z;
    float xa,za,ya;
    xa=abs(x);
    ya=abs(y);
    za=abs(z);
    int opcion=0;
    float velocidad=0;
    if(x>0 && xa>0.4 &&  xa>=ya)
      {opcion=1;velocidad= xa*26.02;} //gira a la izquierda
    else{
      if(x<0 && xa>0.4 &&  xa>=ya)
     {opcion=2;velocidad=xa*26.02; } //gira a la derecha
    }
    if(y>0 && ya>0.4 &&  ya>=xa)
      {opcion=3;velocidad=ya*26.02;} //hacia atras
    else{
      if(y<0 && ya>0.4 && ya>=xa)
     {opcion=4;velocidad=ya*26.02; } //hacia delante
    }
    /*if(z>0  && za>xa && za>ya)
  {opcion=0;velocidad=0;} //quieto
else{
if(z<0 && za>2 && za>xa && za>ya)
  {opcion=0;velocidad=0;}
  } //quieto*/

    option.data=opcion;
    velocid.data=velocidad;

    opt.publish(option);
    veloci.publish(velocid);

  }
```

Como se puede observar se tomará un umbral de +/- 0.4 como margen a partir del el cual el robot debe empezar a moverse. Dicho umbral evita por ejemplo que se produzcan movimientos indeseados al inclinar levemente el Smartphone.

En el algoritmo, se comparan los valores absolutos de la aceleración lineal en cada eje. En función del valor obtenido así como el signo del valor medido se establece como comando el giro a la derecha, a la izquierda avance o retroceso.

En caso de que la aceleración lineal en el eje z sea mayor de 2 m/s y superior a la medida en valor absoluto para ejes *x* e *y* el robot se mantendrá en reposo.

Tanto el publicador *opt* como *veloci* responden a la estructura de datos definida para el posterior modelo de Robot evitador de obstáculos.

De igual modo, definimos también dentro de la clase como instancia privada tanto publicadores como node Handler.

```
private:
  ros::NodeHandle n;

  ros::Subscriber di;
  ros::Publisher veloci;
  ros::Publisher opt;
```

Más tarde y dentro de main() se inicializa el nodo con el nombre “talker”, y se define un rate loop de 100 milisegundos. Es decir, la rutina de control se ejecutará cada 100 microsegundos. Este tiempo de refresco es suficiente para que la experiencia de control de usuario sea suficientemente agradable y rápida de respuesta. De igual modo se define un objeto de clase “miclase”. Sin esta definición del objeto la clase queda únicamente como una abstracción lógica.

5.5.1.1. Control de motores mediante PWM

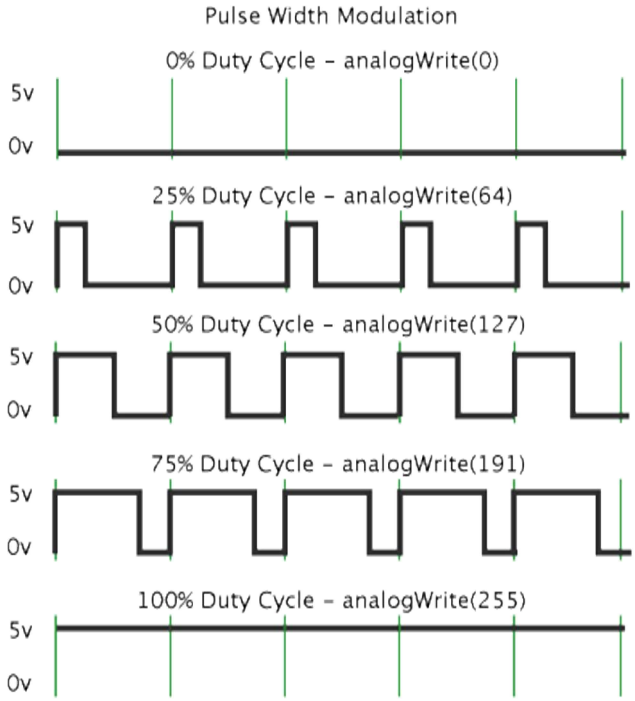
En este modelo se ha tenido que analizar el funcionamiento de los puertos PWM de Arduino de manera que queden adaptados los datos publicados al suscriptor ejecutado en el nodo remoto.

La Modulación por Ancho de Pulso (PWM = Pulse Width Modulation) es una técnica para simular una salida analógica con una salida digital. El control digital se usa para crear una onda cuadrada, una señal que conmuta constantemente entre encendido y apagado. Este patrón de encendido-apagado puede simular voltajes entre 0 (siempre apagado) y 5 voltios (siempre encendido) simplemente variando la proporción de tiempo entre encendido y apagado. A la duración del tiempo de encendido (ON) se le llama Ancho de Pulso (pulse width). Para variar el valor analógico cambiamos, o modulamos, ese ancho de pulso.

En otras palabras, con Arduino la frecuencia PWM es bastante próxima a 500Hz lo que equivale a periodos de 2 milisegundos cada uno. La llamada a la

función analogWrite() debe ser en la escala desde 0 a 255 como se indica en la Figura 19, siendo 255 el 100% de ciclo (siempre encendido), el valor 127 será el 50% del ciclo (la mitad del tiempo encendido), etc.

Figura 19 Evolución de Duty Cycle



En este caso para escalar la señal entrante entre 0 y +/-9.8 basta con multiplicar la variable por 255 diviendo antes por 9.8

En este caso : `velocidad=xa*26.02`

En la placa Arduino el sketch a programar es muy parecido al utilizado en el modelo de Robot evitador de obstáculos (ver apartado 6.2.4). Se trata de establecer un control PWM sobre los puertos conectados a ambos motores que permitan, en función del valor de la variable opción y velocidad publicada por el nodo master (Raspberry pi) controlar los motores en consonancia con estas. En este caso la rutina se establece en el archivo Android .ino cargado y ejecutándose sobre el Arduino.

6. Modelo 2: Robot evitador de obstáculos

6.1. Descripción

El segundo modelo de programación desarrollado consiste en llevar a cabo la funcionalidad de evitación de obstáculo para el robot móvil previamente analizado (el C-8090).

En líneas generales el funcionamiento general del sistema consistirá en dotar al robot de movimiento de manera que éste se vea variado en su dirección una vez detectado un obstáculo. El robot de esta manera se verá activado en función de la luz existente en el exterior. Es decir es posible su configuración tanto en entornos oscuros únicamente como todo lo contrario.

Las capacidades de este modelo hacen al robot al funcionar de manera autónoma útil para aplicaciones de recorrido de entornos cerrados, reconocimiento de espacios etc.

6.2. Solución

La para la implementación de estas capacidades se hace uso de un conjunto de 7 sensores.

- 5 sensores HC-SR04
- 2 sensores LDR.

6.2.1. Detección de obstáculos

Los sensores HC-SR04 son sensores ultrásonicos integrados cuyo principio básico de funcionamiento es similar al de sónar. Este dispositivo evalúa la distancia a un objetivo interpretando los ecos emitidos ondas de ultrasonido.

Este módulo es capaz de medir de manera exacta distancias a objetivos situados entre 0cm y 400 cm del mismo con un error máximo de 3cm. Es de tamaño compacto, alto rango de funcionamiento y muy fácil de integrar lo cual lo convierte en un sensor muy manejable para el mapeo de objetos así como simplemente medida de distancias.

Además, se trata de un dispositivo que puede ser muy fácilmente integrable con placas microcontroladoras. En el presente proyecto interesa sobre todas las cosas su total compatibilidad con Arduino.

El sensor dispone de 4 pines.

- Vcc (5V): Conectable a la interfaz de 5voltios de Arduino. Este pin proporciona la tensión necesaria para la alimentación del circuito receptor y adaptador de la señal de eco.
- Pin GND: Conectable al pin de tierra de Arduino.
- Pin trigger: Conectable a cualquier interfaz digital de Arduino. Este pin será el que marque el pulso de trigger a emitir mediante ultrasonido por el sensor cuya señal será reflejada por los obstáculos que este se encuentre en su camino.
- Pin Echo: En este pin digital Arduino medirá la duración del pulso recibido y su retardo respecto al emitido.

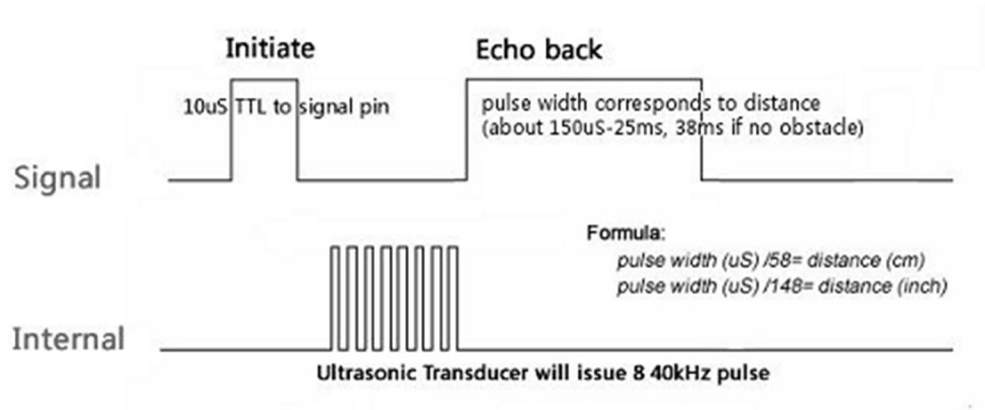
Según este conexionado, al proporcionar un pulso de 10 microsengundos el sensor transmite un pulso de ultrasonido a 40Khz de 8 ciclos y produce un pulso de salida con una anchura respecto al anterior que corresponde con el tiempo requerido por el pulso de trigger en retornar al sensor. Midiendo la anchura y el retardo de este y teniendo en cuenta anchura del pulso anterior así como la velocidad del sonido en el aire, es fácilmente calculable la distancia a un objetivo.

El pulso de echo corresponde a la distancia (sobre 150us-25ms, 38 ms si no hay obstáculo). Aplicando la fórmula:

Anchopulso(uS)/58= distancia (cm)

se obtiene la distancia medida en Arduino mediante la medida de la longitud en uS del pin digital asignado tal como se establece en la Figura 20.

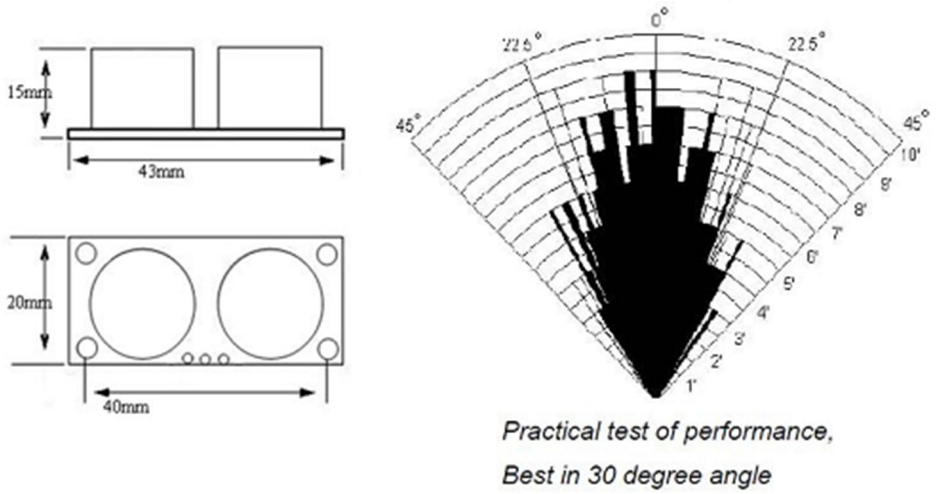
Figura 20 Funcionamiento HC-SR04



Las características del módulo son las siguientes ver Figura 21.

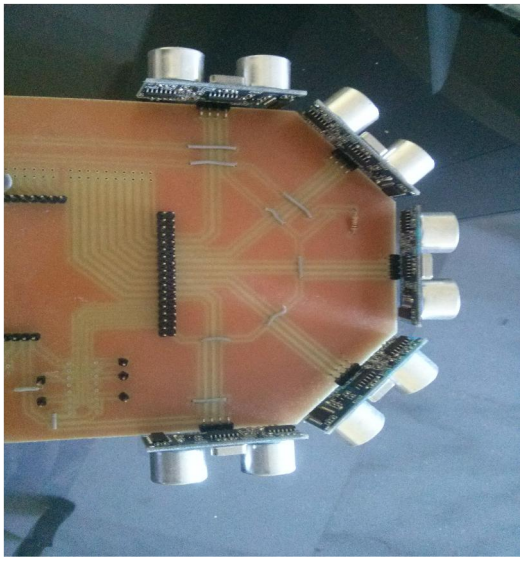
- Voltaje de funcionamiento: 5V (DC)
- Corriente estática: <2 mA
- Trabajo actual: 15mA
- Frecuencia de trabajo: 40KHz
- Señal de salida: frecuencia de la señal eléctrica, 5V alto nivel, 0V bajo nivel
- Ángulo Eficaz: <15 º
- Distancia de detección: 2 cm - 450 cm
- Resolución: 0,3 cm
- Medición de ángulo: 30 º
- Disparo de la señal de entrada: TTL pulso 10µs
- Echo señal de salida: señal PWL de TTL

Figura 21 Medidas y rango de alcance



El ángulo de alcance de cada sensor es de 30º, por lo tanto, con 5 sensores sería suficiente para cubrir hasta 150 º alrededor del dispositivo. Los sensores se dispondrán sobre la placa PCB diseñada como shield del Arduino y rutado de señales hacia sensores y motor de la manera establecida en la Figura 22.

Figura 22 Disposición final HC-SR04



Los sensores se disponen en una semicircunferencia cada 45º, por tanto en 0º, 45º, 90º 135º y 180º. Los ángulos ciegos posibles incluidos entre los mismos se consideran como un componente de error asumible dada la naturaleza de la aplicación a desarrollar.

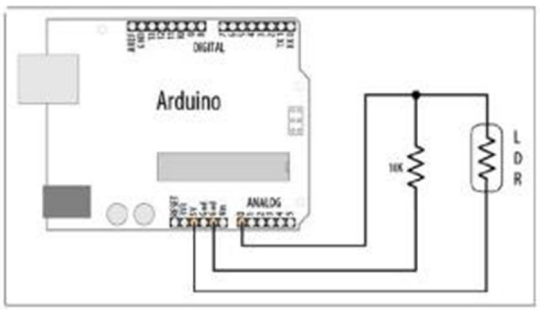
6.2.2. Sensores LDR

El siguiente problema a resolver es la integración de los dos sensores LDR situados en los dos extremos de la placa. Estos sensores son utilizados para la medición de la intensidad lumínica exterior y proporciona una variable a utilizar por el nodo ROS de control ejecutado en Raspberry para determinar el modo de funcionamiento.

La manera más fácil para detectar los niveles de luz es usando una fotorresistencia (LDR). Esta resistencia cambia con los cambios en los niveles de luz. Integrando este sensor dentro de un circuito divisor de tensión, la existencia de luz provoca un cambio de tensión en este circuito el cual puede ser medido por cualquier puerto analógico configurado como entrada en Arduino

Para el montaje, se ha rutado en la PCB diseñada el siguiente circuito diseñado para ofrecer tensión en caso de luz incidente sobre el sensor tal como se establece en la Figura 23.

Figura 23 Conexionado LDR



Este circuito es la forma estándar de usar un sensor que cambia su resistencia a partir de algún fenómeno físico. El circuito de la figura 1 cambiará el voltaje en el pin análogo 0 cuando la resistencia del LDR cambia con la variación de los niveles de luz.

6.2.2.1. Implementación

Un circuito de este tipo no nos dará la gama completa de valores posibles de la entra análoga (0 a 1023) ya que el voltaje no oscila entre 0 y 5 volts. Esto se debe a que siempre habrá una caída de voltaje a través de cada resistencia, por lo que el voltaje donde se juntan nunca llegará a los límites de la fuente de alimentación. Cuando se utilizan sensores de este tipo, es importante comprobar los valores actuales que el dispositivo regresa en la situación en que se estén utilizando. Por lo que, debemos determinar cómo convertir los valores que necesitamos para controlar lo que se desea controlar.

Sin embargo como solo nos interesa determinar la existencia de luz brillante para así ordenar la puesta en marcha o la parada del funcionamiento de nuestro robot, se ha optado por establecer como condición la medición a través del puerto analógico de Arduino de un voltaje superior a 1,5V.

La colocación de dos sensores en lugar de solo 1 intenta evitar que zonas de sombra provocadas por las placas a bordo o por diferentes orientaciones de la luz exterior provoquen un funcionamiento erróneo.

6.2.3. Programa Arduino

En primer lugar se describirá el sketch de Arduino necesario para recabar información de los sensores en cuestión así como su tratamiento para la publicación de datos sobre los mismos referentes a la distancia en cm que separa al robot en cualquiera de sus ángulos a un objeto.

El primer paso en la programación del sketch llamado obstaculos.ino fue la inclusión de las librerías ROS correspondientes.

```
#include <ArduinoHardware.h>
#include <ros.h>
#include <std_msgs/Int8.h>
#include <std_msgs/Float32.h>
```

Por otro lado, para evitar la concurrencia de publicadores vamos a incluir los datos de los 7 sensores disponibles en un mismo mensaje a publicar por un mismo publicador. En cada bucle de ejecución de Arduino (función Loop) se publicarán una vez conformado el tipo de dato dist_msg que integra la distancia en cm hasta cualquier objeto medida por todos los sensores, así como el voltaje medido a través de los puertos analógicos conectados al circuito divisor de tensión con LDR.

6.2.4. Generación de mensaje custom (Distance)

Enviar 7 variables en un sólo mensaje requiere de la configuración de un tipo de mensaje customizado. No existe ninguna variable en la librería std_msg que nos permita tal cometido. Por lo tanto, para ello se han seguido los siguientes pasos:

En primer lugar es necesario elaborar un fichero .msg en el directorio /msg de nuestro paquete /sandbox/proyecto donde definiremos el mensaje que se intercambiará en los dos nodos. En este caso distanci.msg

Este mensaje contiene las variables a incluir en el mensaje, en este caso todas de tipo Float32, el cual es un tipo compatible con el formato de datos a intercambiar.

```
Float32 upfront;
Float32 upleft;
Float32 upright;
Float32 leftedge;
Float32 rightedge;
Float32 frontlight;
Float32 backlight;
```

Una vez definido el fichero se requiere para su uso en Arduino de este tipo de datos la definición en un fichero de cabeceras .h a incluir en nuestro sketch. Para ello ejecutamos la herramienta ros_client make_library.py

```
rosrun rosserial_client make_library.py
/opt/ros/fuerte/fuerte_workspace/sandbox/proyecto proyecto
```

Este comando genera un archivo distanci.h a partir de las definiciones de mensajes que se encuentren dentro del paquete proyecto que se encontrará alojado en la carpeta de nuestro paquete proyecto.

Una vez realizado este paso es posible añadir dicha librería en nuestro sketch mediante la sentencia:

```
#include "proyecto/distanci.h"
```

6.2.5. Inicialización

En primer lugar se inicializan los puertos digitales y analógicos pertinentes. Esto es 2 puertos digitales por cada sensor HC-SR04 para los pines de Trigger y Echo y dos pines analógicos de entrada para medir el voltaje ofrecido por el circuito LDR.

```
#define trigPin 12
#define echoPin 13
#define pinalaog1 2
#define pinanalog2 3

void setup()
{
pinecho[0]=7;
pinecho[1]=8;
pinecho[2]=9;
pinecho[3]=10;
pinecho[4]=11;
pintrigger[0]=12;
pintrigger[1]=13;
pintrigger[2]=14;
pintrigger[3]=15;
pintrigger[4]=16;
pinMode(pinanalog,INPUT);

int i=0;
for( i=0;i==4;i++)
{
  pinMode(pintrigger[i], OUTPUT);
  pinMode(pinecho[i], INPUT);
}
```

6.2.6. Medidas de distancias y publicación

Tras la configuración de los pines necesarios llevaremos a cabo la medida de distancias en los pines correspondientes. Para ello tal como se ha indicado anteriormente, se emitirá un pulso de duración 10us y posteriormente se medirá la duración del pulso de echo.

Para ello se ha generado la función getdistance(float *, float *, float *) la cual recibirá para mayor comodidad punteros a tablas donde almacenamos los valores de distancia y los números de pines echo y trigger. Los valores de

distancia medidos se introducirán en el array distancias gracias al paso de argumento por referencia mediante el puntero.

En loop():

```
float distancias[5];
  float *distance=distancias;
  int *ppintrigger=pintrigger;
  int *ppecho=pinecho;
  float luzdelante=0.2;
  float luzdetras=0.3;
luzdetras=analogread(pinanalog1);
luzdelante=analogread(pinanalog2);
  getdistance(distancias,ppintrigger,ppecho);

void getdistance(float *dist, int *echo, int *trigger){

  float duration, distance;

  int i;
  for (i=0; i==4;i++){
    duration =0;
    distance=0;
    digitalWrite(trigger[i],LOW);
    delayMicroseconds(5);
  digitalWrite(trigger[i], HIGH);
  delayMicroseconds(10);
  digitalWrite(trigger[i], LOW);
  duration = pulseIn(echo[i], HIGH);
  distance= duration*0.01724138;

  if (distance >= 200 || distance <= 0){

    distance=1000;
 dist[i]=distance;

  }
  else {
   dist[i]=distance;}
  }
```

Cabe destacar que en lugar de dividir por 58, y por las capacidades de operación en coma flotante ofrecidas por el procesador AtMEGA de Arduino se lleva a cabo una multiplicación por 1/58 = 0,01724138.

En caso que la distancia sea mayor de 200 cm o que por error se mida una distancia negativa se tomará la distancia objetivo como 1000cm la cual simbolizará el infinito.

Como se puede apreciar en el código, el proceso de medida de distancias es secuencial por lo que existe un cierto riesgo de que al seguir moviéndose el robot haya un desfase entre la medida de los sensores que provoque una

detección de objetos deficiente. Sin embargo, los 50 microsegundos utilizados para la medición de todos los sensores en completo suponen un ciclo de funcionamiento lo suficientemente rápido para reducir muchísimo este riesgo.

A continuación es necesario definir tanto los publicadores como los suscriptores necesarios para el control del motor. En el caso que nos ocupa, se debe definir un publicador que envíe un dato de tipo distanci el cual contendrá las variables medidas por los sensores:

```
distanci.upfront;
distanci.upleft;
distanci.upright;
distanci.leftedge;
distanci.rightedge;
distanci.frontlight;
distanci.backlight;
```

Para ello se añaden al sketch las siguientes sentencias:

Definición de variable tipo distanci llamada dist_msg:

```
proyecto::distanci dist_msg;
```

Definición de manejador del nodo: ejecuta el proceso necesario para el establecimiento del nodo remoto, además provee de una capa de resolución de dominios que permite escribir subcomponentes más fácilmente:

```
ros::NodeHandle nh;
```

Si la instancia NodeHandle es eliminada el nodo es apagado automáticamente dejando tanto de publicar como de escuchar los topics publicados actualmente.

A continuación se define un publicador de un topic llamado “distance” que será único en nuestro en nuestro sistema, de nombre dist y que publicará en el topic determinado cada vez que sea invocado una la variable dist_msg.

```
ros::Publisher dist("distance", &dist_msg);
```

A continuación y una vez definidas todas las entidades se requiere la inicialización de las mismas mediante las sentencias:

```
nh.initNode();
nh.advertise(dist);
```

Por último, asignamos los valores medidos a la variable distancia definida (dist_msg) y llamamos al publicador dist :

```
dist_msg.upfront=distancias[0];
dist_msg.upleft=distancias[1];
dist_msg.upright=distancias[2];
dist_msg.leftedge=distancias[3];
dist_msg.rightedge=distancias[4];
dist_msg.frontlight=luzdelante;
dist_msg.backlight=luzdetras;
dist.publish(&dist_msg);
```

De esta manera, el publicador de nombre “dist”, publicará mensajes de tipo “dist_msg” en el topic “distances”.

Las distancias a un objeto medidas por cada uno de los sensores instalados en el robot permitirán articular un algoritmo que establecerá el modo de movimiento de éste. Al tratarse de un robot móvil con dos grados de libertad, estos movimientos serán el longitudinal y el giro sobre si mismo.

6.3.**Suscriptor y algoritmo de movimiento**

El modo de movimiento ejecutado por el robot en todo momento es establecido según los datos recibidos de un topic publicado por el nodo Master ejecutándose en la Raspberry Pi según los comandos de control requeridos. Además será en este nodo donde se establezca el comando de velocidad de referencia en cada uno de los movimientos.

Cada ciclo de control establecido en 10 ms se recibirá en Arduino una señal de control. Para ello se se configuran una pareja de pines digites por cada motor. Estos pines deberán estar conectados al integrado L293D mediante la PCB diseña en el presente proyecto a los pines IN1 IN2 IN3 y IN4 de éste. Dichos pines controlarán el sentido de giro de cada motor de la siguiente manera:

- Input 1 “HIGH” , Input2 “LOW” , Input 3 “HIGH”, Input 4 “LOW”: ambos motores se moverán en sentido opuesto a las agujas del reloj por lo que el robot avanzará hacia delante.
- Input 1 “HIGH” , Input2 “LOW”, Input 3 “LOW”, Input 4 “HIGH”: el motor derecho girará en sentido de las agujas del reloj mientras el que el izquierdo girará al contrario por lo que el conjunto del robot girará sobre si mismo hacia la derecha
- Input 1 “LOW” , Input2 “HIGH”, Input 3 “HIGH”, Input 4 “LOW”: el motor izquierdo girará en sentido de las agujas del reloj mientras el que el derecho girará al contrario por lo que el conjunto del robot girará sobre si mismo hacia la izquierda.
- Input 1 “LOW” , Input2 “HIGH”, Input 3 “LOW”, Input 4 “HIGH”: ambos motores girarán en sentido de las agujas del reloj por lo que el conjunto del robot avanzará hacia atrás.

Estas salidas vendrán controladas por una variable int recibida a través del suscriptor opt el cual escuchará al topic "opción".

Una vez recibido datos a través de este suscriptor se ejecuta la función denominada Callback. El evento de recepción de un dato a través de un suscriptor provoca una interrupción que llama a esta función y que según el valor de la variable "opción" recibida se genera una salida en los pines digitales asignados a las entradas del L293D anteriormente descritas de la siguiente manera:

- Opción=1. Movimiento del robot hacia delante
- Opción=4. Giro a la izquierda
- Opción=3. Giro a la derecha
- Opción=2. Movimiento del robot hacia detrás.
- Opción=9. Robot parado

La función de callback antes descrita se denomina movimientoCb y se define de la siguiente manera:

```
void movimientoCb(const std_msgs::Int8& option)
{int op;
int a=0;
op=option.data;
if(a==0)
{ switch(op)
{
case 0:
digitalWrite(motizqL,LOW);
digitalWrite(motizqH,LOW);
digitalWrite(motderH,LOW);
digitalWrite(motderL,LOW);
digitalWrite(10,HIGH);
break;

case 1:
digitalWrite(motizqL,HIGH);
digitalWrite(motizqH,LOW);
digitalWrite(motderH,HIGH);
digitalWrite(motderL,LOW);
break;

case 2:
digitalWrite(motizqL,LOW);
digitalWrite(motizqH,HIGH);
digitalWrite(motderH,LOW);
digitalWrite(motderL,HIGH);
break;

case 3:
digitalWrite(motizqL,HIGH);
digitalWrite(motizqH,LOW);
digitalWrite(motderH,LOW);
digitalWrite(motderL,HIGH);
```

```
break;

case 4:
digitalWrite(motizqL,LOW);
digitalWrite(motizqH,HIGH);
digitalWrite(motderH,HIGH);
digitalWrite(motderL,LOW);
break;

default:
digitalWrite(motizqL,LOW);
digitalWrite(motizqH,LOW);
digitalWrite(motderH,LOW);
digitalWrite(motderL,LOW);
}
}
else
{
digitalWrite(motizqL,LOW);
digitalWrite(motizqH,LOW);
digitalWrite(motderH,LOW);
digitalWrite(motderL,LOW);
}
}
```

El suscriptor se define e inicializa de la siguiente manera:

```
ros::Subscriber<std_msgs::Int8> mov("opcion",&movimientoCb);
nh.initNode();
nh.subscribe(mov);
```

Por otro lado, la velocidad de referencia a introducir a los motores se modela como una señal PWM directamente conectada al pin ENABLE del integrado L293D. Esta señal regulará introduciendo un duty cycle mayor o menor en el integrado el puente H de manera que la tensión de entrada aplicada al motor variara según estime conveniente el sistema controlador.

Se define un suscriptor por tanto para recibir el comando de velocidad. Este comando ya ha sido escalado entre 0 y 256 en el nodo Master (Raspberry Pi) por lo que la única tarea pendiente es escribir el valor recibido en el puerto analógico correspondiente. Esta tarea será realizada mediante la función veloCB ejecutada cada vez que se recibe un nuevo dato en el suscriptor del topic "velocidad".

6.4.Programación de Nodo controlador (Raspberry Pi)

6.4.1. Descripción

A partir de ahora, una vez definido el programa que recibirá la información relativa a los sensores, la adaptará y publicará queda pendiente la definición del nodo Master controlador de la solución. Este controlador se ejecutará sobre la placa Raspberry Pi. Se llevarán a cabo las siguientes tareas:

- Aplicación de algoritmo de control del controlador
- Definición de suscriptor del topic distance.
- Adaptación de velocidad objetivo a duty cycle de pulsos PWM
- Definición de ciclo de control (tiempo entre escuchas a topics y ejecución de algoritmo).
- Grabación de resultados una vez detectado un obstáculo (posición)

6.4.2. Algoritmo

El algoritmo de control basado en 5 sensores HC-SR04 tiene en cuenta simultáneamente los datos recibidos a través del topic "distance". Recordamos que los datos publicados en este topic por el nodo remoto Arduino son los siguientes:

```
Float32 upfront;
Float32 upleft;
Float32 upright;
Float32 leftedge;
Float32 rightedge;
Float32 frontlight;
Float32 backlight;
```

En el nodo Master se interpretan estos datos y en función de la distancia hacia el objeto más próximo se tomará una decisión en cuanto al modo de movimiento del robot (topic "opción") así como la velocidad de referencia a asignar.

Para ello se definen hasta 3 intervalos de distancia los cuales se definen a 20, 30 y 40 cm del robot.

```
#define interval1 30
#define interval2 20
#define interval0 40
```

Un objeto en el primer intervalo (0) únicamente activará la detección. En el segundo intervalo se llevará a cabo un cambio del modo de movimiento en función de cual o cuales sean los sensores que han detectado el objeto.

Como es obvio en el algoritmo, todo objeto detectado por 3 sensores delanteros captará más atención que el resto por lo que la medida adoptada en cuanto a cambio del modo de funcionamiento tendrá prioridad sobre las demás. El hecho de que esto sea así es simplemente porque según el algoritmo el robot por defecto y mientras no detecte ningún obstáculo se moverá siempre hacia delante.

Se establece un algoritmo teniendo en cuenta en función de los datos detectados por cada sensor una acción definida.

En primer lugar se analiza si algún objeto ha sido detectado por alguno de los sensores. Para ello se leen los datos recibidos y se configura una variable llamada "detected" en caso de que la lectura de algún sensor ofrezca un objeto situado a menos de 40 cm del robot (interval0).

En caso de que ningún objeto sea detectado se establecerá opción=1;

Se establecen prioridades a la hora de tratar los datos por parte del control. Si alguna de las distancias medidas supone la cercanía a menor al primer umbral definido (20cm) la opción de movimiento determinada para estos casos tendrá más prioridad en su aplicación que el resto. Dentro de este caso tendrán más prioridad las acciones asociadas a la medida referida a los sensores frontales frente a los dos laterales.

6.4.3. Algoritmo de prioridad 1

En caso de que algún objeto sea detectado dentro del intervalo 2:

- En caso de que un objeto sea detectado a una distancia menor de 20 cm por alguno de los sensores frontales (upfront, frontleft o frontright) el robot retrocederá hacia atrás (opción=4) a velocidad media (Velocidad=180). Este caso cumple condiciones de establecimiento repentino de obstáculo pues en caso de que este fuese fijo, las medidas de evitación del mismo se hubiesen tomado mediante las acciones de prioridad 2.

  Si no se ha cumplido esta condición si quien ha detectado el obstáculo han sido los sensores laterales (rightedge o lefteedge) el robot seguirá moviéndose girará a la izquierda (opción=2) o a la derecha (opción=3) pero a una velocidad menor a la máxima (velocidad=155).

6.4.4. Algoritmo de prioridad 2

- En caso de que no se hayan producido las condiciones de prioridad 1 y que un objeto sea detectado entre el intervalo 1 y 2 (entre 30 y 40 cm) por parte del sensor frontal y alguno de los 2 laterales el robot girará a velocidad moderada hacia la izquierda o derecha. Si se ha detectado el

objeto en este intervalo por el sensores upright girará a la izquierda (opción=2), si ha sido por el upleft girará a la derecha (opción 3). La velocidad se establecerá con un dutycycle de 155.

El hecho de establecer esta velocidad responde a la posibilidad de que se deje de recibir el objeto como un obstáculo en cuyo caso el robot puede seguir su camino. Si la velocidad de giro es superior, es posible que la desviación respecto a su camino resultante sea mayor al llevar a cabo un movimiento más brusco no habiendo ningún obstáculo en su camino.

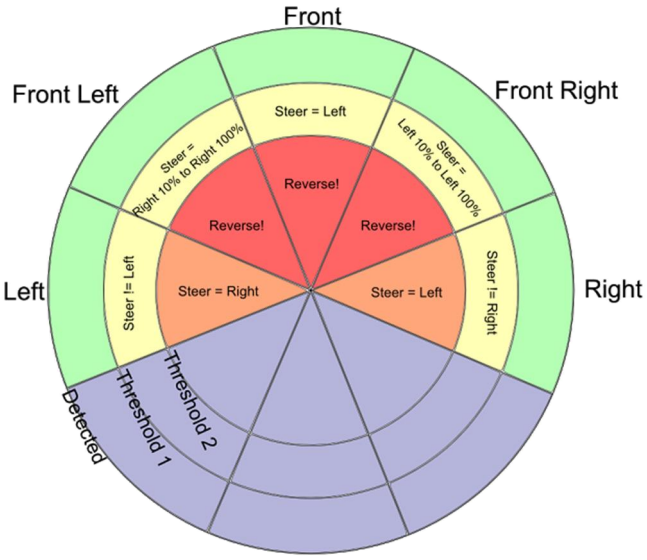
- El último caso posible es que ningún sensor reciba un obstáculo dentro de ningún intervalo. En este caso el movimiento indicado es el de avanzar hacia delante a máxima velocidad (opción=1 y velocidad=200).

El código asociado al algoritmo e implementado es el siguiente incluyendo la definición y recepción de datos referentes a las variables:

```
distan[0]=dist_msg->upfront;
distan[1]=dist_msg->upleft;
distan[2]=dist_msg->upright;
distan[3]=dist_msg->leftedge;
distan[4]=dist_msg->rightedge;
int detected=0;

if(distan[0]<interval0 || distan[1]<interval0 ||
distan[2]<interval0 || distan[3]<interval0 ||
distan[4]<interval0 )  {
    detected=1;
  }

  if(detected==1){
    if(distan[0]<interval2 || distan[1]<interval2 ||
distan[2]< interval2){
      opcion=4; velocidad=180;
    }else if (distan[4]< interval2){
      opcion=2; velocidad=155;}
    else if( distan[3]<interval2){
      opcion=3; velocidad=155;} else if (( interval1>distan[0]
&& distan[0]>interval2) || (interval1>distan[2] &&
distan[2]>interval2)){
      opcion=2; velocidad=155;}else if (interval1>distan[1]
&& distan[1]>interval2){
      opcion=3;velocidad=155;
    }}else{
    opcion=1; velocidad =200;
  }
```

El esquema resumen del algoritmo implementado es el establecido en la Figura 24.

Figura 24 Esquema Algoritmo utilizado



Para establecer un valor a publicar para la velocidad de referencia se toma como hemos comentado en el apartado referente al modelo 1 un valor incluido entre 0 y 256. Este es el nivel de escalado de la señal PWM.

En el modelo de funcionamiento anterior el comando de velocidad dependía de la aceleración lineal medida por los sensores del Smartphone. Se debía escalar el valor medido de está entre 0 y 9.8 en valor absoluto en el intervalo 0 a 256.

En este caso el control de la velocidad es programable por el usuario por lo que se tomarán 3 valores predefinidos a utilizar en diferentes ocasiones.

- Velocidad estándar: duty-cycle =256. Movimiento hacia adelante
- Velocidad media: duty-cycle=180. Giro a la izquierda y derecha en situaciones de prioridad 1 así como avance hacia atrás.
- Velocidad baja: duty-cycle=155. Giro a la izquierda y derecha en situaciones de prioridad 2.

6.5.**Implementación sobre plataforma real**

Una vez escrito todo el código fuente del sistema referente al nodo máster (Raspberry Pi) para su implementación se requiere:

- Inclusión de la tarea de compilación en las listas de Make
- Inclusión de la tarea de generación de las cabeceras de mensajes customizados

Para estas tareas debemos incluir en el archivo CMakeLists.txt de nuestro proyecto las tareas referentes a la compilación de nuestro código fuente. En este caso nuestro archivo automatic.cpp.

Para ello añadimos la siguiente línea en nuestro archivo:

```
# rosbuild_add_executable(proyecto src/automatic.cpp)
```

Al ejecutar el make se generará un ejecutable denominado proyecto dentro de nuestro paquete de trabajo llamado de igual forma. El archivo creado es proyecto.o.

Para la generación de los archivos de cabecera para nuestro mensaje customizado el cual llamábamos distance.msg:

```
# rosbuild_genmsg()
```

Al ejecutar la orden make en el directorio de raíz de nuestro proyecto se ejecutaran ambas órdenes las cuales compilarán nuestro proyecto así como generará las dependencias necearias para su ejecución. Asimismo establecerá los ficheros de cabecera de mensaje necesarios para poder ser utilizados en el intercambio de topics como tipo de dato válido en el sistema.

Para arrancar el robot en su funcionamiento deberemos ahora ejecutar todos los servicios necesarios:

- Arduino Mega programado y conectado a Raspberry a través del puerto USB
- Ejecución de roscore.
- Ejecución del servicio serial_node.py del paquete Rosserial el cual redirigirá todos los tópics recibidos por el puerto seleccionado a los nodos pertinentes.
- Ejecución del programa controlador del nodo máster (Raspberry Pi).

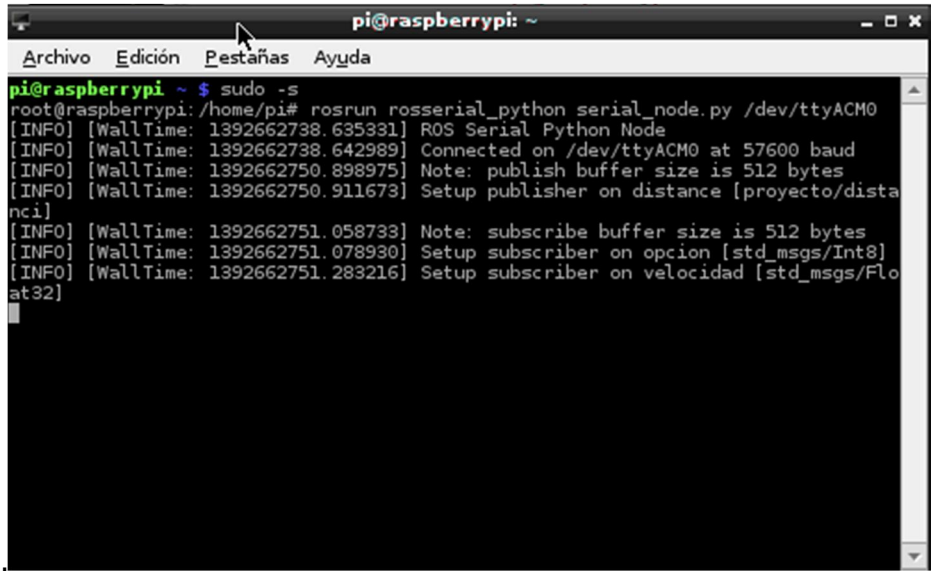
6.6. **Pruebas**

En primer lugar iniciamos roscore. Posteriormente iniciamos el proceso serial_node.py. Una vez inicializado este proceso, los publicadores y suscriptores establecidos en este puerto serie se virtualizarán en la placa nativa como si se estuviesen ejecutando en la misma máquina. Ejecutamos:

```
#rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Lo cual establece el nodo RosSerial tal como se establece en la Figura 25.

Figura 25 Establecimiento de conexión serie Arduino-Raspberry

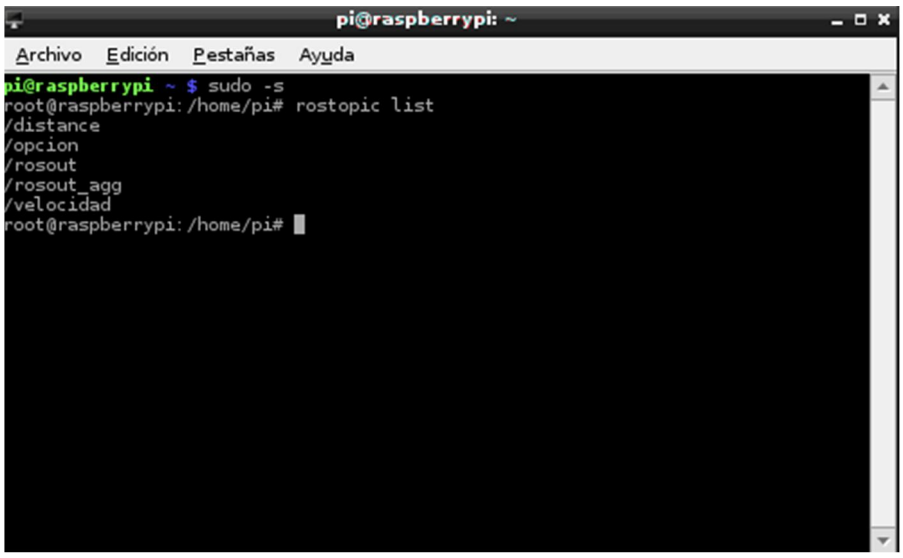


Tras la ejecución de proyecto.o (el nodo controlador)

```
#rosrun proyecto proyecto
```

se comprueba la correcta publicación de topics tal como se establece en la Figura 26.

Figura 26 Topics publicados



Y tras esto, haciendo uso de la herramienta rxgraph (rqtgraph) se puede visualizar en un esquema el correcto establecimiento de nodos, de topics así como cuales de esto son publicados por cada nodo o a cuales están suscritos cada uno. Rxgraph es una herramienta de gran valor para labores de debugging la cual no solo ofrece un esquema muy visual sobre la configuración en ejecución en

cada momento sino que también ofrece información haciendo click sobre cada nodo acerca de los topics, mensajes recibidos y tipos de datos incluidos en cada uno tal como se observa en la Figura 27,29 30.

Figura 27 Esquema Rxgraph

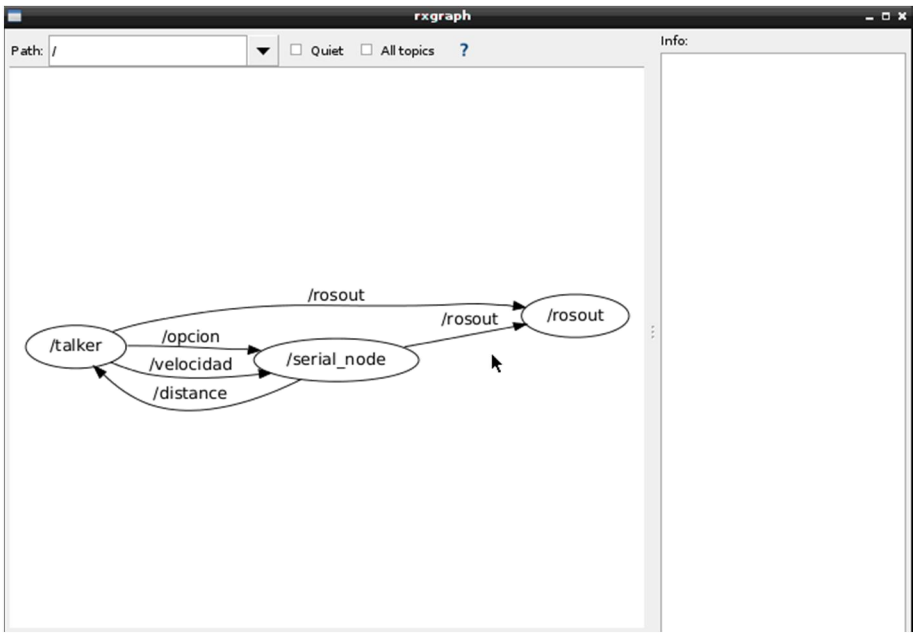


Figura 28 Información del nodo Master (Rasp Pi)

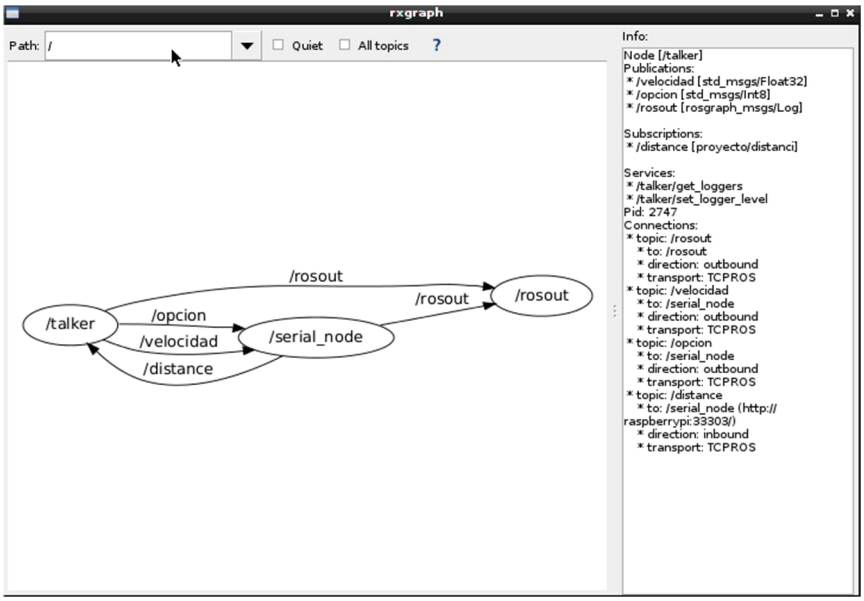
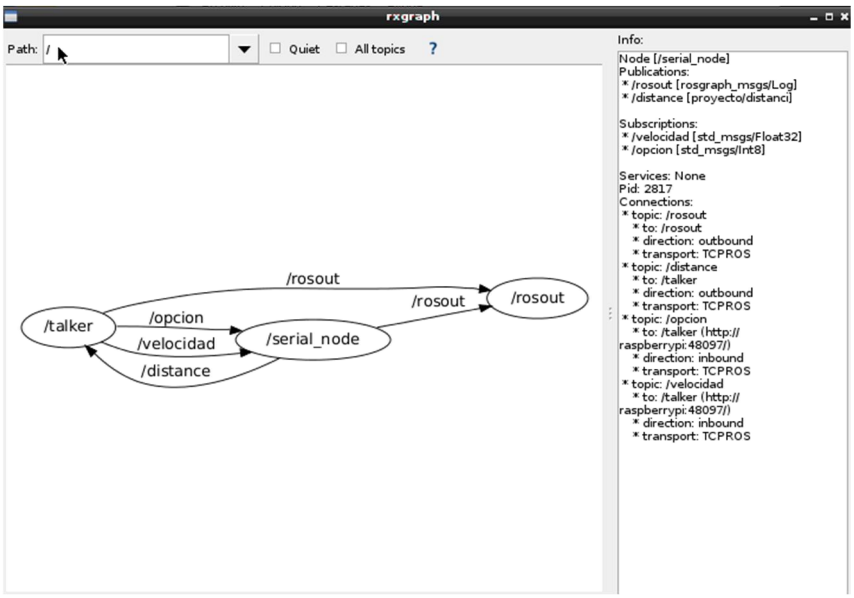


Figura 29 Información de nodo remoto (Arduino)



En las anteriores figuras se observa como la herramienta rxgraph ofrece una visión esquemática sobre como se estructura el sistema robot asociado (en este caso la de un robot evitador de obstáculos.
Se establece para cada nodo:

- Topic publicados.
- Conexiones estqblcidas
- Puertos de comunicación entre nodos
- Dirección de comunicación para cada topic (publicado o suscrito)
- Trasporte (protocolo de comunicación)

Como se observa, Arduino consta como un nodo serial. Este es un nodo virtualizado por rosserial, el cual simplemente establece los publicadores y suscriptores programados en Arduino dándoles entidad de nodo remoto en el sistema ROS.

7. Diseño de escudo para Arduino

Con el objetivo de conseguir una integración de la circuitería necesaria para el cableado del robot se ha realizado un diseño de placa PCB. El objetivo es realizar un circuito con pines a medida para poder "pinchar" la placa Arduino sobre éste. Lo que comúnmente se denomina "shield". El diseño de esta PCB responde a los siguientes objetivos.

- Fiabilidad en las conexiones
- Sistema compacto
- Escalabilidad del diseño: se han rutado una serie de salidas digitales y analógicas no utilizadas para su uso en futuras aplicaciones.
- Búsqueda de bajas perdidas en circuitos y conexiones
- Integración de sensores y componentes.

7.1. Diseño de PCB

En esta placa se incluirá el rutado de las pistas necesarias para integrar conectores de manera que Arduino Mega quede totalmente acoplado con sensores y salidas analógicas así como con el circuito que actúa como driver de los motores (L293D) de manera compacta.

Para ello se ha diseñado en Altium una placa PCB en la que en primer lugar se ha buscado una geometría que permita a colocación de sensores HC-SR04 y LDR dispuestos tal como se ha contemplado anteriormente.

Se han colocado pines en los lugares adecuados para poder colocar el Arduino sobre esta y se han rutado las pistas de cobre convenientemente para llevar cada salida digital y analógica PWM a su pin correspondiente.

7.1.1. Diseño en altium

El diseño en Altium ha constado de dos fases:

- Diseño del esquemático del circuito diseñado. Teniendo en cuenta los pines de salida de Arduino así como los de entrada y salida del circuito LDR y del puente H L293 así como los pines de entrada y salida, alimentación y tierra de los sensores HC-SR04.

- Diseño del layout de la placa

Para el diseño de la placa PCB se ha hecho uso del software Altium. Se trata de un software EDA para el diseño de placas de circutos impresos, FPGAs y diseños de software embebido. En el caso de diseño de una PCB con un propósito concreto Altium ofrece las siguientes herramientas:

- Librerías de componentes ya establecidos
- Edición de esquemático del circuito a diseñar a nivel de emplazamiento, conectividad y definición de reglas de diseño.
- Integración con componentes de distribuidor y acceso a datasheets de los mismos.
- Simulación de circuito mediante SPICE.
- Integración pre-layout.
- Exportación de Netlist del circuito.
- Soporte para elementos embebidos
- Mejoras en el rutado de circuitos par diferencial
- Optimización en el rutado del circuito y colocación de componentes.
- Definición de huella de componentes preestablecidos.

Para el diseño de la PCB se deben de tener en cuenta los siguientes componentes:

- Esquemático: copia del circuito de la hoja de datos del componentes esto es para poder exportar el circuito al PCB.

- PCB: hacemos referencia al circuito en si, esto abarca el diseño de las pistas y los componentes que utilizamos (en este caso Arduino Mega ADK, sensores HC-SR04 y LDR).

- Bottom Layer: En el programa nos marca como van a estar las pistas de cobre.

- Top Layer: contorno de los componentes con sus designadores.

- Multi Layer: marca los pads donde van los huecos para poner los pins de los componentes.

- Kep-Out Layer: marca el contorno de la plaqueta.

- FootPrint: tamaño de los componentes con sus pines así como la disposición del mismo.

- Pines: patas de los componentes.

- Net List: cada pista del Bottom Layer que sea continua, no importa la forma tiene un código o Net List.

7.1.1.1. Diseño de Esquemático

En primer lugar se define el esquemático generando un nuevo proyecto y abriendo un nuevo archivo de extensión .schdoc, definiendo todos los pines existentes en Arduino, HC-SR04 y LDR. Para establecer las cajas negras que definirán cada componente existen 2 maneras:

- Selección desde una librería existente.
- Diseño del componente.

En este caso Arduino Mega ADK no existe como componente en ninguna librería compatible con Altium por lo que se procede al diseño de su esquemático. Para ello se selecciona en la librería un circuito genérico de 64 pines el cual podrá editarse dando a cada pin su etiqueta correspondiente en función del pin de Arduino al que representa. Por otro lado añadimos los 5 sensores HC-SR04 así como los dos sensores LDR. El componente restante es el integrado puente H L293D, existe librería predefinida para este componente por lo que su inclusión es inmediata

Una vez añadidos cada puerto se les asigna sus características así como el designador y el modo (INPUT/OUTPUT). Una vez realizado este paso, se debe llevar a cabo la interconexión para todos y cada uno de los puertos haciendo uso de la herramienta.

Es necesario analizar los pines disponibles en Arduino para por orden ir definiendo éstos en el archivo esquemático.

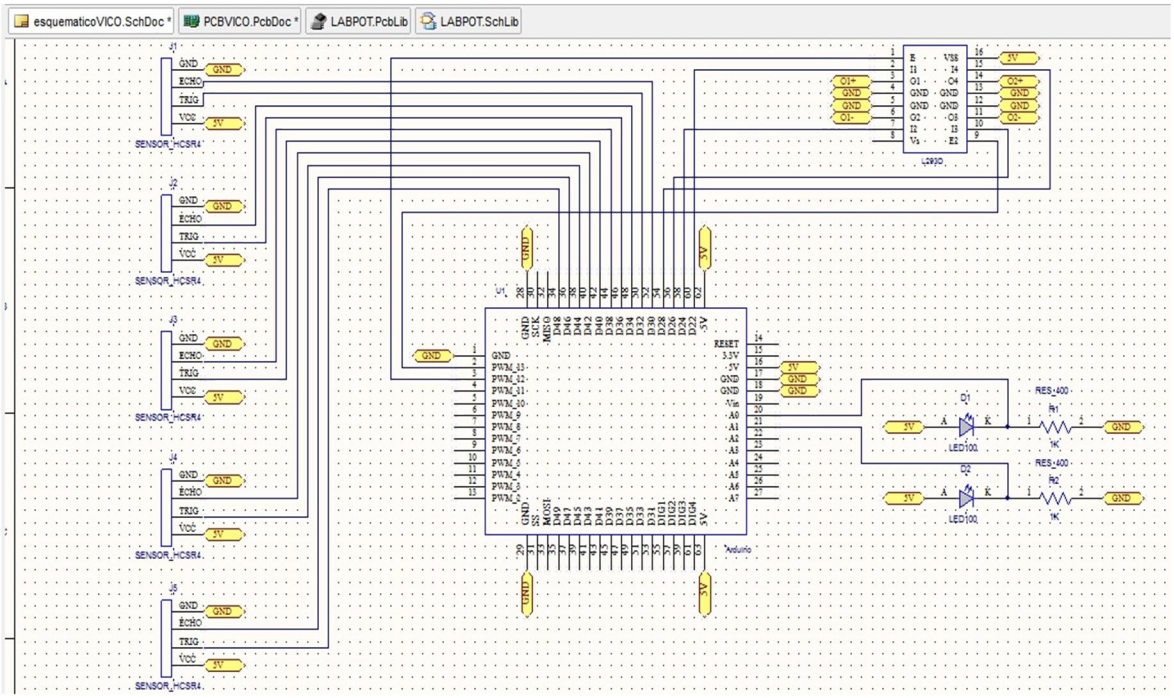
Al editar los valores de cada pin establecemos la huella por defecto definida para cada uno. En este caso la huella predefinida medida para los pines de Arduino es de 5 mm. A continuación añadimos las resistencias de 1 KOhmio del circuito LDR y les asignamos su valor.

El siguiente paso es el de asignar a los pines correspondientes la Netlist. En este caso conectaremos a los pines de tierra la etiqueta definida GND haciendo lo propio para los pines de alimentación de circuito de Arduino a 3,3 y 5 Voltios respectivamente.

Una vez definidos todos los componentes y Netlist, se llevará a cabo el conexionado de los diferentes componentes hacia los puertos requeridos de Arduino (Digitales, Analógicos y PWM) según se estableció en el apartado donde se han definido los dos modelos de robot implementados.

Una vez conectado todo el circuito se obtienen el siguiente esquemático establecido en la Figura 30.

Figura 30 Esquemático del robot



7.1.1.2. Diseño del layout del circuito

Una vez definido el esquemático mediante el archivo .schdoc se puede proceder al diseño del layout como tal del circuito. Para ello, creamos en el proyecto actual un nuevo archivo .PCBDOC donde se generarán desde el esquemático las huellas de todas las conexiones establecidas previamente para cada componente.

Tras esto, se hace uso de la herramienta Update PCB Document "nombre del archivo PCB". Una vez validado el listado de pines y sus valores se puede establecer desde el archivo PCBdoc la disposición de cada uno así como la interconexión a realizar. Antes será necesario compilar el diseño esquemático y corregir los errores que Altium indique.

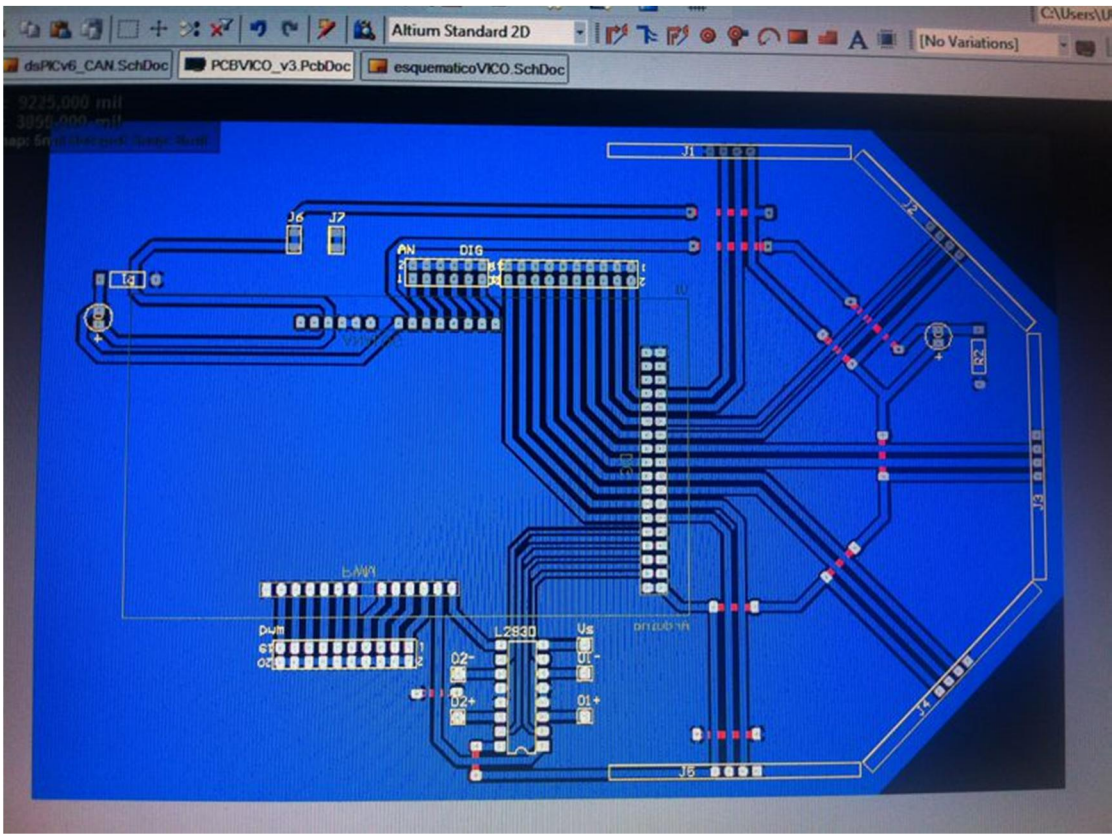
Antes de empezar a situar los elementos del circuito, pines y enrutamiento de pistas es necesario definir varios parámetros:

- Routing Widths= Ancho de las pistas un Valor Máximo, Común y Mínimo diferencia entre unidades métricas e imperiales.
- Routing Via Styles= Las dimensiones de los pads.
- Electrical Clearances= La distancia mínima entre Elementos del PCB.

Haciendo uso de la herramienta "Interactively Route Connections" se colocarán las pistas de anchura común uniendo los componentes. Previamente deberemos colocar los pads (pines) seleccionando la huella de cada componente correctamente en cada una de las ubicaciones definidas en el proyecto. En este

caso, imprimiendo una PCB de 25 cm de largo y 15 de ancho y teniendo en cuenta la restricción de colocación de los sensores HC-SR04 en un extremo de esta en 0º, 45º, 90º, 135º y 180º se obtiene el siguiente layout del circuito establecido en la Figura 31.

Figura 31 Layout de la PCB



7.1.2. Revelado de la placa y montaje de componentes.

7.1.2.1. Revelado de la placa

Una vez impreso el diseño se procede al revelado de la placa en sí. Para ello se hace uso de los siguientes elementos.

- Placa fotosensible
- Revelador
- Agua oxigenada
- Fotolito
- Cloruro férrico (no necesario, si se usa agua fuerte y agua oxigenada)
- Taladro de precisión con broca especial de 1mm.

Tras retirar el plástico de la placa tras haber sido recortada con una segueta para cumplir las dimensiones que se requieren para esta aplicación, la introducimos en una insoladora de tubos actínicos de manera que colocando el

fotolito impreso sobre la misma, quede bañada por la luz la parte blanca, es decir, sin ninguna impresión sobre el papel.

Tras unos 8 minutos de exposición a la luz procedemos a su retirada y la sumergimos en el líquido revelador. Una vez en el líquido removemos bien la placa bañándola por completo durante unos minutos. Tras esto, la placa está preparada para su introducción en la mezcla ácida preparada para para la disolución del cobre que no ha sido atacado por la luz. Tras remover durante unos 30 segundos para eliminar todo el resto de cobre sobrante, extraemos la placa de la disolución y en este caso se elimina el ácido sobrante con agua.

Tras el revelado y el taladrado de la placa sobre la localización de los pines en el diseño se procede a la limpieza con acetona de la PCB. La acetona al ser un material disolvente retira de la placa la capa de resina existente sobre las pistas de cobre. Este paso es imprescindible para poder proceder al soldado de componentes.

7.1.2.2. Montaje de componentes

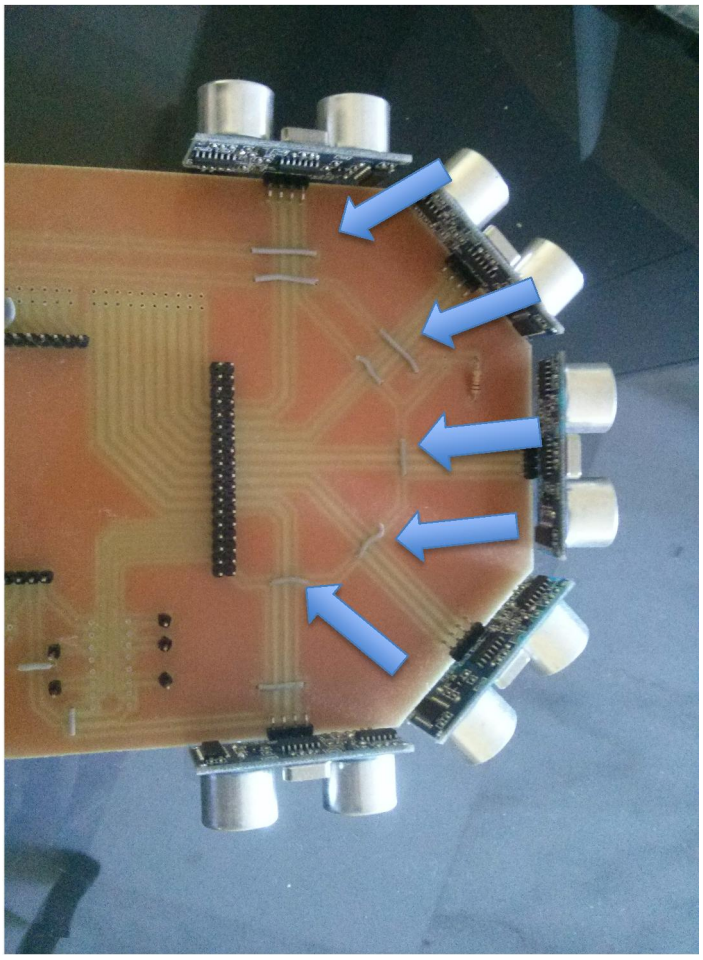
En primer lugar se procede a la inserción a través de los orificios taladrados de pines estándar de 1mm de grosor el cual se puede observar en la Figura 32.

Figura 32 Pines utilizados

A continuación, tras el soldado con estaño de toda la batería de pines utilizada para el montaje de todos los componentes (Arduino, sensores , etc) Se procede al soldado de los puentes necesarios que conectaran alimentación de 5V y tierra en hacia las pistas correspondientes de los sensores HC-SR04 así como a los circuitos de medición de la intensidad lumínica por recibida por los LDR.

Estos puentes son realizados mediante porciones de cable convenientemente cortadas tal como se indica en la Figura 33.

Figura 33 Puenteo de líneas de alimentación y GND



La necesidad de estos puentes sobre la placa son necesidades del diseño ya que era imposible llevar a cabo un diseño de PCB a una cara con un rutado que permitiese rutar las señales 5V y GND a los pines correspondientes de todos los sensores sin requerir de un salto sobre otras pistas.

Tras el montaje de todos los componentes se da por finalizado el proceso de diseño del robot en cuanto a infraestructura se requiere.

7.2.**Resultado final**

Tras el montaje de todos los elementos sobre la estructura de robot se obtiene como resultado un sistema compacto, independiente y con conectividad Wireless para la programación del nodo principal Raspberry Pi. Se trata de un

Figura 34 Montaje del robot final

Se trata de un sistema robot móvil con dos grados de libertad. El cual puede reproducir mediante un simple proceso de programación el comportamiento descrito en los apartados anteriores. Tanto como un robot evitador de obstáculos como robot gestionado mediante controlador haptico haciendo uso de los sensorse acelerómetros de un Smartphone.

El peso total del robot con todos los componentes es de 1,35 kg. Con esta solución a medida podemos distribuir sensores de manera óptima la cual permite que el funcionamiento del robot sea más exacto y eficiente.

8. Conclusiones y Posibles mejoras:

8.1. Posibilidades ofrecidas por ROS

Las posibilidades que ofrece ROS son muy susceptibles de ser aplicadas en un sistema Robot móvil. Entre ellas, existen herramientas completas y drivers para ciertos sensores que permiten afrontar a alto nivel funciones como el control de la navegación del robot y la generación de mapas.

Existe una amplia lista de robots genéricos para los cuales existe un paquete ROS asociado que implementa funciones de control de la navegación de manera inmediata. Estos son:

ROBOT	PAQUETE ROS
PR2	pr2_navigation_global
TurtleBot	turtlebot_navigation
Care-O-bot	cob_navigation
rob@work	cob_navigation
Kurt	kurt_navigation_global
Volksbot	to be published
Pioneer 3AT	navigation.launch
youBot	youbot_navigation
Segbot	segbot_navigation (Documentation coming soon)
Clearpath Husky	husky_navigation
Clearpath Grizzly	To be released
Wubble2	wubble_navigation_apps
Yujin Robot Robosem, Homemate & Topseung	Documentation not available
REEM & REEM-C	Not released
GROK2	Launchpad
Robotnik Guardian	guardian_2dnav
Robotnik Summit XL, XL OMNI, X-WAM	summit_xl_2dnav
Powerbot	2dnav_powerbot
Cerberus	Proprietary
Pi Robot	In development...
QBO Robot	qbo_navigation
Pioneer 3D-X	pioneer_2dnav (using ROSARIA)

Stingbot and Traxbot	mrl_robots_2dnav
Otter-4 (RobOtter Club Hamburg)	In development...

8.2.**Navigation stack**

La navigation stack de ROS es una aplicación ROS la cual toma información odométrica (aceleración lineal, velocidad lineal) medida de sensores de proximidad y una ubicación objetivo y proporciona para llegar a esta unos comandos de velocidad adecuados hacia la base móvil del robot.

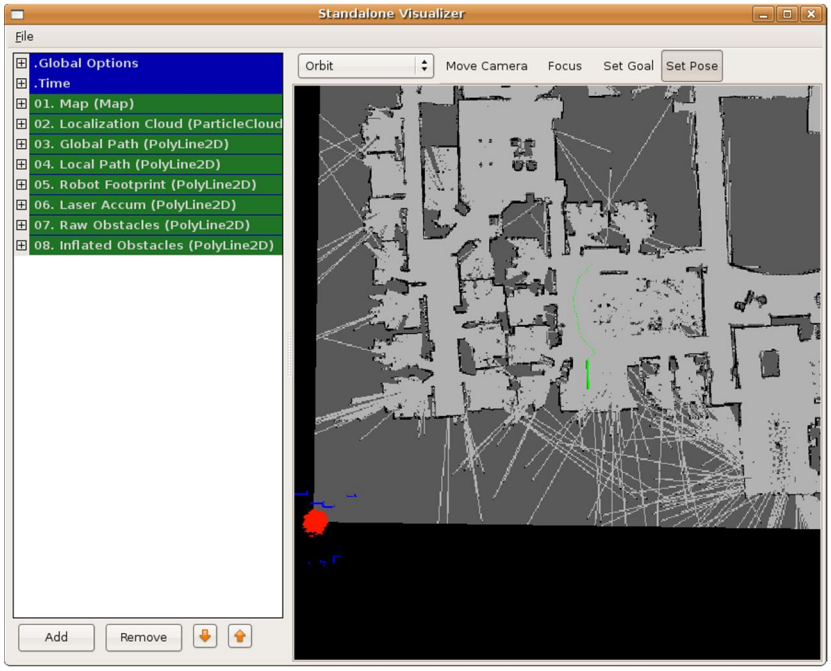
Según el soporte existente actualmente para los paquetes navigation de ROS, se requiere por ahora de la disponibilidad de escáneres laser de medida de la proximidad para su integración. En caso de dotar a nuestro robot de este hardware sería totalmente sencillo y casi inmediato dotar a nuestro robot de esta funcionalidad. Tan solo es necesario seguir varios pasos que incluyen:

- Configuración del modelo de transformadas del robot.
- Configuración de la publicación de datos odométricos.

La herramienta gráfica de ROS asociada al paquete navigation es Rviz. Esta herramienta de visualización permite la visualización en una recreación virtualizada del robot tanto en 2D como en 3D, representación de obstáculos en el mapa, generación de los mismos, introducción de datos de ubicación objetivo así como planificación de caminos a introducir.

Para el uso de esta herramienta es necesario contar con un entorno previamente preparado, es decir, un escáner laser así como la configuración de transformadas del robot y la recepción de datos odométricos.

Figura 35 Captura del menú de herramienta Rviz



Otra de las herramientas muy utilizadas en este caso y disponible en ROS es la herramienta generadora de mapas gmapping.

Esta herramienta permite, basándose en los datos recibidos de mensajes laser_scan así como de las transformadas de los datos odómetros generar una nube de puntos interpretable por rviz como un mapa.

8.3.**Aplicaciones de los modelos**

Aprovechando las herramientas que ROS pone a nuestro alcance así como el algoritmo conjunto implementado para la evitación de obstáculos así como los sensores de luminosidad se puede decir que el robot implantado podría tener aplicaciones sencillas en cualquier entorno que requiera la activación de un trabajo programado que consista en la ejecución de un recorrido periódico y aleatorio que cubra toda la ubicación.

- Robot detector de intrusos (Figura 36)
- Robot aspirador.(Figura 37)
- Limpiafondos de piscina.
- Robot generador de mapas mediante ROS.
- Robot de movimiento autónomo

Figura 36 APLICACIONES

8.4.**Conclusiones**

En la programación de Robots móviles, encontramos en ROS una herramienta de gran potencia en cuanto nos ofrece un framework de programación de alto nivel para ciertas funciones las cuales suponen una complejidad añadida las cuales pasan a ser transparentes. Todo esto, haciendo uso de una plataforma libre instalable sobre cualquier máquina capaz de correr un sistema operativo GNU/Linux.

La comunidad de desarrollo de herramientas y paquetes ROS está centrada en la construcción de librerías y funciones que supongan para robots comerciales la programación de servicios y funcionalidades integrando éstos con sensores comerciales de manera transparente al programador. Por lo tanto, la implementación de un robot customizado supone tener que dar un paso extra en la programación de ciertas funciones no estandarizadas si no se dispone de un robot comercial como los indicados en apartados anteriores. Por ejemplo, la navegación y la capacidad para generación de mapas.

En el presente proyecto se implementan dos diseños sobre un robot totalmente customizado haciendo uso de una plataforma embebida de bajo costo como es Raspberry Pi que hará las veces de controlador y utilizando elementos genéricos como son los sensores LDR y HC-SR04.

En este caso, ciertas herramientas de ROS son muy útiles a la hora de programar procesos claves como la comunicación entre procesos, la encapsulación y la reconstrucción de mensajes, así como la posibilidad de generación de librerías para poder construir fácilmente mensajes customizados de información a intercambiar entre los diferentes nodos del robot. Esto obviamente supone un horizonte en el que ROS facilita la programación de un robot comercial o en este caso custom poniendo esta a un alto nivel accesible por usuarios a los que no se requerirá un nivel de expertise tan elevado.

Por otro lado, ROS ofrece herramientas de monitorización de los nodos establecidos y comunicación entre los mismos lo cual facilita de gran manera el debugging de las aplicaciones.

Además, la posibilidad mediante rosbag de guardar información circulante en el sistema hace de ROS una herramienta que en caso de requerir una aplicación con una cierta estandarización de funciones proporciona una abstracción y funciones plug&play más que interesante para el desarrollo de robots y aplicaciones caseros y comerciales.

9. Bibiografía

1. **ROS Powering the world Robots**

   www.ros.orh

2. **Wiki ROS**

   Wiki.ros.org

3. **Questions—ROS Answers**

   Answers.ros.org/questions

4. **Raspberry Pi wiki**

   www.raspberrypi.org

5. **Arduino wiki**

   www.arduino.cc

6. **Wildcircuits**
   www.wildcircuits.com/

7. ***ROS An Open Source Robot Operating System***: Computer Science Department Stanford University. Computer Science Department, University of Southern California. Morgan Quingley , Brian Gerzkey, Ken Conley , Josh Faust.

8. ***Distributed algorithms for guiding navigation across a sensor network.*** Dartmouth College. Qun Li, Michael De Rosa, Daniela Rus

9. **The C++ Resource Network**

   www.cplusplus.com

10. **Github- Build software**

    Github.com