



# Trabajo de Fin de Grado

---

## Sistema de navegación autónoma para un robot diferencial

*Autonomous navigation system for a differential robot*  
Andrés Cidoncha Carballo

---

La Laguna, 4 de julio de 2016

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78698554-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

## C E R T I F I C A (N)

Que la presente memoria titulada:

*“Sistema de navegación autónoma para un robot diferencial”*

ha sido realizada bajo su dirección por D. **Andrés Cidoncha Carballo**, con N.I.F. 54113787-P.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de julio de 2016.

## Agradecimientos

A Jonay, por su paciencia y su ayuda indispensable para comprender los apartados y conceptos más complicados de todos los necesarios para la realización de este trabajo, así como por ser la fuente de inspiración para sentirme apasionado por este tema.

A mi familia, por ayudarme a tener disponibles los recursos necesarios para realizar este trabajo.

Y, sobre todo, a Natalia, por ser mi principal fuente de apoyo, aguantarme estresado y ayudarme a mantener la calma en los momentos más complicados del desarrollo.

Mención especial a mis compañeros Fabián y Miguel, pues, aunque nuestros trabajos fueran diferentes, compartir el gusto por este tema ha resultado muy enriquecedor.

# Licencia



© Esta obra está bajo una licencia de Creative Commons  
Reconocimiento 4.0 Internacional.

## **Resumen**

*El objetivo de este trabajo ha sido el diseño y el montaje de un sistema autónomo de navegación para un robot diferencial. En este trabajo, se ha realizado un estudio a bajo nivel del funcionamiento de la electrónica, además de la configuración de un sistema empotrado para la ejecución del software necesario para el desarrollo de la navegación autónoma.*

*Este sistema se basa en el uso del framework de desarrollo para sistemas robotizados ROS, montado sobre un sistema empotrado ODROID; así como en el uso de un sistema Kinect para la realización del mapa de entorno.*

*Todo el desarrollo se ha realizado en sistemas GNU/Linux en los lenguajes C++ y Python.*

**Palabras clave:** *ROS, ODROID, Kinect, GNU/Linux, C++, Python.*

## **Abstract**

*This project's objective has been the making of an autonomous navigation system for the AmigoBot, a little robot that Jonay gave me for the development of this End Of Grade work. The autonomous navigation system is based on ROS, a framework for the development of robot systems, mounted on an Odroid system; as in the use of a Kinect system for the making of an environment map.*

*At the beginning, the system was going to use the AmigoBot's original electronic, but due to several errors, I had to implement a new one based on the use of an Arduino board and an H bridge module.*

**Keywords:** *ROS, Odroid, Kinect, GNU/Linux, C++, Python.*

# Índice General

<b>Capítulo 1. Introducción</b>	<b>5</b>
1.1 Contextualización.....	5
1.2 Objetivos.....	5
1.3 Herramientas utilizadas .....	6
<b>Capítulo 2. Introducción a ROS</b>	<b>7</b>
2.1 Que es ROS.....	7
2.2 Como funciona .....	7
2.3 Sistemas de navegación con ROS .....	8
2.4 Navegación autónoma con ROS.....	9
<b>Capítulo 3. Presentación del AmigoBot</b>	<b>10</b>
3.1 Vistazo General.....	10
3.2 Electrónica de motores .....	11
3.3 Sónares.....	11
3.4 AmigoBot en ROS (Rosaria) .....	12
<b>Capítulo 4. Sistema Kinect</b>	<b>13</b>
4.1 Características.....	13
4.2 Funcionamiento.....	13
4.3 Ventajas e Inconvenientes .....	14
4.4 Kinect en ROS (openni_kinect) .....	14
<b>Capítulo 5. Sistema ODROID</b>	<b>16</b>
5.1 Características.....	16
5.2 Preparación .....	17
<b>Capítulo 6. Electrónica con Arduino</b>	<b>18</b>
6.1 Driver L298N .....	18

6.2	Encoders.....	19
6.3	Montaje.....	19
6.4	Arduino con ROS.....	23
<b>Capítulo 7.</b>	<b>Sistema de navegación autónoma</b>	<b>24</b>
7.1	Funcionamiento.....	24
7.2	Preparación .....	24
<b>Capítulo 8.</b>	<b>Conclusiones y líneas futuras</b>	<b>25</b>
<b>Capítulo 9.</b>	<b>Summary and Conclusions</b>	<b>26</b>
9.1	First Section.....	26
<b>Capítulo 10.</b>	<b>Presupuesto</b>	<b>27</b>
10.1	Sección Uno.....	27
<b>Apéndice A.</b>	<b>Título del Apéndice 1</b>	<b>28</b>
A.1.	Algoritmo XXX.....	28
A.2.	Algoritmo YYY.....	33
<b>Apéndice B.</b>	<b>Título del Apéndice 2</b>	<b>37</b>
B.1.	Otro apendice: Seccion 1 .....	37
B.2.	Otro apendice: Seccion 2 .....	37
<b>Bibliografía</b>		<b>38</b>

# Índice de figuras

Figura 1. Funcionamiento de los <i>temas</i> en ROS.....	8
Figura 2. Grafo de un sistema de navegación.....	8
Figura 3. Grafo de sistema de navegación autónoma .....	9
Figura 4. AmigoBot.....	10
Figura 5. Vista superior de los motores.....	11
Figura 6. Esquema de los sónares.....	11
Figura 7. Circuito base de los sónares.....	12
Figura 8. Matriz de puntos infrarrojos.....	14
Figura 9. ODROID-U3.....	16
Figura 10. Módulo de driver L298N y puente H.....	18
Figura 11. Esquema de conexión del módulo L298N.....	19
Figura 12. Conectores del regulador de voltaje.....	20
Figura 13. Conectores de alimentación de la electrónica.....	20
Figura 14. Módulo de leds e interruptores.....	21
Figura 15. Montaje final.....	22

# Índice de tablas

Tabla 1. Conexión de los sónares .....	21
Tabla 2. Conexión de los leds/interruptores.....	22
Tabla 3. Conexión de los encoders y el módulo L298N.....	22
Tabla 4. Tabla resumen de los Tipos. ....	27

# Capítulo 1. Introducción

## 1.1 Contextualización

El Grupo de Robótica de la Universidad de La Laguna, es un grupo de investigación pionero en el archipiélago por el desarrollo de algunos sistemas, como el coche autónomo (Verdino) o en la silla de ruedas (Perenquén). Es con el contacto con miembros de este grupo, y con la demostración del funcionamiento de algunos de estos desarrollos, de donde surge el deseo de conocer más a fondo el funcionamiento de los sistemas de navegación autónoma.

Por ello, este Trabajo de Fin de Grado, se marca como objetivo primordial, la comprensión de todos los elementos involucrados en el funcionamiento de un sistema de navegación autónoma, así como la obtención de las aptitudes necesarias para el desarrollo de un sistema de estas características.

Para el desarrollo de este trabajo, los elementos hardware principales con los que se cuentan son: un robot AmigoBot de la marca MobileRobots, un sistema Kinect de Microsoft y un ODROID U3. En la parte software, todo el desarrollo se basará en el uso del framework de desarrollo ROS.

## 1.2 Objetivos

Los objetivos específicos de este trabajo son los siguientes:

- Comprensión del funcionamiento y utilización de la electrónica del AmigoBot.
- Instalación y configuración de ROS (en su versión Indigo) en un sistema empotrado Odroid U3+.
- Instalación, montaje y configuración de la Kinect en el entorno de trabajo ROS.
- Instalación y configuración de los nodos necesarios para el sistema de navegación autónoma en ROS.

### **1.3 Herramientas utilizadas**

Para la realización de este Trabajo de Fin de Grado, se han utilizado las siguientes herramientas:

- Hardware:
  - Robot AmigoBot de MobileRobots
  - Kinect v.1 de Microsoft
  - ODROID U3
  - Portátil Acer Aspire 5755G
- Software:
  - Ubuntu 16.04 LTS (Xenial)
  - Ubuntu 14.04 LTS (Trusty)
  - Editor Atom
  - Arduino IDE
  - ROS Indigo

# Capítulo 2.

# Introducción a ROS

## 2.1 Que es ROS

ROS, siglas de Robot Operating System, es un framework para el desarrollo de software para robots basado en una arquitectura de grafos y distribuido bajo licencia BSD. Nos proporciona una serie de capacidades como son abstracción del hardware, un servidor de parámetros, un sistema de intercambio de mensajes, etc. Tiene una comunidad muy extendida, a la vez que una cantidad de recursos y documentación impresionante, por lo que es la base de muchos proyectos de robótica compleja.

Los paquetes en ROS representan la manera en que el software desarrollado se distribuye, de manera que un paquete puede contener nodos, librerías, etc. De esta manera, encontramos paquetes de ROS para, por ejemplo, obtener toda la información que nos proporciona una Kinect (mapa de profundidad, imagen a color, etc). El objetivo es proveer funcionalidades fácilmente reutilizables.

Los nodos en ROS son básicamente, procesos que realizan un cómputo, ejecutables dentro de un paquete que utilizan las librerías cliente de ROS para la comunicación con otros nodos.

## 2.2 Como funciona

ROS tiene un nodo principal, *roscore*, encargado de cargar todos los procesos necesarios para que cualquier nodo pueda ser ejecutado con ROS. Entre ellos se encuentra un nodo *master*, encargado de mediar en el intercambio de mensajes, que está basado en un sistema de publicación/suscripción a temas (podríamos considerar los temas como buses identificados).

Cuando un nodo comienza a publicar datos en un tema, avisa al nodo *master*, para, en caso de que algún nodo quiera suscribirse a ese tema, le indique a

donde debe ir a buscar los datos. En el orden inverso, indica al nodo publicador donde están esperando los datos de ese tema.

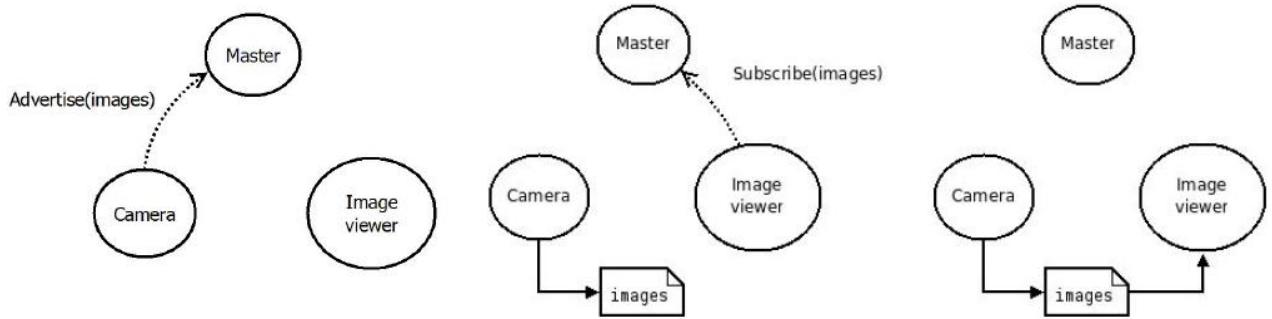


Figura 1. Funcionamiento de los *temas* en ROS.

## 2.3 Sistemas de navegación con ROS

Comprendiendo cómo funciona el sistema de paso de mensajes de ROS, podemos pasar a idear como sería idealmente el grafo de un sistema de navegación con ROS. Para ello, y basándonos en el uso de un robot diferencial, se hacen necesaria una serie de nodos:

- Nodo de control de movimiento
- Nodo de odometría y sensores
- Nodo de mapa de navegación
- Nodo de visualización y control

Un esquema simple del grafo de comunicación entre los distintos nodos sería el siguiente:

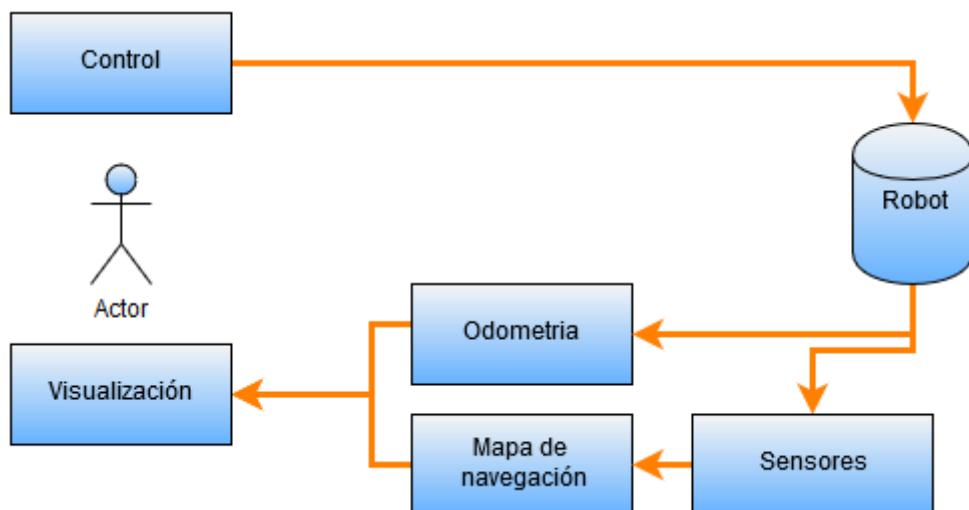


Figura 2. Grafo de un sistema de navegación.

El problema de este modelo es que requiere un operador humano, que este pendiente de la información que devuelve el robot además de tener que decidir que ruta tomar y que instrucciones debe dar al robot para seguir dicha ruta.

## 2.4 Navegación autónoma con ROS

El problema descrito anteriormente puede ser fácilmente solucionado si utilizamos algún nodo de navegación autónoma, ROS dispone de varios basados en distintos algoritmos, pero el que me resulta más interesante para este proyecto es el *slam\_gmapping*, basado en SLAM (Simultaneous Localization And Mapping), que se encarga de realizar el mapa de navegación a la vez que toma las decisiones sobre el movimiento del robot.

El grafo quedaría así:

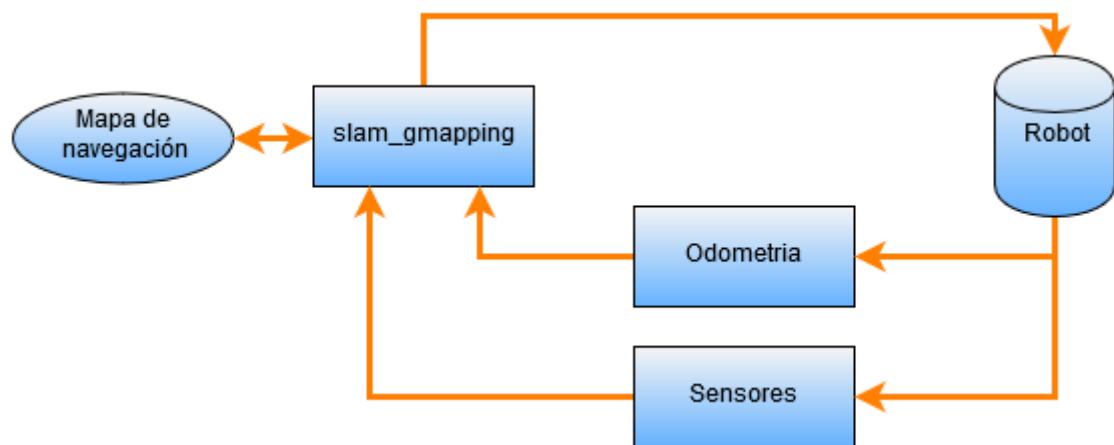


Figura 3. Grafo de sistema de navegación autónoma.

# Capítulo 3.

# Presentación del AmigoBot

El AmigoBot es un pequeño robot diferencial orientado a la enseñanza fabricado por MobileRobots. Para este trabajo, tomaremos como base su electrónica para el montaje del sistema de navegación, aprovechando de él su electrónica de motores, su sistema de sónares y su batería.



Figura 4. AmigoBot.

## 3.1 Vistazo General

El AmigoBot es un robot de gama básica, pero aun así viene con unas características muy completas de cara a este proyecto:

- Dos motores con codificadores integrados.
- 8 sónares (6 frontales y 2 traseros).
- Batería de 2.3 Ah.
- Cámara.
- Altavoz.
- Conjunto de tres leds y dos interruptores.
- Sistema de comunicación serial y librería propia (LibAria).

## 3.2 Electrónica de motores

Entrando más en detalle sobre la electrónica de motores, el AmigoBot dispone de dos motores reversibles de alta velocidad y alto empuje. Cada uno de ellos se alimenta a 12 Voltios e incluye un codificador óptico de alta resolución, que proporcionan 9550 tics por vuelta de rueda (aproximadamente 30 tics por milímetro).



Figura 5. Vista superior de los motores.

## 3.3 Sónares

El AmigoBot cuenta con un total de 8 sónares, 6 frontales y 2 traseros, que proporcionan información de distancia en un rango entre 10 cm y 3 metros en un campo de prácticamente 360° al robot de cara a su desplazamiento. El ratio de escaneo es de 25 Hz (40 milisegundos por sonar) y la sensibilidad se puede calibrar por medio de un potenciómetro integrado en el circuito.

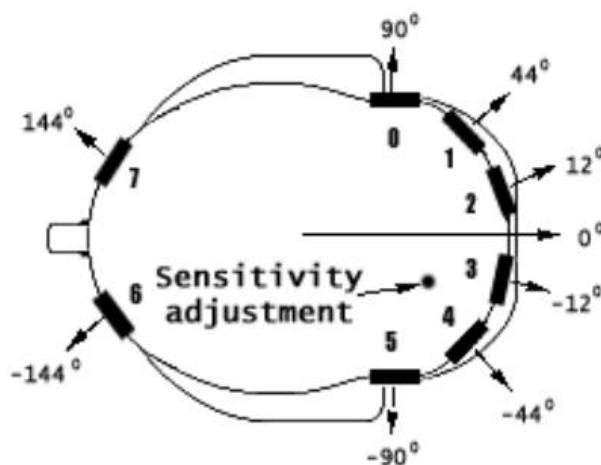


Figura 6. Esquema de los sónares.

La información de distancia que nos proporciona cada sónar se obtiene a través de un pequeño circuito serializado, basado en el uso de un TL852CN y un TL851.

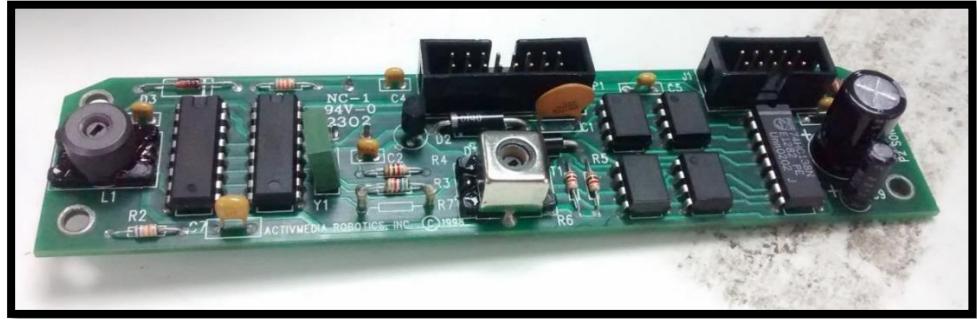


Figura 7. Circuito base de los sónares.

### 3.4 AmigoBot en ROS (Rosaria)

De cara al uso del AmigoBot en un sistema de navegación en ROS, existe un paquete llamado ROSARIA destinado a crear todos los *temas* necesarios para aprovechar todas las capacidades del robot.

Su instalación es sencilla, clonar el repositorio del paquete en el *workspace* de ROS y realizar un *catkin\_make*. Para ejecutar el nodo, teniendo el AmigoBot conectado al ordenador, ejecutamos *rosrun rosaria RosAria*. Por defecto el sistema buscará el AmigoBot en */dev/ttyUSB0*, pero se puede especificar el puerto añadiendo al comando el parámetro *\_port:=/dev/ttyS0*.

Una vez iniciado, el nodo conectará con el robot y comenzará a publicar la información recibida en los siguientes temas:

- *pose*: Publica la información de la odometría
- *sonar\_pointcloud2*: Publica la nube de puntos que recoge de los sónares.
- *battery\_voltage*: Publica el voltaje actual de la batería.
- *battery\_recharge\_state*: Publica el estado de recarga de la batería (0 si no está cargando, >0 si está cargando y <0 con error o sin datos).
- *motors\_state*: Publica el estado de los motores (activo o inactivo).

Además, se suscribirá al tema *cmd\_vel* (para recibir los comandos de movimiento) y proporcionará dos servicios: *enable\_motors* y *disable\_motors*, para activar y desactivar los motores respectivamente.

# Capítulo 4.

# Sistema Kinect

Para obtener información del entorno se ha escogido como dispositivo principal una Kinect v1 de Microsoft, puesto que dispone de una serie de características (detalladas más adelante) que nos permitirán recoger de forma fiable la información necesaria para la construcción de un mapa de navegación. Este dispositivo fue lanzado periférico para la consola XBOX 360, pero gracias a su utilidad y bajo coste, se ha popularizado su uso en robótica como medio de obtención de odometría visual.

## 4.1 Características

- Campo de visión (horizontal/vertical): 57° x 43°
- Rango de profundidad: 0.8 m – 4 m
- Emisor y receptor de IR
- Matriz de micrófonos: Audio 16-bits @ 16 kHz
- 320 x 240 16-bits de profundidad @30 fps
- 640 x 480 32-bits de color @30 fps
- Latencia: 90 ms

## 4.2 Funcionamiento

El modo de funcionamiento de este dispositivo es el siguiente:

1. La cámara emite una matriz de puntos infrarrojos, invisibles al ojo humano. Estos puntos, una vez impactan en una superficie, rebotan.
2. El sensor de infrarrojos recoge el rebote de los puntos, y midiendo el tiempo de llegada de cada punto desde el momento de salida, es capaz de determinar la distancia a la que se encuentra el objeto donde ese punto ha rebotado.

3. El procesador integrado se encarga de procesar esta información y crear el mapa de profundidad.

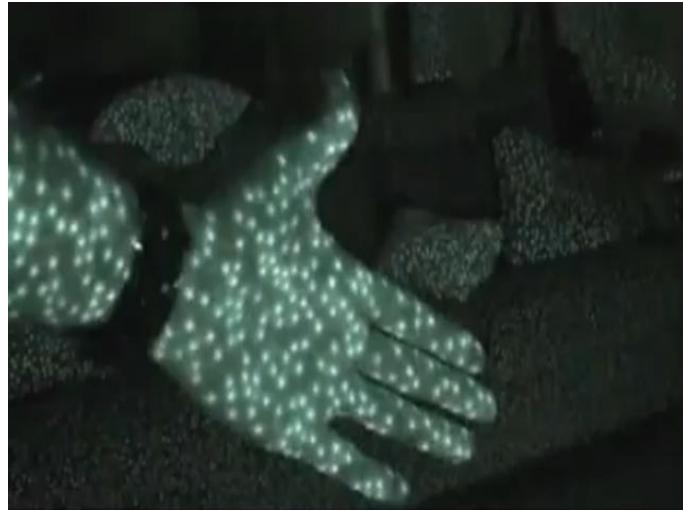


Figura 8. Matriz de puntos infrarrojos.

### 4.3 Ventajas e Inconvenientes

Como ventajas de este dispositivo tenemos su bajo coste, la fiabilidad de la información que nos proporciona y los buenos resultados que consigue en entornos de interior.

Como inconveniente, en exteriores la luz solar, por tener un alto nivel de radiación infrarroja interfiere demasiado afectando al sensor, dejándolo prácticamente ciego.

### 4.4 Kinect en ROS (`openni_kinect`)

Para el uso de la Kinect en el entorno ROS, necesitaremos la instalación de dos *stacks*: *openni\_camera* y *openni\_launch*. El sentido de esto es que el paquete *openni\_launch* proporciona procesamiento adicional para el RGB-D producido por el nodo *openni*, además de simplificar el arranque del nodo. Para arrancarlo, se debe ejecutar: *roslaunch openni\_launch openni.launch*. Una vez arrancado, el nodo nos proporciona información a través de una serie de temas:

- RGB:
  - *rgb/camera\_info*: Datos de calibrado y metadados.
  - *rgb/camera\_raw*: Imagen en formato Bayer GRBG.

- Profundidad (publica cuando  $\sim depth\_registration$  es false):
  - $depth/camera\_info$ : Datos de calibrado y metadatos.
  - $depth/image\_raw$ : Profundidades en mm en enteros de 16 bits
- Profundidad alineada con RGB (publica cuando  $\sim depth\_registration$  es true):
  - $depth\_registered/camera\_info$ : Datos de calibrado y metadatos. Lo mismo que  $rgb/camera\_info$  pero sincronizado con  $depth\_registered/image\_raw$ .
  - $depth\_registered/image\_raw$ : Profundidades en mm en enteros de 16 bits.
- Infrarrojos:
  - $ir/camera\_info$ : Datos de calibración y metadatos.
  - $ir/image\_raw$ : Imagen infrarroja en crudo (uint16).
- Proyector Infrarrojo:
  - $projector/camera\_info$ : Lo mismo que  $depth/camera\_info$  pero con la línea base codificada en la matriz P.

Además, nos proporciona los siguientes servicios:  $rgb/set_camera_info$  e  $ir/set_camera_info$ . Usados para ajustar la calibración de las cámaras RGB e infrarrojas respectivamente.

# Capítulo 5.

# Sistema ODROID

Para realizar el sistema de navegación, necesitaremos un sistema empotrado que sea el encargado de manejar todos los nodos necesarios en el robot como son el de la electrónica y el de la Kinect.

Para esta tarea hemos escogido el ODROID U3, un sistema pequeño pero potente que dispone de todas las características necesarias para el desempeño de esta tarea.

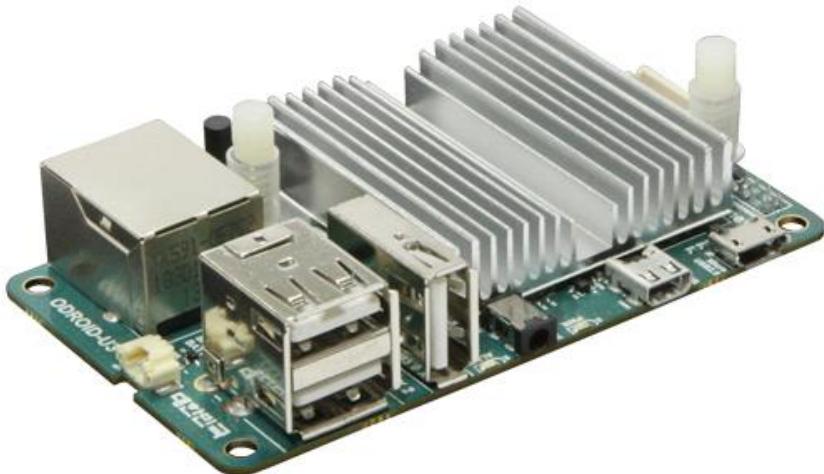


Figura 9. ODROID-U3

## 5.1 Características

- Cuatro núcleos a 1.7GHz y 2 GB de RAM
- 10/100 Mbps Ethernet con conector RJ-45
- 3 x USB 2.0 + micro HDMI + micro USB 2.0
- Códice de Audio con salida jack
- Pines GPIO/UART/I2C
- Xubuntu 14.04.4 (instalado en una microSD de 8 GB)
- Tamaño: 83 x 48 mm, Peso: 48g con disipador

## 5.2 Preparación

De cara a tener el ODROID preparado para el sistema final, se debieron realizar una serie de tareas:

1. Configurar la interfaz de red inalámbrica (mediante el módulo USB oficial) para tener acceso a Internet.
2. Al disponer de un espacio de almacenamiento de 8 Gb para el sistema y los paquetes que deseemos instalar, la primera tarea a realizar fue una limpieza de los paquetes innecesarios en el sistema para ahorrar espacio. Se eliminaron, por ejemplo, GIMP, Chromium, LibreOffice...
3. Una vez liberado todo el espacio posible, pasamos a la instalación de ROS. En este caso, se instalará la versión *Indigo*, que es la compatible con la versión 14.04 de ubuntu, usando el siguiente comando:

```
sudo apt-get install ros-indigo-desktop-full
```

4. Una vez instalado el entorno ROS, debemos preparar un directorio de trabajo para realizar la construcción de los paquetes que no estén incluidos en los repositorios oficiales. Para ello, debemos crear un directorio en el *home* de nuestro equipo (de nombre *ros\_ws*, en nuestro caso) y dentro de este, un directorio llamado *src*. Este último directorio será donde deberemos colocar el código fuente de los paquetes (en su subdirectorio correspondiente) para posteriormente volver al directorio *ros\_ws* y hacer un *catkin\_make* desde ahí. Una vez finalice, los paquetes ya estarán instalados y preparados para su uso.

Es importante añadir al archivo de perfil de la terminal (zsh en mi caso) las siguientes líneas para asegurarnos que todos los paquetes instalados estarán incluidos en el PATH de ROS:

```
source /opt/ros/indigo/setup.zsh  
source ~/ros_ws/devel/setup.zsh
```

# Capítulo 6.

# Electrónica con Arduino

En el capítulo 3 se describió la electrónica de motores incluida en el AmigoBot y de la que se hacía uso mediante el paquete ROSARIA, pero a mitad de realización del trabajo, esta electrónica dejó de funcionar completamente, por lo que se debió buscar una alternativa. Se decidió montar una nueva electrónica de motores mediante el uso de un Arduino UNO y un módulo basado en el driver L298N y el uso de un puente H, que mediante el código del proyecto *diff\_arduino\_drive* y el nodo *rosserial\_arduino*, permite emular el funcionamiento de ROSARIA.

## 6.1 Driver L298N

Este módulo, basado en el chip L298N nos permite controlar dos motores de corriente continua o un motor paso a paso bipolar. Dispone de diodos de protección, así como de un regulador LM7805 para proveer los 5 Voltios de la alimentación de la parte lógica, activable mediante un jumper. En caso de estar desactivado, se debe alimentar el módulo entre 12 y 35 Voltios, además de proporcionar 5 Voltios en la entrada correspondiente. En caso contrario, simplemente hay que alimentar el módulo a un voltaje entre 6 y 12 Voltios, por lo que este modo es el que nos resulta más cómodo para este trabajo.

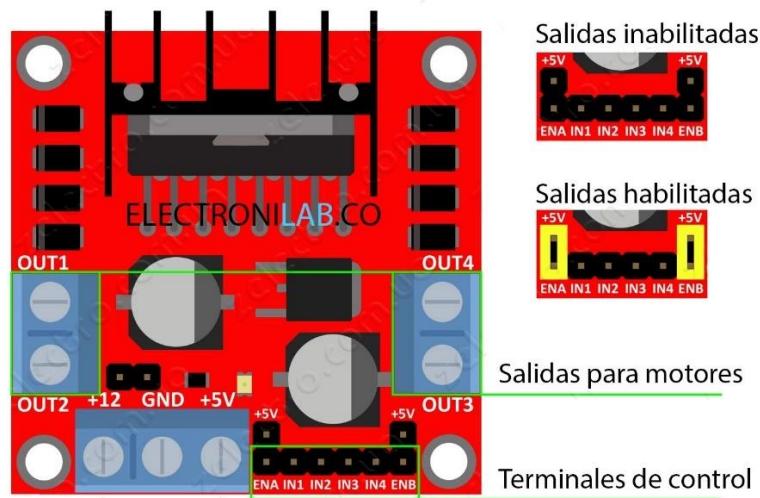


Figura 10. Módulo de driver L298N y puente H.

Para el control de un motor mediante PWM con este módulo, debemos usar los pines IN correspondientes a ese motor (por ejemplo, para la salida B, serian los IN3 e IN4), aplicarles corriente a uno u a otro (10 o 01) para seleccionar un sentido de giro del motor, y luego aplicar la PWM al pin de enable de ese motor (en este caso, ENB).

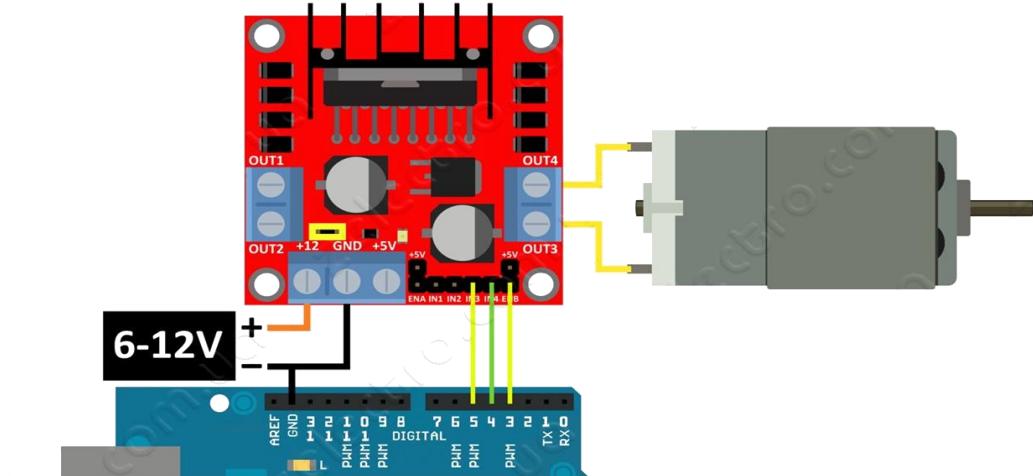


Figura 11. Esquema de conexión del módulo L298N.

## 6.2 Encoders

Para la integración de los encoders en la nueva electrónica, se dispone de una librería *Encoder* preparada para varias configuraciones y que nos simplifica en extremo el control de estos dispositivos. Se alimentan a 5 Voltios, y las dos salidas de cada encoder se conectan a los pines con capacidad de interrupción del Arduino. En este caso, como en el Arduino UNO no disponemos sino de dos pines de interrupción y el resultado óptimo se obtiene con las dos salidas de cada encoder en un pin con capacidad de interrupción, se decide hacer uso del Arduino Leonardo, que dispone de cinco.

## 6.3 Montaje

Intentando aprovechar el circuito de la electrónica original del AmigoBot, puesto que integra el circuito de carga, interruptor y una serie de conectores que pueden ayudar a dejar el circuito más limpio y entendible, se realizan un par de modificaciones:

- Se quita el conversor de corriente estropeado, se sustituye por unos pines donde conectar la salida del conversor de voltaje variable (12 V a 5 V).

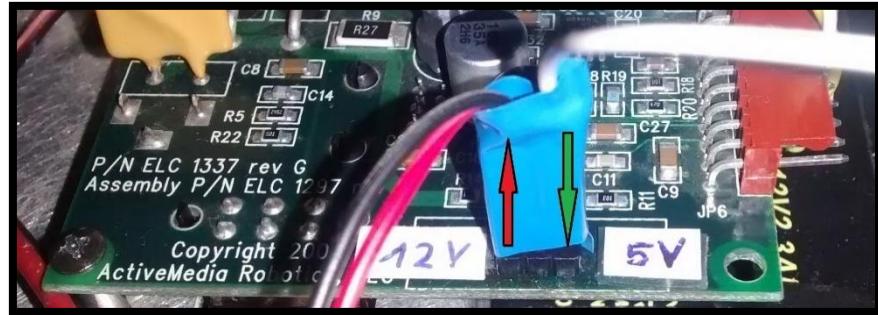


Figura 12. Conectores del regulador de voltaje.

- Se quitan los controladores originales de los motores, puesto que se calientan al usar el circuito.
- Se modifican algunos conectores para que den corriente (5 V) por más pines y así aprovecharlos para alimentar los encoders, el Arduino y el ODROID. Se utiliza un conector para llevar 12 V al módulo del L298N y a la Kinect.

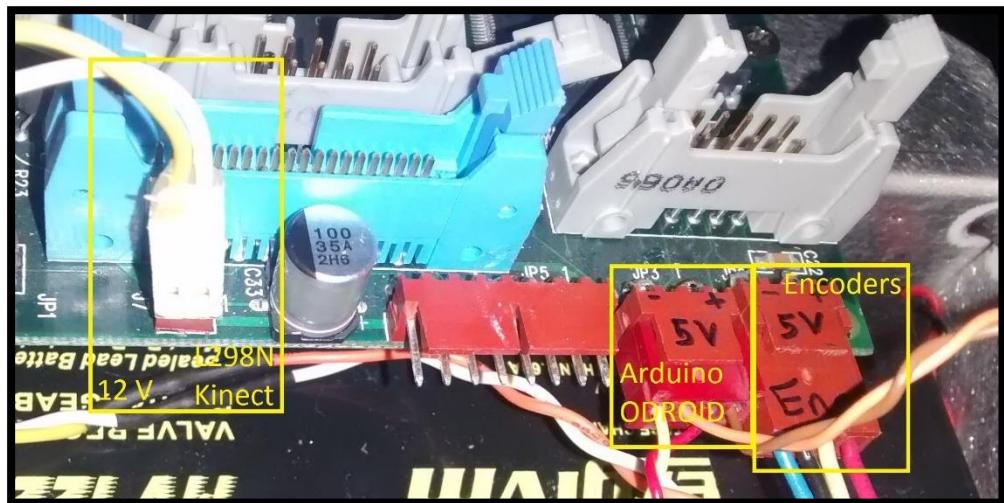


Figura 13. Conectores de alimentación de la electrónica.

De cara a tener algo de información visual del Arduino para depuración, se decide utilizar el pequeño módulo de leds e interruptores que tenía la electrónica original. El led verde (ON) indica cuando el Arduino está encendido, el led amarillo (BUSY) indica cuando el robot ha recibido algún comando y está procesándolo, y el led rojo (ERROR) se usa para indicar error. Además, el

interruptor rojo se usa para reiniciar el Arduino (está conectado directamente al pin RESET).

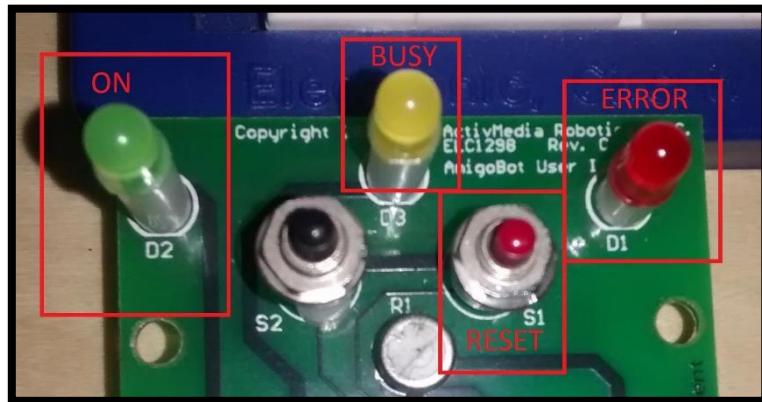


Figura 14. Módulo de leds e interruptores.

Una vez preparados todos los elementos, se procede a instalarlos en el chasis del AmigoBot. Todos los elementos que se colocan, se instalan sobre una pequeña capa de cinta aislante para evitar cualquier tipo de contacto con el chasis metálico y se fijan con cinta de doble cara. Conectando los elementos al Arduino Leonardo, se observa que los pines disponibles son insuficientes, así como la memoria para el sketch final (descrito más adelante), por lo que se decide instalar un Arduino MEGA en su lugar. Finalmente, la configuración de pines del Arduino queda así:

PIN SONAR	FUNCION	PIN ARDUINO
1	SEL0	40
2	SEL1	41
3	SEL2	42
4	BINH	44
5	INIT	45
6, 7	VCC	5V
8, 9	GND	GND
10	ECHO	43

Tabla 1. Conexión de los sónares.

PIN LEDS/SWITCH	FUNCION	PIN ARDUINO
1	ON-, BUSY-	GND
2	ERROR+	12
3	S1	RST
6	BUSY+	11
7	S2	10
8	ON+	13
9	ERROR-, S1, S2	GND

Tabla 2. Conexión de los leds/interruptores.

FUNCION	PIN ARDUINO
Encoder M1 A	18
Encoder M1 B	19
Encoder M2 A	20
Encoder M2 B	21
IN 1 (M1)	2
IN 2 (M1)	3
Enable A (M1)	4
IN 3 (M2)	5
IN 4 (M2)	6
Enable B (M2)	7

Tabla 3. Conexión de los encoders y el módulo L298N.

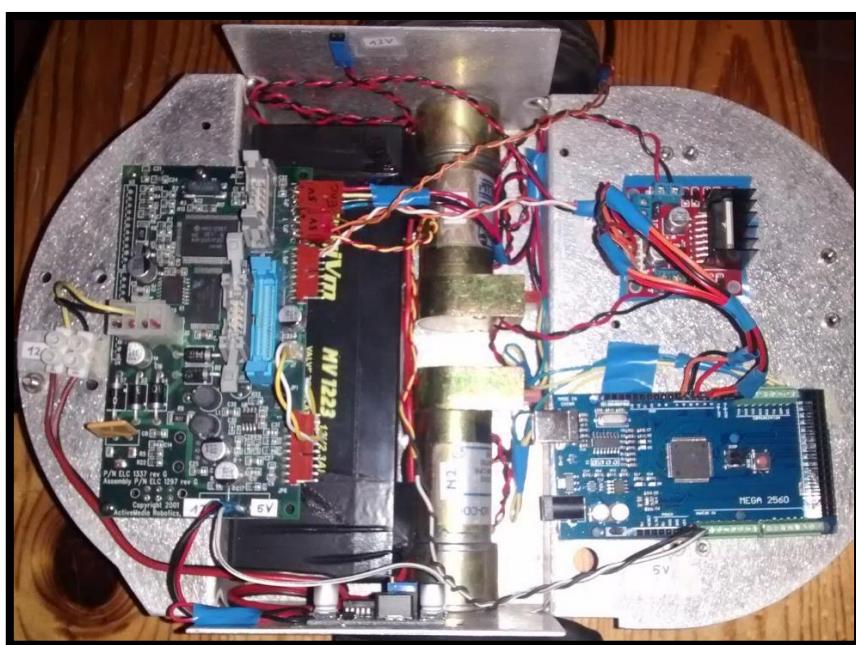


Figura 15. Montaje final.

## 6.4 Arduino con ROS

# **Capítulo 7. Sistema de navegación autónoma**

Los capítulos intermedios servirán para cubrir los siguientes aspectos: antecedentes, problemática o estado del arte, objetivos, fases y desarrollo del proyecto.

En el capítulo 1 se describió bla, bla, bla.....

## **7.1 Funcionamiento**

## **7.2 Preparación**

# **Capítulo 8.**

# **Conclusiones y líneas futuras**

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir unas conclusiones y unas líneas de trabajo futuro.

# **Capítulo 9.**

# **Summary and Conclusions**

This chapter is compulsory. The memory should include an extended summary and conclusions in English.

## **9.1 First Section**

# Capítulo 10.

# Presupuesto

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir un presupuesto.

## 10.1 Sección Uno

Tipos	Descripción
AAA	BBB
CCC	DDD
EEE	FFF
GGG	HHH

Tipos	Descripción
AAA	BBB
CCC	DDD
EEE	FFF
GGG	HHH

Tabla 4. Tabla resumen de los Tipos.

# Apéndice A.

# Códigos del Arduino

## A.1. Sketch principal

```
*****  
***  
*  
* SketchAmigoBot.ino  
*  
*****  
***  
*  
* AUTORES  
* Andrés Cidoncha Carballo  
*  
* FECHA  
* 25/06/2016  
*  
* DESCRIPCION  
* Sketch principal para el Arduino MEGA. Encargado de publicar y suscribirse a  
los topics  
* Basado en el proyecto ros_arduino_diff_drive  
https://github.com/SimonBirrell/ros_arduino_diff_drive  
*****  
****/  
  
*****LIBRARIES*****  
/* Include the appropriate ROS headers */  
#include <ros.h>  
#include <geometry_msgs/Twist.h>  
#include <geometry_msgs/Quaternion.h>  
#include <nav_msgs/Odometry.h>  
#include <tf/transform_broadcaster.h>  
#include <math.h> /* We need sin and cos for odometry calcuations */  
#include "pid_params.h" /* Load the custom typedefs for tracking encoder info */  
#include "AmigoBot.h" /* AmigoBot Robot Class */  
*****  
  
*****CONSTANTS*****  
#define MAXOUTPUT 255 // PWM  
#define PID_UPDATE_RATE 30 // Hz  
#define ODOM_PUBLISH_RATE 10 // Hz  
*****  
  
void cmdVelCb(const geometry_msgs::Twist& msg);  
  
*****OBJECTS*****  
/* Structs defined in PIDTypes.h */  
SetPointInfo leftPID, rightPID;  
OdomInfo odomInfo;
```

```

ros::NodeHandle nh; /* Create the ROS node handle */

/* Odometry publisher */
nav_msgs::Odometry odom_msg;
ros::Publisher odomPub("/odom", &odom_msg);

/* A subscriber for the /cmd_vel topic */
ros::Subscriber<geometry_msgs::Twist> cmdVelSub("/cmd_vel", &cmdVelCb);

/* The Robot Object */
AmigoBot* robot;
//*********************************************************************VARIABLES*****/
const float PID_INTERVAL = 1000.0 / PID_UPDATE_RATE;
const float ODOM_INTERVAL = 1000.0 / ODOM_PUBLISH_RATE;
char baseFrame[] = "/base_link";
char odomFrame[] = "/odom";
unsigned long nextPID = 0;
unsigned long nextOdom = 0;
unsigned char moving = 0; // is the base in motion?
//*********************************************************************METHODS*****/
/* Convert meters per second to ticks per time frame */
int SpeedToTicks(float v) {
    return int(v * cpr / (PID_UPDATE_RATE * PI * wheelDiameter));
}

/* The callback function to convert Twist messages into motor speeds */
void cmdVelCb(const geometry_msgs::Twist& msg) {
    float x = msg.linear.x; // m/s
    float th = msg.angular.z; // rad/s
    float spd_left, spd_right;

    robot->busy();

    lastMotorCommand = millis(); /* Reset the auto stop timer */

    if (x == 0 && th == 0) {
        moving = 0;
        robot->stopMotors();
        return;
    }

    moving = 1; /* Indicate that we are moving */

    if (x == 0) { // Turn in place
        spd_right = th * wheelTrack / 2.0;
        spd_left = -spd_right;
    }
    else if (th == 0) { // Pure forward/backward motion
        spd_left = spd_right = x;
    }
    else { // Rotation about a point in space
        spd_left = x - th * wheelTrack / 2.0;
        spd_right = x + th * wheelTrack / 2.0;
    }

    /* Set the target speeds in meters per second */
    leftPID.TargetSpeed = spd_left;
    rightPID.TargetSpeed = spd_right;
}

```

```

/* Convert speeds to encoder ticks per frame */
leftPID.TargetTicksPerFrame = SpeedToTicks(leftPID.TargetSpeed);
rightPID.TargetTicksPerFrame = SpeedToTicks(rightPID.TargetSpeed);

robot->ready();
}

/* PID routine to compute the next motor commands */
void doPID(SetPointInfo * p) {
    long Perror;
    long output;

    Perror = p->TargetTicksPerFrame - (p->Encoder - p->PrevEnc);

    // Derivative error is the delta Perror
    output = (Kp * Perror + Kd * (Perror - p->PrevErr) + Ki * p->Ierror) / Ko;
    p->PrevErr = Perror;
    p->PrevEnc = p->Encoder;

    output += p->output;

    if (output >= MAXOUTPUT)
        output = MAXOUTPUT;
    else if (output <= -MAXOUTPUT)
        output = -MAXOUTPUT;
    else
        p->Ierror += Perror;

    p->output = output;
}

/* Read the encoder values and call the PID routine */
void updatePID() {
    leftPID.Encoder = robot->leftMotor->read();
    rightPID.Encoder = robot->rightMotor->read();

    /* Record the time that the readings were taken */
    odomInfo.encoderTime = millis();
    odomInfo.encoderStamp = nh.now();

    if (!moving) /* If we're not moving there is nothing more to do */
        return;

    /* Compute PID update for each motor */
    doPID(&leftPID);
    doPID(&rightPID);

    robot->setSpeeds(leftPID.output, rightPID.output); /* Set the motor speeds */
}

/* Calculate the odometry update and publish the result */
void updateOdom() {
    double dt, dleft, dright, dx, dy, dxy_ave, dth, vxy, vth;
    static tf::TransformBroadcaster odom_tf;

    /* Get the time in seconds since the last encoder measurement */
    dt = (odomInfo.encoderTime - odomInfo.lastEncoderTime) / 1000.0;

    /* Save the encoder time for the next calculation */
    odomInfo.lastEncoderTime = odomInfo.encoderTime;
}

```

```

/* Calculate the distance in meters traveled by the two wheels */
dleft = (leftPID.Encoder - odomInfo.prevLeftEnc) / ticksPerMeter;
dright = (rightPID.Encoder - odomInfo.prevRightEnc) / ticksPerMeter;

odomInfo.prevLeftEnc = leftPID.Encoder;
odomInfo.prevRightEnc = rightPID.Encoder;

/* Compute the average linear distance over the two wheels */
dxy_ave = (dleft + dright) / 2.0;

/* Compute the angle rotated */
dth = (dright - dleft) / wheelTrack;

/* Linear velocity */
vxy = dxy_ave / dt;

/* Angular velocity */
vth = dth / dt;

/* How far did we move forward? */
if (dxy_ave != 0) {
    dx = cos(dth) * dxy_ave;
    dy = -sin(dth) * dxy_ave;
    /* The total distance traveled so far */
    odomInfo.linearX += (cos(odomInfo.angularZ) * dx - sin(
        odomInfo.angularZ) * dy);
    odomInfo.linearY += (sin(odomInfo.angularZ) * dx + cos(
        odomInfo.angularZ) * dy);
}

/* The total angular rotated so far */
if (dth != 0)
    odomInfo.angularZ += dth;

/* Represent the rotation as a quaternion */
geometry_msgs::Quaternion quaternion;
quaternion.x = 0.0;
quaternion.y = 0.0;
quaternion.z = sin(odomInfo.angularZ / 2.0);
quaternion.w = cos(odomInfo.angularZ / 2.0);

/* Publish the distances and speeds on the odom topic. Set the timestamp
   to the last encoder time. */
odom_msg.header.frame_id = odomFrame;
odom_msg.child_frame_id = baseFrame;
odom_msg.header.stamp = odomInfo.encoderStamp;
odom_msg.pose.pose.position.x = odomInfo.linearX;
odom_msg.pose.pose.position.y = odomInfo.linearY;
odom_msg.pose.pose.position.z = 0;
odom_msg.pose.pose.orientation = quaternion;
odom_msg.twist.twist.linear.x = vxy;
odom_msg.twist.twist.linear.y = 0;
odom_msg.twist.twist.linear.z = 0;
odom_msg.twist.twist.angular.x = 0;
odom_msg.twist.twist.angular.y = 0;
odom_msg.twist.twist.angular.z = vth;
odomPub.publish(&odom_msg);

geometry_msgs::TransformStamped transform;
transform.header = odom_msg.header;
transform.child_frame_id = odom_msg.child_frame_id;
transform.transform.translation.x = odom_msg.pose.pose.position.x;

```

```

transform.transform.translation.y = odom_msg.pose.pose.position.y;
transform.transform.translation.z = odom_msg.pose.pose.position.z;
transform.transform.rotation = odom_msg.pose.pose.orientation;
odom_tf.sendTransform(transform);
}

void clearPID() {
    moving = 0;
    leftPID.PrevErr = 0;
    leftPID.Ierror = 0;
    leftPID.output = 0;
    rightPID.PrevErr = 0;
    rightPID.Ierror = 0;
    rightPID.output = 0;
}

void clearAll() {
    clearPID();
    robot->resetEncoders();
}
//*********************************************************************SETUP*****
void setup() {
    Serial.begin(57600);
    robot = new AmigoBot();
    /* Set the target speeds in meters per second */
    leftPID.TargetSpeed = 0.0;
    rightPID.TargetSpeed = 0.0;

    /* Convert speeds to encoder ticks per frame */
    leftPID.TargetTicksPerFrame = SpeedToTicks(leftPID.TargetSpeed);
    rightPID.TargetTicksPerFrame = SpeedToTicks(rightPID.TargetSpeed);

    /* Zero out the encoder counts */
    robot->resetEncoders();

    /* Initialize the ROS node */
    nh.initNode();

    nextPID = PID_INTERVAL;
    nextOdom = ODOM_INTERVAL;

    /* Advertise the Odometry publisher */
    nh.advertise(odomPub);

    /* Activate the Twist subscriber */
    nh.subscribe(cmdVelSub);

    robot->ready();
}
//*********************************************************************LOOP*****
void loop() {
    /* Is it time for another PID calculation? */
    if (millis() > nextPID) {
        updatePID();
        nextPID += PID_INTERVAL;
    }

    /* Is it time for another odometry calculation? */
}

```

```

if (millis() > nextOdom) {
    updateOdom();
    nextOdom += ODOM_INTERVAL;
}

/* AUTOSTOP? */
if ((millis() - lastMotorCommand) > AUTO_STOP_INTERVAL) {};
    robot->setSpeeds(0, 0);
    moving = 0;
}

nh.spinOnce();

}

```

## A.2. Clase AmigoBot

```

/****************************************************************************
*
* AmigoBot.h
*
*****
*
* AUTORES
* Andrés Cidoncha Carballo
*
* FECHA
* 25/06/2016
*
* DESCRIPCION
* Clase para representar la electrónica del AmigoBot de cara a simplificar la
* interaccion en el sketch principal
* LICENSED UNDER GNU v3
*****
/
#include "MotorEncoder.h"

/* PINES */
#define ENC1_P1    18
#define ENC1_P2    19
#define ENC2_P1    20
#define ENC2_P2    21
#define M1_IN1     2
#define M1_IN2     3
#define M1_ENA     4
#define M2_IN1     5
#define M2_IN2     6
#define M2_ENA     7
#define SW2o       10
#define BUSY_LED   11
#define ERROR_LED 12
#define ON_LED     13

class AmigoBot{
public:
    MotorEncoder* leftMotor = new MotorEncoder(M1_IN1, M1_IN2, M2_ENA, ENC1_P1,
ENC1_P2); //M1
    MotorEncoder* rightMotor = new MotorEncoder(M2_IN1, M2_IN2, M2_ENA, ENC2_P1,
ENC2_P2); //M2
    AmigoBot() {

```

```

pinMode (SW2o, INPUT);
pinMode (ON_LED, OUTPUT);
pinMode (BUSY_LED, OUTPUT);
pinMode (ERROR_LED, OUTPUT);
digitalWrite (ON_LED, LOW);
digitalWrite (BUSY_LED, HIGH);
digitalWrite (ERROR_LED, LOW);
}
/* LED */
void ready();
void busy();
void error();
/* MOTORS */
void resetEncoders();
void setSpeeds(uint8_t leftSpeed, uint8_t rightSpeed);
void stopMotors();
};

void AmigoBot::ready(void) {
    digitalWrite (BUSY_LED, HIGH);
    digitalWrite (ON_LED, HIGH);
}

void AmigoBot::busy(void) {
    digitalWrite (ON_LED, HIGH);
    digitalWrite (BUSY_LED, LOW);
}

void AmigoBot::error(void) {
    digitalWrite (ON_LED, LOW);
    digitalWrite (BUSY_LED, LOW);
    digitalWrite (ERROR_LED, HIGH);
}

void AmigoBot::resetEncoders(void) {
    leftMotor->reset();
    rightMotor->reset();
}

void AmigoBot::setSpeeds(uint8_t leftSpeed, uint8_t rightSpeed) {
    leftMotor->setSpeed(leftSpeed);
    rightMotor->setSpeed(rightSpeed);
}

void AmigoBot::stopMotors(void) {
    leftMotor->setSpeed(0);
    rightMotor->setSpeed(0);
}

```

### A.3. Clase MotorEncoder

```
*****
*
* MotorEncoder.h
*
*****
*
* AUTORES
* Andrés Cidoncha Carballo
*
* FECHA
* 25/06/2016
*
* DESCRIPCION
* Clase para representar el conjunto motor+encoder. Utiliza la libreria Encoder
* http://www.pjrc.com/teensy/td_libs_Encoder.html
* LICENSED UNDER GNU v3
*****
#include <Encoder.h>

class MotorEncoder{
    private:
        uint8_t IN1, IN2, ENABLE;
        Encoder* encoder;
    public:
        MotorEncoder(uint8_t IN1PIN, uint8_t IN2PIN, uint8_t ENABLEPIN,
                    uint8_t ENCPIN1, uint8_t ENCPIN2){
            encoder = new Encoder(ENCPIN1, ENCPIN2);
            IN1 = IN1PIN;
            IN2 = IN2PIN;
            ENABLE = ENABLEPIN;
            pinMode (ENABLE, OUTPUT);
            pinMode (IN1, OUTPUT);
            pinMode (IN2, OUTPUT);
            digitalWrite (IN1, HIGH);
            digitalWrite (IN2, LOW);
            analogWrite (ENABLE, 0);
        }
        int32_t read();
        void write(int32_t position);
        void reset();
        void setSpeed(uint8_t speed);
        void reverse();
    };

    int32_t MotorEncoder::read(void) {
        return encoder->read();
    }

    void MotorEncoder::write(int32_t position) {
        encoder->write(position);
    }

    void MotorEncoder::reset(void) {
        encoder->write(0);
    }

    void MotorEncoder::setSpeed(uint8_t speed) {
        analogWrite(ENABLE, speed);
    }
}
```

```
void MotorEncoder::reverse(void) {
    digitalWrite (IN1, HIGH-digitalRead(IN1));
    digitalWrite (IN2, HIGH-digitalRead(IN2));
}
```

# **Apéndice B.**

## **Título del Apéndice 2**

### **B.1.      Otro apendice: Seccion 1**

Texto

### **B.2.      Otro apendice: Seccion 2**

Texto

# Bibliografía

- [1] ACM LaTeX Style. [http://www.acm.org/publications/latex\\_style/](http://www.acm.org/publications/latex_style/).
- [2] FACOM OS IV SSL II USER'S GUIDE, 99SP0050E5. Technical report, 1990.
- [3] D. H. Bailey and P. Swarztrauber. The fractional Fourier transform and applications. *SIAM Rev.*, 33(3):389–404, 1991.
- [4] A. Bayliss, C. I. Goldstein, and E. Turkel. An iterative method for the Helmholtz equation. *J. Comp. Phys.*, 49:443–457, 1983.
- [5] C. Darwin. *The Origin Of Species*. November 1859.
- [6] C. Goldstein. Multigrid methods for elliptic problems in unbounded domains. SIAM, *J. Numer. Anal.*, 30:159–183, 1993.
- [7] P. Swarztrauber. *Vectorizing the FFTs*. Academic Press, New York, 1982.
- [8] S. Taásan. *Multigrid Methods for Highly Oscillatory Problems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1984.
- [9] Berndtsson, M., Hansson, J., Olsson, B., Lundell, B. (2008), *A Guide for Students in Computer Science and Information Systems*, Springer.