



# UNIVERSIDAD POLITÉCNICA DE JUVENTINO ROSAS

## TESIS

**“Implementación de algoritmos en un sistema embebido en base  
a ROS para la manipulación de dispositivos autónomos”**

PARA OBTENER EL TÍTULO DE  
**INGENIERO EN REDES Y TELECOMUNICACIONES**

*Presenta:*  
**TÉLLEZ CASTELLANOS ARANZAZÚ MAHELET**

*Asesores:*  
Luis Rey Lara González  
Juan Israel Yañez Vargas

Juventino Rosas, Gto. 03 de Septiembre de 2020.

# Agradecimientos

Después de todo este periodo de trabajo y aprendizaje me he dado cuenta de la gran importancia que tiene este apartado, ya que gracias a todas las personas que han estado a mi lado ha sido posible realizar con éxito mi tesis.

En primer lugar, me agradezco por seguir adelante, por ser valiente esas veces que quise salir corriendo, por seguir intentando sin rendirme, por soñar y amar a pesar de las circunstancias, me agradezco, me valoro y me felicito.

Me gustaría dar las gracias a mi tutor principal por haberme dado la oportunidad de realizar este proyecto. A mi tutor Luis Rey Lara González por nunca dudar de mí y siempre incitarme a ir por más y no darme por vencida, por estar ahí cuando estaba por rendirme y por siempre tener una palabra de aliento para hacerme seguir adelante.

Quiero agradecer a mi familia el apoyo y amor incondicional durante todos estos años. Muy en especial a mis padres, sin los cuales nada de esto hubiera sido posible, a los cuales profeso mi más sincero amor y profunda devoción. A mi padre, Javier, del cual siempre me he sentido orgullosa de ser su “Chata”, por representar siempre esa honradez, vocación por su familia y profesión, esa persona que me ha enseñado a no darme por vencida, a no dejar nada a medias e incontables cualidades más en las que siempre he tratado de fijarme para crecer como persona. A mi madre, Susana, por transmitirme esos valores de lucha, de no rendirme nunca, de pelear hasta el último aliento para conseguir mis sueños y metas, por su inagotable paciencia en mis momentos de frustración.

No podría olvidarme de mis hermanos y mi sobrino, Vanessa, Diego y Santino. Pese a que no siempre fuimos muy unidos, el tiempo nos dio la razón y hoy día son de lejos mi mayor alegría. Esas personas a las que siempre he podido recurrir cuando más necesitaba una sonrisa y con quienes no me canso de compartir anécdotas y aventuras.

A mi mejor amigo, Diego, que me ha acompañado a lo largo ya no solo de la carrera, si no de tantos años de sana y bonita amistad, incontables aventuras, apoyo en malos mo-

mentos, risas y siempre ha intentado que abriera la mente para ver todo aquello que no veía y que no creía que fuera posible de lograr por mí misma, por eso le considero de mi familia.

A mi gran compañero de trabajos, de desvelos y de frustraciones, Alejandro, quien siempre me apoya en todos los momentos difíciles del proceso de realización y me ayuda cuando más lo necesite con palabras de aliento, escuchando mis quejas o ayudando con mis problemas de programación.

A mi mayor amuleto de la buena suerte, mi compañero fiel de desvelos, quien fue para mí más que una mascota. Solo me bastaba verte dormir en mi cama para no sentirme sola y continuar trabajando. Fue mi bebé que me enseñó a luchar y valorar la vida, que a pesar de las circunstancias siempre me lleno de felicidad, donde quiera que estés te agradezco cada día que pasamos juntos y todo el amor que me diste, gracias por tanto Duke.

Por último y no menos importante, a mi persona favorita, Alexis, por constituirse como esa pieza insustituible y de valor incalculable en mi vida. Por haberme hecho crecer como persona desde el primer momento que nos conocimos, por constituir ese apoyo incondicional, ese hombro sobre el que llorar, por transmitir tanta alegría en tantos momentos y por contagiarde ese positivismo sin el cual no sería posible avanzar. Por ayudarme a comerme el mundo y a siempre mantener los pies firmes sobre la tierra pero sin olvidar que no importa que tan imposible suenen las cosas siempre podrán cumplirse si así lo deseó. En definitiva, por ser la persona a la que más quiero y que me hace soñar con la luna.

# Resumen

Nuevas tecnologías surgen día con día, las actuales generaciones hemos sido testigos de inventos, herramientas y servicios que han revolucionado y mejorado la calidad de vida de como la conocíamos hace algunos años, desde casas de alto presupuesto totalmente automatizadas, hasta dispositivos autónomos que cumplen tareas simples como recreación o inclusive que llevan acabo acciones de alto riesgo.

Este proyecto trata principalmente de aumentar los campos de la robótica dentro de la automatización de los dispositivos ya existentes. Estas aplicaciones, que se intentan conseguir con la automatización, únicamente son métodos de ayudar a las personas quitándoles la carga que supone realizar esos trabajos, por eso se ha decidido realizar este proyecto.

El presente reporte de tesis es el producto de un trabajo de grado para la obtención del título de Ingeniero en Redes y Telecomunicaciones. Presenta el desarrollo e implementación de algoritmos en un sistema embebido para la manipulación de dispositivos autónomos.

El documento describe un proceso de diseño e instalación de software mediante la plataforma ROS. Con ayuda de la plataforma de propósito general Arduino UNO se logra tener una comunicación constante con el hardware de cada uno de los dispositivos. Para, finalmente, obtener un sistema de control de instrucciones de un dispositivo autónomo de bajo costo.

# Abstract

New technologies emerge every day, current generations have witnessed inventions, tools and services that have revolutionized and improved the quality of life as we knew it some years ago, from fully automated high-budget houses, to autonomous devices that fulfill tasks simple as recreation or even that carry out high risk actions.

This project mainly tries to increase the fields of robotics within the automation of existing devices. These applications, which are tried to be achieved with automation, are only methods of helping people by taking away the burden of carrying out those jobs, that is why it has been decided to carry out this protection.

This thesis report is the product of a degree project to obtain the title of Network and Telecommunications Engineer. It presents the development and implementation of algorithms in an embedded system for the manipulation of autonomous devices.

The document describes a software design and installation process using the ROS platform. With the help of the general purpose platform Arduino UNO it is possible to have constant communication with the hardware of each of the devices. To finally get a low-cost standalone device instruction control system.

# Índice general

<b>Índice de figuras</b>	<b>7</b>
<b>1. Introducción</b>	<b>9</b>
1.1. Objetivos . . . . .	10
1.1.1. Objetivo general . . . . .	10
1.1.2. Objetivos específicos . . . . .	10
1.2. Justificación . . . . .	10
1.3. Estado del arte . . . . .	11
1.3.1. Sistema de posicionamiento 3-D Autónomo de Instrumentos Quirúrgicos en Cirugía Laparoscópica . . . . .	11
1.3.2. Integración de ROS con Arduino y Raspberry Pi . . . . .	11
1.3.3. Diseño de un robot móvil para estudio de la ocupación en habitaciones	12
1.3.4. Sistema de supervisión y telemetría para un robot móvil con pila de combustible . . . . .	13
<b>2. Marco Teórico</b>	<b>15</b>
2.1. Sistemas embebidos . . . . .	15
2.1.1. Como están constituidos . . . . .	16
2.2. Arduino . . . . .	17
2.2.1. Arduino UNO . . . . .	17
2.3. Módulo Bluetooth . . . . .	19
2.4. Firmware . . . . .	22
2.5. Open Source Software . . . . .	22
2.6. Open Source Hardware . . . . .	24
2.7. Introducción a ROS . . . . .	27
2.7.1. ¿Qué es ROS y por qué debe utilizarse? . . . . .	27
2.7.2. Sistemas operativos compatibles con ROS . . . . .	28
2.7.3. Distribuciones de ROS . . . . .	31
2.7.4. Software . . . . .	35

<b>3. Implementación</b>	<b>36</b>
3.1. Instalación de ROS en Ubuntu . . . . .	36
3.1.1. Requisitos de instalación . . . . .	36
3.2. Instalación de ROS . . . . .	37
3.3. Instalación de la librería ros_lib en el entorno de Arduino . . . . .	37
<b>4. Pruebas</b>	<b>38</b>
4.1. Comunicación Arduino y ROS . . . . .	38
4.1.1. Ejemplo de servocontrolador . . . . .	38
4.1.2. Ejemplo de blink . . . . .	41
4.2. Control de dispositivo autónomo . . . . .	44
<b>5. Resultados y conclusiones</b>	<b>45</b>
5.1. Trabajos a futuro . . . . .	46
<b>6. Glosario</b>	<b>47</b>
<b>A. Instalación de ROS</b>	<b>53</b>
A.0.1. Prerrequisitos . . . . .	53
A.0.2. Creación de Catkin Workspace . . . . .	54
A.0.3. Compilación de Catkin Workspace . . . . .	55
<b>B. Instalación de la librería ros_lib en el entorno de Arduino</b>	<b>56</b>
B.0.1. Actualización de Debian package index . . . . .	56
B.0.2. Compatibilidad ROS y Arduino . . . . .	56
B.0.3. Instalación de drivers en Catkin Workspace . . . . .	56
B.0.4. Compilación de drivers . . . . .	57
B.0.5. Reinicio del IDE Arduino . . . . .	57
<b>C. Ejemplo de servocontrolador</b>	<b>58</b>
<b>D. Ejemplo de blink</b>	<b>60</b>
<b>E. Control de dispositivo autónomo</b>	<b>61</b>
<b>F. Comunicación Bluetooth</b>	<b>65</b>
F.0.1. Algoritmo Master . . . . .	65
F.0.2. Algoritmo Slave . . . . .	66
F.0.3. Algoritmo Control . . . . .	68
F.0.4. Algoritmo Orden . . . . .	69

# Índice de figuras

1.1.	Integración de ROS con Arduino y Raspberry Pi . . . . .	12
1.2.	Diseño de un Robot Móvil para estudio de la ocupación en habitaciones . . . . .	13
1.3.	Sistema de supervisión y telemetría para un robot móvil . . . . .	14
2.1.	Sistemas embebidos y sus aplicaciones . . . . .	16
2.2.	Componentes de un sistema embebido (nivel lógico) . . . . .	16
2.3.	Logotipo de la empresa Arduino . . . . .	17
2.4.	Sistema embebido Arduino UNO . . . . .	18
2.5.	Master - Slaves . . . . .	20
2.6.	Módulos Bluetooth . . . . .	20
2.7.	Módulos Bluetooth HC-05 . . . . .	21
2.8.	Módulos Bluetooth HC-06 . . . . .	21
2.9.	Firmware . . . . .	22
2.10.	Ejemplos de Open Source . . . . .	24
2.11.	Logo de Open Source Hardware . . . . .	26
2.12.	Logo ROS . . . . .	28
2.13.	Logo Ubuntu . . . . .	29
2.14.	Logo Debian . . . . .	29
2.15.	Logo HomeBrew . . . . .	30
2.16.	Logo Gentoo . . . . .	30
2.17.	Logo OpenEmbedded . . . . .	31
2.18.	Logo Android NDK . . . . .	31
2.19.	Logo ROS Noetic . . . . .	32
2.20.	Logo ROS Melodic . . . . .	33
2.21.	Logo ROS Lunar . . . . .	33
2.22.	Logo ROS Kinetic . . . . .	34
2.23.	Logo ROS Jade . . . . .	34
2.24.	Logo ROS Indigo . . . . .	35
3.1.	Logo Ubuntu 16.04 . . . . .	36

3.2. Comunicación entre ROS y Arduino UNO . . . . .	37
4.1. Conexión de Servo . . . . .	39
4.2. Inicio de librerías de ROS . . . . .	40
4.3. Configuración de puerto serial . . . . .	41
4.4. Comunicación entre Arduino, ROS y Servo . . . . .	41
4.5. Ubicación del led en la placa Arduino UNO . . . . .	42
4.6. Inicio de librerías de ROS . . . . .	43
4.7. Configuración de puerto serial . . . . .	43
4.8. Comunicación de ROS, Arduino y Led puerto 13 . . . . .	44

# Capítulo 1

## Introducción

De entre los últimos avances que se están realizando actualmente en el campo de la robótica (Véase la página 48), uno de los más llamativos y que poco a poco se está implantando más en la vida cotidiana es el de los vehículos autónomos (Véase la página 47). Los progresos tecnológicos en materia de procesamiento y reconocimiento de imágenes, además de en la adquisición de datos del vehículo, están logrando que se avance poco a poco en este campo, situación que está propiciada por la constante necesidad de acceso a lugares con difícil acceso para los humanos o que se prefiere evitar la fatiga de realizar actividades.

En el presente proyecto se aborda la tarea de la integración de ROS (Robot Operating System) con la plataforma de propósito general Arduino UNO.

ROS se constituye como un **middleware** [2] (Véase la página 48) ya que se trata de un software que asiste a las aplicaciones en concreto diseñadas en este proyecto así como a cualquier aplicación de propósito general diseñada sobre él, para interactuar o comunicarse entre sí y el hardware utilizado. Funciona como una capa de abstracción de software distribuida (Véase la página 48) situada entre las capas de aplicaciones y las capas inferiores. El **middleware** abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos (Véase la página 49) y lenguajes de programación.

ROS es una colección de frameworks (Véase la página 47) para el desarrollo de software (Véase la página 49) de robots. A pesar de no ser un sistema operativo, ROS provee los servicios estándar de uno de estos tales como la abstracción del hardware (Véase la página 48), el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que

pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. [5]

La misión principal que se pretende conseguir con la integración es la de, mediante un sistema distribuido, realizar la programación y control de un robot móvil a través del diseño de aplicaciones que doten a dicho sistema de diferentes características y funcionalidades.

Hasta ahora el acceso o la contribución en todo este tipo de tecnología estaba al alcance de muy pocos, pero con la aparición y creciente desarrollo de entornos de programación asociados a la robótica como ROS, basados en el código abierto, estas barreras han desaparecido, permitiendo que se creen grandes comunidades de desarrolladores, que en definitiva contribuyen en una gran medida al ya vertiginoso avance de la robótica.

## 1.1. Objetivos

### 1.1.1. Objetivo general

Implementar algoritmos en un sistema embebido en base a ROS para la manipulación de dispositivos autónomos.

### 1.1.2. Objetivos específicos

- Desarrollar algoritmos basados en ROS, que permitan establecer una ruta de seguimiento para el control de un dispositivo autónomo.
- Implementar algoritmos en un sistema embebido para el control del dispositivo.
- Mostrar el funcionamiento de los algoritmos hechos en ROS en un dispositivo autónomo con una ruta predeterminada.

## 1.2. Justificación

En los últimos años ha habido un gran avance de la tecnología robótica dentro de la automatización. Cada vez son más las empresas que apuestan por desarrollar robots móviles automatizados para realizar tareas que realizan las personas de forma repetitiva, tareas que suponen un riesgo físico para las personas o cualquier otra función que beneficie

o haga más sencilla la vida de la gente.

Un campo muy necesario para el diseño de robots autónomos es el estudio del entorno para la toma de decisiones por parte del robot.

Debido a esta demanda, en el estudio de la semántica del entorno para que el robot realice una navegación más eficiente, se ha desarrollado este proyecto en el que se espera aportar más información como puede ser su localización a través del GPS, distancia recorrida o imágenes recolectadas por medio de video.

## 1.3. Estado del arte

En este capítulo se revisan exhaustivamente los métodos y las tecnologías relacionadas con este proyecto.

A continuación, se presentará un breve resumen de los antecedentes en los usos de dispositivos autónomos con ROS ya implementados.

### 1.3.1. Sistema de posicionamiento 3-D Autónomo de Instrumentos Quirúrgicos en Cirugía Laparoscópica

En (Plaza, P.) [1] se analiza un sistema de visión robótica que automáticamente recupera y posiciona instrumentos quirúrgicos durante operaciones laparoscópicas robotizadas. El instrumento se monta al final del efecto de un robot quirúrgico que está controlado a través de Visual Servoing. El objetivo de la tarea automatizada es llevar un instrumento de forma segura desde una posición tridimensional desconocida u oculta a la deseada.

Unos diodos emisores de luz se encuentran unidos a la punta del instrumento y un soporte para el instrumento fabricado específicamente y equipado con fibra óptica proyecta un conjunto de puntos láser en la superficie de los órganos. Los marcadores ópticos son detectados en la imagen endoscópica y permiten localizar el instrumento con respecto a la escena. El instrumento es recuperado y centrado en la imagen empleando un algoritmo basado en detectar errores en detalles de las imágenes.

### 1.3.2. Integración de ROS con Arduino y Raspberry Pi

En (Romero, A.) [2] se aborda la tarea de la integración de ROS (Robot Operating System) con las plataformas de propósito general Arduino Mega ADK y Raspberry Pi (Figura 1.1). La misión principal de este proyecto fue conseguir la integración, mediante

un sistema distribuido, la programación y el control de un robot móvil a través del diseño de aplicaciones que doten a dicho sistema de diferentes características funcionalidades.



Figura 1.1: Integración de ROS con Arduino y Raspberry Pi

### 1.3.3. Diseño de un robot móvil para estudio de la ocupación en habitaciones

En (Cabalgente, R.) [3] se presenta la creación de un robot móvil autónomo capaz de monitorizar la detección de personas en cada una de las habitaciones por las que vaya pasando (Figura 1.2), de esta forma se tiene una estimación del número de personas que se localizan en cada una, el tiempo que se tarda en detectar a todas las personas dentro de la habitación y el tiempo en recorrer cada habitación.

El robot consta de un microcontrolador Arduino encargado de realizar el recorrido autónomo por las diferentes estancias del recinto siguiendo la pared que tenga a su izquierda mediante el uso de unos sensores.



Figura 1.2: Diseño de un Robot Móvil para estudio de la ocupación en habitaciones

#### **1.3.4. Sistema de supervisión y telemetría para un robot móvil con pila de combustible**

En (Bergmann, P.) [4] el principal objetivo es el diseño y la instalación de un sistema capaz de medir la intensidad demandada por el robot a las baterías en tiempo real. Esto con ayuda de un sensor de intensidad, conectado a una placa Arduino. Además esta medición, junto con otros parámetros del sistema EMS, instalado en el Summit junto a la pila de combustible, son enviados vía WiFi a una estación para ser monitorizados a tiempo real. Esto trata del diseño de un sistema de telemetría sobre el que corre una capa ROS y el cual envía vía WiFi todos los datos requeridos para ser monitorizados (Figura 1.3). En conclusión, es un sistema que por una parte sea capaz de medir la intensidad demandada a las baterías y por otra parte adquiera todos los parámetros relacionados con la pila, provenientes del EMS. Dicho sistema realiza envíos de todos los datos adquiridos a una estación que monitorizará en tiempo real todos los parámetros necesarios.

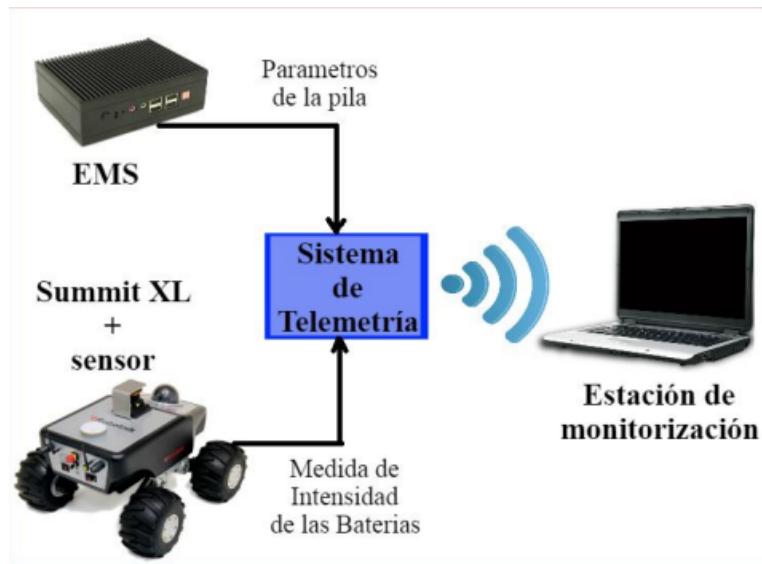


Figura 1.3: Sistema de supervisión y telemetría para un robot móvil

En conclusión, en el desarrollo del estado del arte se encontraron aportes e ideas significativas en diferentes áreas de conocimiento presentes en el proyecto.

# **Capítulo 2**

## **Marco Teórico**

En esta sección se presentan los conceptos claves que se están teniendo en cuenta en el desarrollo del proyecto.

El presente proyecto analiza el diseño e implementación de sistemas embebidos como fuente principal de órdenes de un dispositivo autónomo, dando así la oportunidad de que el usuario sea capaz de dar sus propias instrucciones al dispositivo de manera sencilla pero eficiente. En ese sentido, es preciso aclarar algunos conceptos.

### **2.1. Sistemas embebidos**

Los sistemas embebidos, son módulos o placas creados con el fin de ser controlados por microprocesadores o microcontroladores, llevados a un fin completamente sistematizado y sin llevar tantas tareas, son, mejor dicho, sistemas que cumplen con una tarea en específico, al contrario de lo que ocurre con las computadoras, las cuales tienen un propósito general, ya que están diseñadas para cubrir un amplio rango de necesidades. [18] El número de sistemas embebidos, desplegados en el mundo alcanza los 10,000 millones, una cifra que sigue creciendo día a día. Se estima que para el año 2020 existirán cerca de 40,000 millones de dispositivos embebidos. [5] Esto será equivalente a 5 dispositivos embebidos por cada ser humano que habite en el planeta y el mercado de tecnología embebida (software y hardware) se ubicará alrededor de los 200,000 millones de dólares (Figura 2.2).

Los sistemas embebidos no solo están desplegados en la industria, sino que afectan a cualquier faceta de nuestra vida, ya que gran cantidad de equipos de uso diario integran estos sistemas, como coches, ascensores, juguetes, etc, y estos ejemplos están mostrados en la figura 2.1.



Figura 2.1: Sistemas embebidos y sus aplicaciones

### 2.1.1. Como están constituidos

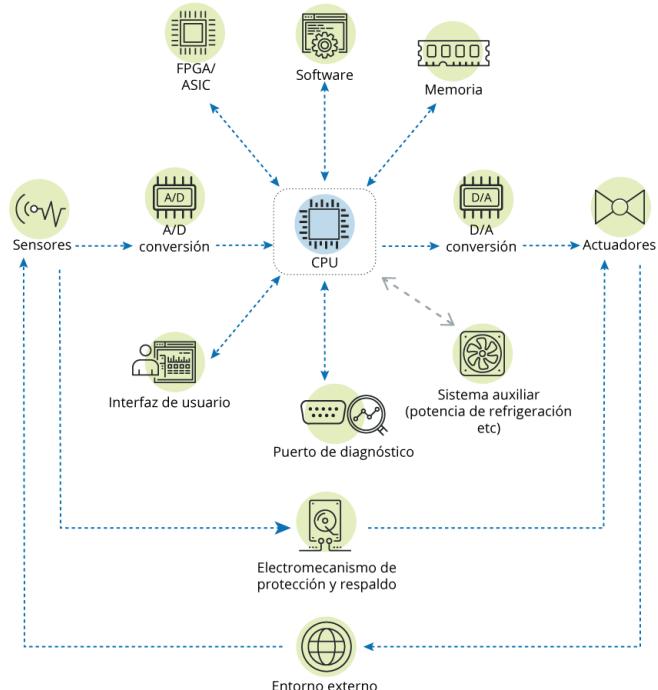


Figura 2.2: Componentes de un sistema embebido (nivel lógico)

## 2.2. Arduino

Es una plataforma de desarrollo de computación física (physical computing) de código abierto, basada en una placa con un sencillo microcontrolador y un entorno de desarrollo para crear software (programas) para la placa. Puedes usar Arduino para crear objetos interactivos, leyendo datos de una gran variedad de interruptores y sensores y controlar multitud de tipos de luces, motores y otros actuadores físicos. Los proyectos con Arduino pueden ser autónomos o comunicarse con un programa (software) que se ejecute en tu ordenador. [14] El proyecto nació en 2003, cuando varios estudiantes del Instituto de Diseño Interactivo de Ivrea, Italia, con el fin de facilitar el acceso y uso de la electrónica y programación. Lo hicieron para que los estudiantes de electrónica tuvieran una alternativa más económica a las populares Basic Stamp, unas placas que podían llegar a valer más de cien dólares, y que no todos se podían permitir. El resultado fue Arduino, una placa con todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador, y que puede ser programada tanto en Windows como MacOS y GNU/Linux. Un proyecto que promueve la filosofía 'learning by doing'.



Figura 2.3: Logotipo de la empresa Arduino

### 2.2.1. Arduino UNO

La arduino UNO es una placa electrónica (Figura 2.4) basada en un microcontrolador Atmega328. Tiene 14 pines de entrada/salida digital (de los cuales 4 pueden ser utilizados para salidas PWM), 6 entradas analógicas, un resonador cerámico de 16 MHz, un conector

para USB tipo hembra, un Jack para fuente de poder, un conector ICSP y botón reset. [32]

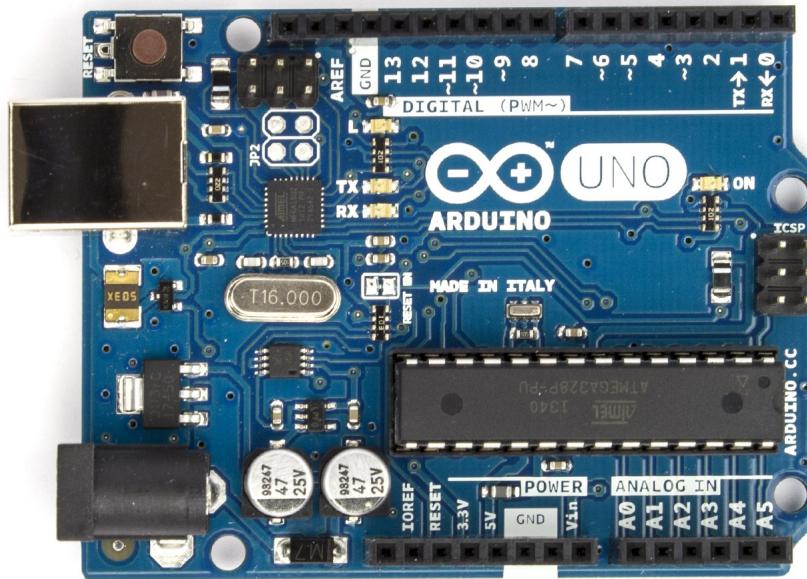


Figura 2.4: Sistema embebido Arduino UNO

Especificaciones :[21]

- Microcontrolador: ATmega328.
- Tensión de Operación (nivel lógico): 5 V.
- Tensión de Entrada (recomendado): 7-12 V.
- Pines E/S Digitales: 14 (de los cuales 6 proveen de salida PWM).
- Entradas Analógicas: 6.
- Memoria Flash: 32 KB (ATmega328) de los cuales 0,5 KB son usados por el bootloader.
- SRAM: 2 KB (ATmega328).
- EEPROM: 1 KB (ATmega328).
- Frecuencia de reloj: 16 MHz.

## 2.3. Módulo Bluetooth

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2.4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles.
- Eliminar los cables y conectores entre estos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Los dispositivos que incorporan este protocolo pueden comunicarse entre sí cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso en habitaciones separadas si la potencia de transmisión es suficiente.

El hardware que compone el dispositivo Bluetooth está compuesto por dos partes:

- Un dispositivo de radio, encargado de modular y transmitir la señal.
- Un controlador digital, compuesto por una CPU (Véase la página 47), un procesador de señales digitales (DSP-Digital Signal Processor) llamado Link Controller (o controlador de Enlace) y de las interfaces con el dispositivo anfitrión.

Los dispositivos Bluetooth pueden actuar como Masters o como Slaves. La diferencia es que un Bluetooth Slave solo puede conectarse a un Master y a nadie más, en cambio un Bluetooth Master puede conectarse a varios Slaves o permitir que ellos se conecten y recibir y solicitar información de todos ellos, arbitrando las transferencias de información (hasta un máximo de 7 Slaves) así como se muestra en la figura 2.5.

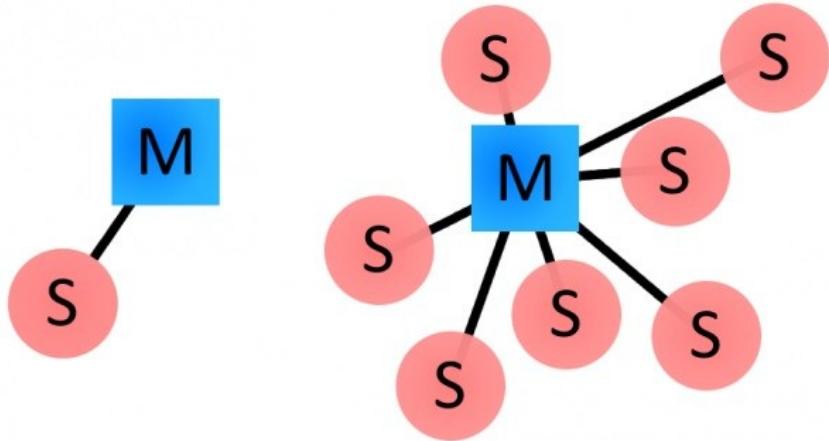


Figura 2.5: Master - Slaves

Los módulos HC-05 y HC-06 (Figura 2.6) son Bluetooth V2 y, considerando sus especificaciones, serán usados para establecer la comunicación entre el dispositivo encargado de enviar las instrucciones a nuestro dispositivo móvil.

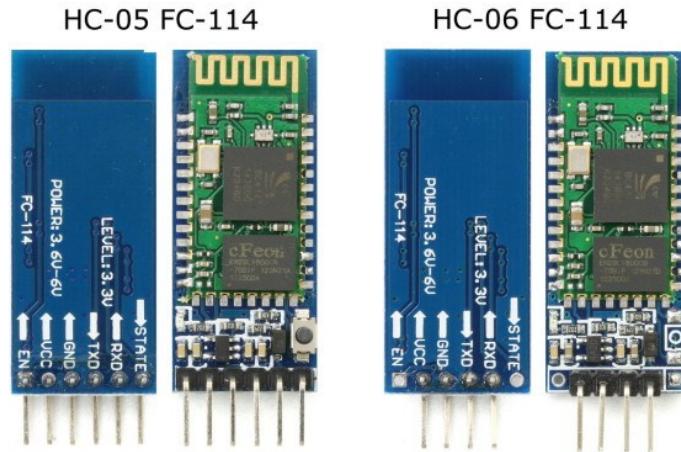


Figura 2.6: Módulos Bluetooth

El módulo Bluetooth HC-05 (Figura 2.7) viene configurado de fábrica como “Esclavo” (Slave), pero se puede cambiar para que trabaje como “Maestro” (Master), además, al igual que el HC-06, se puede cambiar el nombre, código de vinculación, velocidad y otros parámetros más.



Figura 2.7: Módulos Bluetooth HC-05

El módulo Bluetooth HC-06 (Figura 2.8) viene configurado de fábrica como “Esclavo” (Slave) y no puede ser cambiado a “Maestro” como si lo permite el módulo HC-05.



Figura 2.8: Módulos Bluetooth HC-06

## 2.4. Firmware

Cuando se habla de **firmware** se refiere al conjunto de instrucciones de un programa informático que se encuentra registrado en una memoria ROM, flash o similar. Estas instrucciones fijas una lógica primaria que ejerce el control de los circuitos de alguna clase de artefacto.

El firmware forma parte del hardware, ya que se encuentra integrado a la electrónica, pero también es considerado parte del software, al estar desarrollado bajo un lenguaje de programación. Se podría decir que funciona como el nexo entre las instrucciones que llegan al dispositivo desde el exterior y sus diversas partes electrónicas. [17]

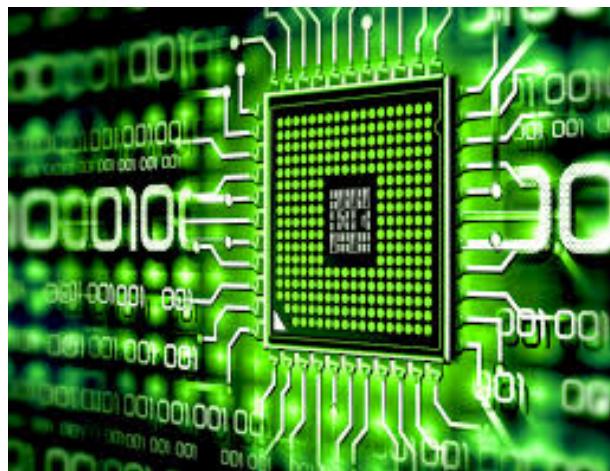


Figura 2.9: Firmware

## 2.5. Open Source Software

Un software **Open Source**, o de código abierto en español, es un término que se utiliza para denominar a cierto tipo de software que se distribuye mediante una licencia que le permite al usuario final, si tiene los conocimientos necesarios, utilizar el código fuente del programa para estudiarlo, modificarlo y realizar mejoras en el mismo, pudiendo incluso hasta redistribuirlo. [16]

Las condiciones de distribución de un programa open source deben cumplir con el siguiente criterio:

- **Libre distribución**

La licencia no debe restringir a nadie vender o entregar el software como un componente de una distribución de software que contenga programas de distintas fuentes. La licencia no debe requerir ningún tipo de cuota por su venta.

- **Código fuente**

El programa debe incluir el código fuente, y se debe permitir su distribución tanto como código fuente como compilado. Cuando de algún modo no se distribuya el código fuente junto con el producto, deberá proveerse un medio conocido para obtener el código fuente sin cargo, a través del Internet. El código fuente es la forma preferida en la cual un programador modificará el programa.

- **Trabajos derivados**

La licencia debe permitir modificaciones y trabajos derivados, y debe permitir que estos se distribuyan bajo las mismas condiciones de la licencia del software original.

- **Integridad del código fuente del autor**

La licencia puede restringir la distribución de código fuente modificado *sólo* si se permite la distribución de 'patch files' con el código fuente con el propósito de modificar el programa en tiempo de construcción. La licencia puede requerir que los trabajos derivados lleven un nombre o número de versión distintos a los del software original.

- **No discriminar personas o grupos**

La licencia no debe hacer discriminación de personas o grupos de personas.

- **No discriminar campos de aplicación**

La licencia no debe restringir el uso del programa en un campo específico de aplicación.

- **Distribución de la licencia**

Los derechos concedidos deben ser aplicados a todas las personas a quienes se redistribuya el programa, sin necesidad de obtener una licencia adicional.

- **La licencia no debe ser específica a un producto**

Los derechos aplicados a un programa no deben depender de la distribución particular de software de la que forma parte. Si el programa es extraído de esa distribución y usado o distribuido dentro de las condiciones de la licencia del programa, todas las personas a las que el programa se redistribuya deben tener los mismos derechos que los concedidos en conjunción con la distribución original de software.

- **La licencia no debe contaminar otro software**

La licencia no debe imponer restricciones sobre otro software que es distribuido junto con el.

- **Ejemplos de licencias**

Las licencias GNU GPL, BSD, X Consortium, Artistic son ejemplos de licencias que se consideran que cumplen con la definición de Open Source.[15]



Figura 2.10: Ejemplos de Open Source

## 2.6. Open Source Hardware

Hardware de Fuentes Abiertas (**OSHW** en inglés) es aquel hardware cuyo diseño se hace disponible públicamente para que cualquier persona lo pueda estudiar, modificar, distribuir, materializar y vender, tanto el original como otros objetos basados en ese diseño.

El **Hardware de fuentes abiertas** es un término para denominar artefactos tangibles -máquinas, dispositivos, u otros objetos del mundo físico- cuyo diseño ha sido publicado de forma tal que cualquier persona pueda fabricar, modificar, distribuir y usar esos objetos. Esta definición tiene la intención de proveer una guía para el desarrollo y evaluación de licencias para 'Hardware de fuentes abiertas'.

Los términos de distribución del **Hardware de fuentes abiertas** habrán de seguir los siguientes criterios: [13]

#### ■ **Documentación**

El hardware liberado ha de incluir documentación en la forma de ficheros de diseño y deberá permitir la modificación y redistribución de los mismos.

#### ■ **Alcance**

La documentación del hardware deberá especificar claramente que parte del diseño, sino todo, se libera bajo la licencia.

#### ■ **Programas informáticos necesarios**

Si el diseño bajo licencia necesita de un paquete de informático, bien como parte del mismo, bien para operar de forma apropiada y cumplir con sus funciones básicas, la licencia podría requerir que se cumplieran alguna de las condiciones siguientes:

- Que las interfaces habrán de estar documentados suficientemente como para considerar la posibilidad de crear un paquete informático en código abierto que permitan al dispositivo operar de forma apropiada y cumplir con sus funciones básicas.
- Que el paquete informático necesario venga liberado bajo una licencia de código abierto aprobada por la OSI.

#### ■ **Obras derivadas**

La licencia deberá permitir modificaciones y obras derivadas, y permitirá que éstas se distribuyan bajo los mismo términos que la licencia de la obra original.

#### ■ **Libre distribución**

La licencia no podrá restringir a nadie de la venta o distribución de la documentación del proyecto. La licencia no podrá requerir el pago de derechos de autor por la mencionada venta.

#### ■ **Atribución**

La licencia podría requerir de los documentos derivados y notificaciones de derechos de copia asociadas con los dispositivos atribuyan la autoría del/los autor/es licenciante/s a la hora de distribuir ficheros de diseño, bien manufacturados y/o productos derivados de los mismo.

#### ■ **No discriminación a personas o grupos**

La licencia no puede discriminar ninguna persona o grupo de personas.

#### ■ **No discriminación a campos de aplicación**

La licencia no puede restringir a nadie de hacer uso del trabajo (incluyendo el objeto manufacturado) en un campo específico de aplicación.

- **Distribución de la licencia**

Los derechos proporcionados por la licencia deberán ser aplicados a todos aquellos a los que sea redistribuido el trabajo sin la necesidad de ejecutar una licencia adicional.

- **La licencia no será específica a un producto**

Los derechos proporcionados por la licencia no dependen de que el trabajo licenciado sea parte de un producto determinado. Si una parte de una obra licenciada se usa y distribuye bajo los términos de la licencia, todos aquellos a los que se les redistribuya la obra deberán tener los mismos derechos que proporcione la obra original.

- **La licencia no deberá restringir otro Hardware o Software**

La licencia no deberá colocar restricciones a elementos añadidos a la obra con el trabajo licenciado pero no derivados de él.

- **La licencia será neutra en términos tecnológicos**

Ninguna de las cláusulas de la licencia dependerá de una tecnología específica, componente, material o estilo de interface o uso de la misma.

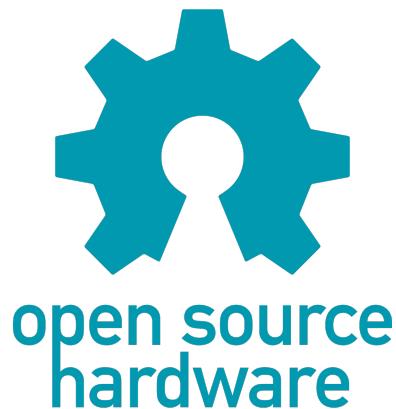


Figura 2.11: Logo de Open Source Hardware

## 2.7. Introducción a ROS

### 2.7.1. ¿Qué es ROS y por qué debe utilizarse?

La comunidad de la robótica ha evolucionado a pasos agigantados en los últimos años en lo que se refiere al alcance de servicios prestados y autonomía. Sin embargo, no hay que obviar la considerable dificultad que supone el desarrollo de los mismos. Desde el punto de vista software, se presenta una plataforma llamada ROS (*Robot Operative System*) que intenta unificar y facilitar el desarrollo de los robots.

La definición de ROS, según su página oficial es la siguiente:

“ROS (*Robot Operating System*) provee librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes y más. ROS está bajo la licencia open source, BSD.” [20]

Como se puede observar en la definición, ROS no es un sistema operativo (Véase la página 49) en sí mismo, sino que trabaja conjuntamente con otros sistemas operativos para prestar nuevos servicios a la hora de desarrollar software para robots. Sin la aportación de estos servicios, la inversión de tiempo y energía para el aprendizaje de ROS carecería de sentido. Las principales ventajas que ofrece ROS se resumen enseguida: [9]

- **Computación distribuida:** Los sistemas de robótica modernos están basados en software que usan una infinidad de procesadores y ordenadores distintos, necesitando un sistema de comunicación entre ellos. ROS contiene mecanismos de comunicación que cubren estas necesidades.
- **Reutilización de Software:** A menudo las tareas que desarrolla un robot son comunes, permitiendo la reutilización del código desarrollado para las mismas. La reutilización del software solo es posible si se permite una integración sencilla. ROS permite esta reutilización al proveer paquetes de código estable y unas interfaces estándar que aportan interoperabilidad a los robots.
- **Análisis/pruebas rápidas:** Uno de los mayores retos que presenta el desarrollo de software para robots es la complejidad a la hora de realizar test. La disponibilidad de robots a la hora de realizar pruebas no siempre es posible. ROS proporciona sistemas de simulación que sustituyen a hardware/software normalmente requerido y permite la reproducción de datos de sensores y otro tipo de mensajes.

Finalmente, se debe aclarar lo que “ROS no es”:

- ROS no es un lenguaje de programación: El hecho de que ROS esté escrito en C ++ no limita la implementación de librería en otros lenguajes como Python, Java o Lisp.
- ROS no es una librería: Aunque ROS incluye numerosas librerías, también podemos encontrar un servidor central, herramientas y un sistema de compilación.
- ROS no es un sistema de desarrollo integrado (IDE, por sus siglas en inglés): ROS no está ligado a ningún IDE, aunque puede ser usado con los más populares.



Figura 2.12: Logo ROS

### 2.7.2. Sistemas operativos compatibles con ROS

Debido a que ROS no es un sistema operativo, sino un **framework** (Véase la página 47) que proporciona una serie de servicios y librerías, es necesario utilizar un sistema operativo compatible. A continuación, se mencionan los principales sistemas operativos compatibles con ROS: [7]

- **Ubuntu:**

Sistema operativo de código abierto (Véase la página 22) basado en una distribución de Linux. Principalmente usado en ordenadores personales (PCs), enfocado en la facilidad de uso y la mejora de la experiencia del usuario. ROS es compatible con las siguientes versiones de Ubuntu:

- Wily: amd64 i386
- Xenial: amd64 i386 armhf



Figura 2.13: Logo Ubuntu

■ **Debian:**

Sistema operativo libre formado por un conjunto de programas y utilidades básicas basadas en un núcleo de Linux o FreeBSD. ROS es compatible con las siguientes versiones de Debian:

- Wheezy: amd64 arm64
- Jessie: amd64 arm64



Figura 2.14: Logo Debian

■ **OS X (HomeBrew):**

Sistema operativo basado en Unix desarrollado y comercializado por Apple Inc. Este sistema operativo es utilizado en la gama de computadoras Macintosh. A la

hora de realizar instalaciones relativas a ROS se necesita el gestor de instalaciones HomeBrew.



Figura 2.15: Logo HomeBrew

■ **Gentoo:**

Sistema operativo de código abierto basado en Linux o FreeBSD. Teniendo como base de sus funciones principales la distribución Portage, se puede utilizar tanto en ordenadores personales, servidores o sistemas empotrados. Provee sistemas de compilación e instalación, además de actualizaciones automáticas.

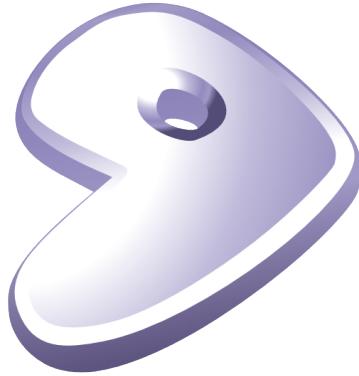


Figura 2.16: Logo Gentoo

■ **OpenEmbedded/Yocto:**

*Framework* de código abierto usado para la creación de distribuciones basadas en Linux, mantenido por *YoctoProject* y *OpenEmbedded Project*. Principalmente utilizado en sistemas empotrados, aunque no está en distintas capas de aplicación y librerías que forman un conjunto de metadatos.



Figura 2.17: Logo OpenEmbedded

- **Android NDK:**

Android NDK es una herramienta complementaria del SDK de Android que permite reutilizar librerías y código a través de JNI (*Java Native Interface*).



Figura 2.18: Logo Android NDK

### 2.7.3. Distribuciones de ROS

Una distribución de ROS es un conjunto de versionado de librerías, las cuales permiten a los desarrolladores trabajar con una versión estable de código. Cuando estas distribuciones se publican, los cambios se limitan a correcciones de fallos y modificaciones que no alteren el funcionamiento de los módulos principales de la distribución. A continuación, se presentan las distribuciones más importantes y recomendadas respecto a ROS: [8]

- **ROS Noetic:**

- Publicada: 23/05/2020

- Plataformas:
  - Ubuntu: Focal



Figura 2.19: Logo ROS Noetic

■ ROS Melodic:

- Publicada: 23/05/2018
- Plataformas:
  - Ubuntu: Bionic; Artful
  - Mac OS X
  - Andriod
  - Windows
  - Debian: Stretch



Figura 2.20: Logo ROS Melodic

▪ **ROS Lunar:**

- Publicada: 23/05/2017
- Plataformas:
  - Ubuntu: Willy; Xenial
  - Debian: Stretch



Figura 2.21: Logo ROS Lunar

▪ **ROS Kinetic:**

- Publicada: 23/05/2016
- Plataformas:
  - Ubuntu: Willy; Xenial

- Debian: Jessie
- OS X (HomeBrew)
- Gentoo
- OpenEmbedded/Yocto



Figura 2.22: Logo ROS Kinetic

■ **ROS Jade:**

- Publicada: 23/05/2015
- Plataformas:
  - Ubuntu
  - OS X (HomeBrew)
  - Gentoo
  - Android (NDK)



Figura 2.23: Logo ROS Jade

- **ROS Indigo:**

- Publicada: 22/04/2014
- Plataformas:
  - Ubuntu
  - Debian - Wheezy
  - OS X (HomeBrew)
  - Gentoo
  - OpenEmbedded/Yocto
  - Android (NDK)

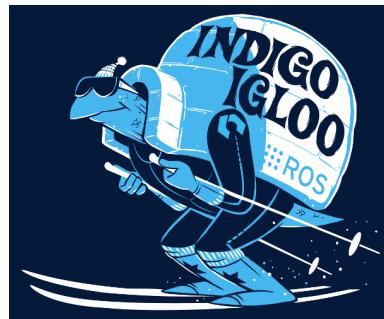


Figura 2.24: Logo ROS Indigo

#### 2.7.4. Software

Ros tiene compatibilidad con una gran cantidad sistemas operativos.

La selección del sistema operativo viene determinada por la compatibilidad con ROS y la cantidad de documentación necesaria para la instalación del mismo. Debido a que ROS.org recomienda la utilización de la distribución kinetic para el control de drones, robots o vehículos a escala. En este caso se ha seleccionado *Ubuntu Mate 16.04* por ser el sistema operativo oficial por su fácil implementación y tener la mayor cantidad de documentación proporcionada por ROS.

# Capítulo 3

## Implementación

### 3.1. Instalación de ROS en Ubuntu

#### 3.1.1. Requisitos de instalación

- **Versión de Ubuntu 16.04:** ROS tiene una gran variedad de sistemas operativos con los que es compatible y uno de los más usados y con mayor número de documentación es Ubuntu. La versión que mejor funcionamiento ha tenido en conjunto con ROS ha sido Ubuntu 16.04, la cual será la versión a utilizar en nuestro desarrollo.

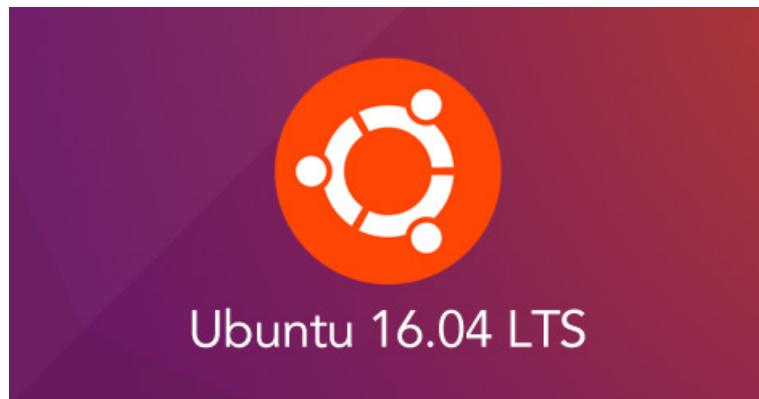


Figura 3.1: Logo Ubuntu 16.04

## 3.2. Instalación de ROS

Dicho proceso de instalación es explicado en el anexo A con cada uno de los comandos necesarios para llevarla a cabo.

## 3.3. Instalación de la librería ros\_lib en el entorno de Arduino

Para poder utilizar ROS en el entorno de Arduino es necesario realizar la instalación y actualización de la librería ros\_lib, la cual permite a los programas de Arduino interactuar con ROS. Dicho proceso se encuentra en la comunidad de ROS y detallado en el anexo B [24].

En la figura 3.2 se muestra un ejemplo del uso de la comunicación entre ROS y Arduino UNO, donde se emite un intercambio de información.

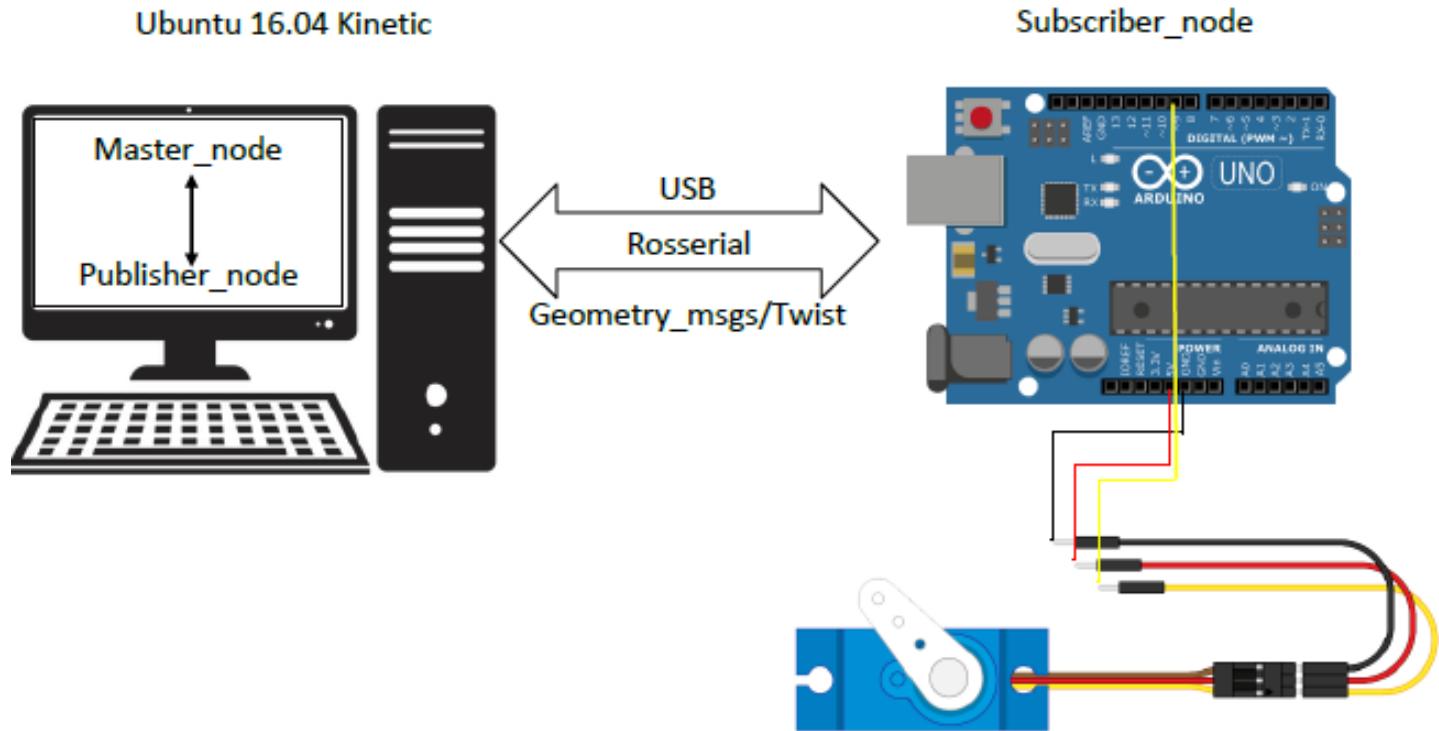


Figura 3.2: Comunicación entre ROS y Arduino UNO

# Capítulo 4

## Pruebas

### 4.1. Comunicación Arduino y ROS

Este capítulo tiene la finalidad de explicar cada una de las prácticas realizadas para establecer la comunicación entre Arduino y ROS describiendo detalladamente el proceso de cada una de ellas. Dichos procesos se encuentran disponibles en la comunidad de ROS.

#### 4.1.1. Ejemplo de servocontrolador

Este ejemplo explica cómo controlar un Servo R/C (Véase la página 48) usando Arduino y un **rosserial**. Esto puede usarse para controlar un mecanismo de liberación, un brazo de robot sencillo o cualquier cosa donde se necesite de un actuador bárato. El código que proporciona el ejemplo, es un código básico y muestra las funciones básicas de un servo.

#### Hardware

En la figura 4.1 se muestra la conexión de los dispositivos requeridos para la realización del ejemplo de comunicación entre ROS y Arduino UNO.

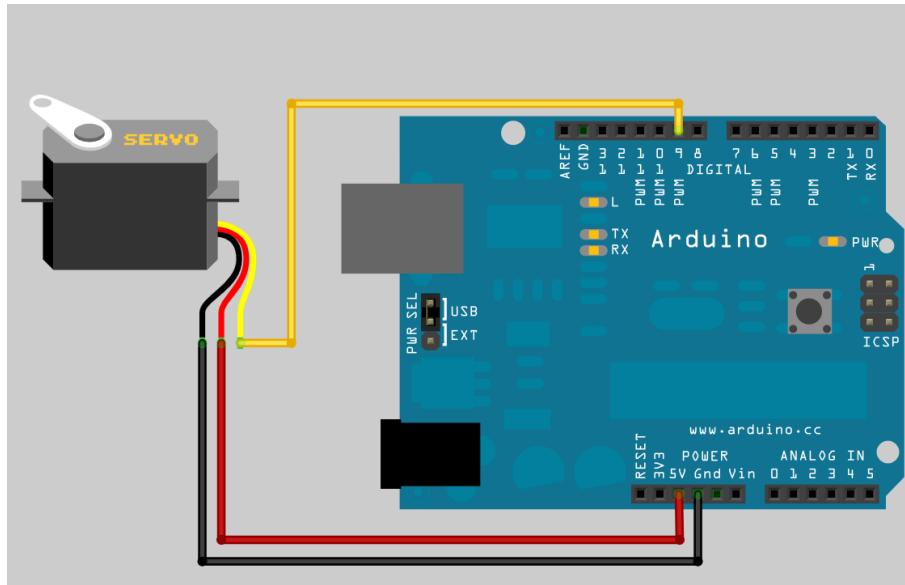


Figura 4.1: Conexión de Servo

## Software

El código de este ejemplo esta basado en el uso de la biblioteca de **Servo Arduino**. Este boceto muestra el control de la dirección de un Servo usando ROS y Arduino.

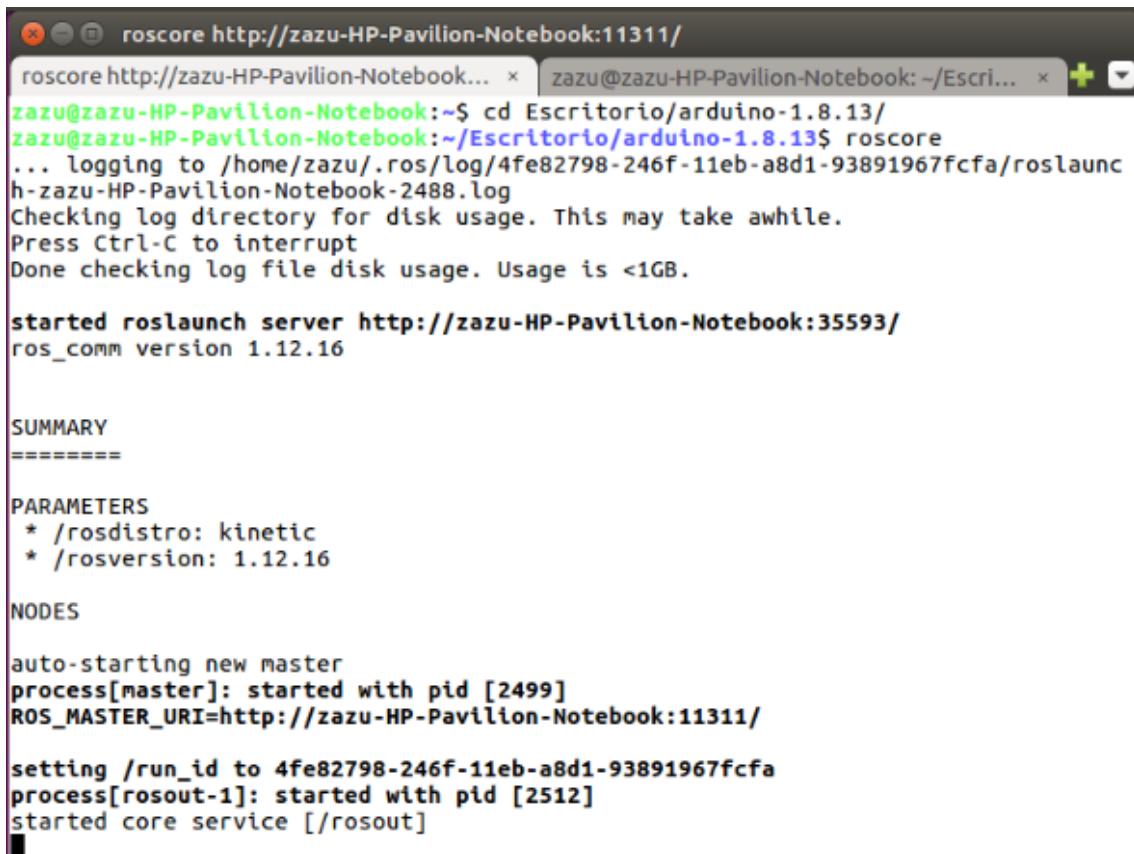
El siguiente algoritmo se puede encontrar en el sitio oficial de ROS [http://wiki.ros.org/rosserial\\_arduino/Tutorials/Servo%20Controller](http://wiki.ros.org/rosserial_arduino/Tutorials/Servo%20Controller) con una breve explicación del funcionamiento.

En el anexo C se muestra el algoritmo usado para la realización de la práctica que se describe.

Las áreas clave específicas del servo aquí son el hecho de que hicimos un objeto Servo global, conectado al pin arduino correcto, y luego en cada llamada de tema de servo, escribimos el nuevo ángulo del servo al objeto servo.

## Demostración

Primero se inicia el **roscore** (Figura 4.2) que es con el comando que se ejecuta las librerías de ROS en su propia ventana terminal.



The screenshot shows a terminal window with two tabs. The active tab displays the output of the command `roscore http://zazu-HP-Pavilion-Notebook:11311/`. The log output is as follows:

```
zazu@zazu-HP-Pavilion-Notebook:~$ cd Escritorio/arduino-1.8.13/
zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ roscore
... logging to /home/zazu/.ros/log/4fe82798-246f-11eb-a8d1-93891967fcfa/roslaunc
h-zazu-HP-Pavilion-Notebook-2488.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://zazu-HP-Pavilion-Notebook:35593/
ros_comm version 1.12.16

SUMMARY
========
PARAMETERS
* /rosdistro: kinetic
* /rosversion: 1.12.16

NODES

auto-starting new master
process[master]: started with pid [2499]
ROS_MASTER_URI=http://zazu-HP-Pavilion-Notebook:11311/

setting /run_id to 4fe82798-246f-11eb-a8d1-93891967fcfa
process[rosout-1]: started with pid [2512]
started core service [/rosout]
```

Figura 4.2: Inicio de librerias de ROS

Como siguiente paso se realiza la ejecución del nodo **rosserial** (Figura 4.3) de python en su ventana de terminal, esto con el fin de declarar el puerto de la placa Arduino y hacer la comunicación serial entre arduino y ROS.

```

zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ rosrun rosserial_python serial_node.py /dev/ttyACM0
[INFO] [1605138922.859522]: ROS Serial Python Node
[INFO] [1605138922.872577]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1605138924.985502]: Requesting topics...
[INFO] [1605138925.018719]: Note: subscribe buffer size is 280 bytes
[INFO] [1605138925.019869]: Setup subscriber on servo [std_msgs/UInt16]

```

Figura 4.3: Configuración de puerto serial

Y finalmente, en una nueva ventana de terminal, se usa **rostopic pub** (Figura 4.4) para el control del servo. Al final del comando solo se debe especificar el ángulo de 0-180.

```

zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ rostopic pub servo std_msgs/UInt16 45
publishing and latching message. Press ctrl-C to terminate
^Czazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ rostopic pub servo std_msgs/UInt16 0
publishing and latching message. Press ctrl-C to terminate
^Czazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ 

```

Figura 4.4: Comunicación entre Arduino, ROS y Servo

#### 4.1.2. Ejemplo de blink

Rosserial permite la comunicación de hardware basado en Arduino con ROS. Este ejemplo explica cómo usar el led incluído en la placa Arduino.

##### Hardware

En este ejemplo no es necesaria una conexión de dispositivos electrónicos, basta con conocer el puerto donde se permite la conexión del led de la placa que sería el puerto 13 (Figura 4.5).

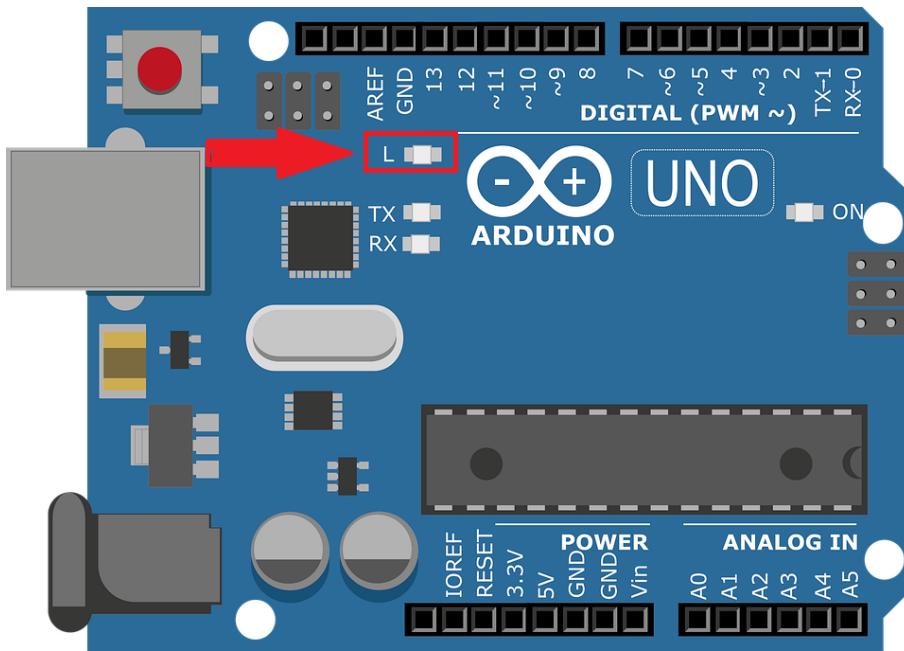


Figura 4.5: Ubicación del led en la placa Arduino UNO

## Software

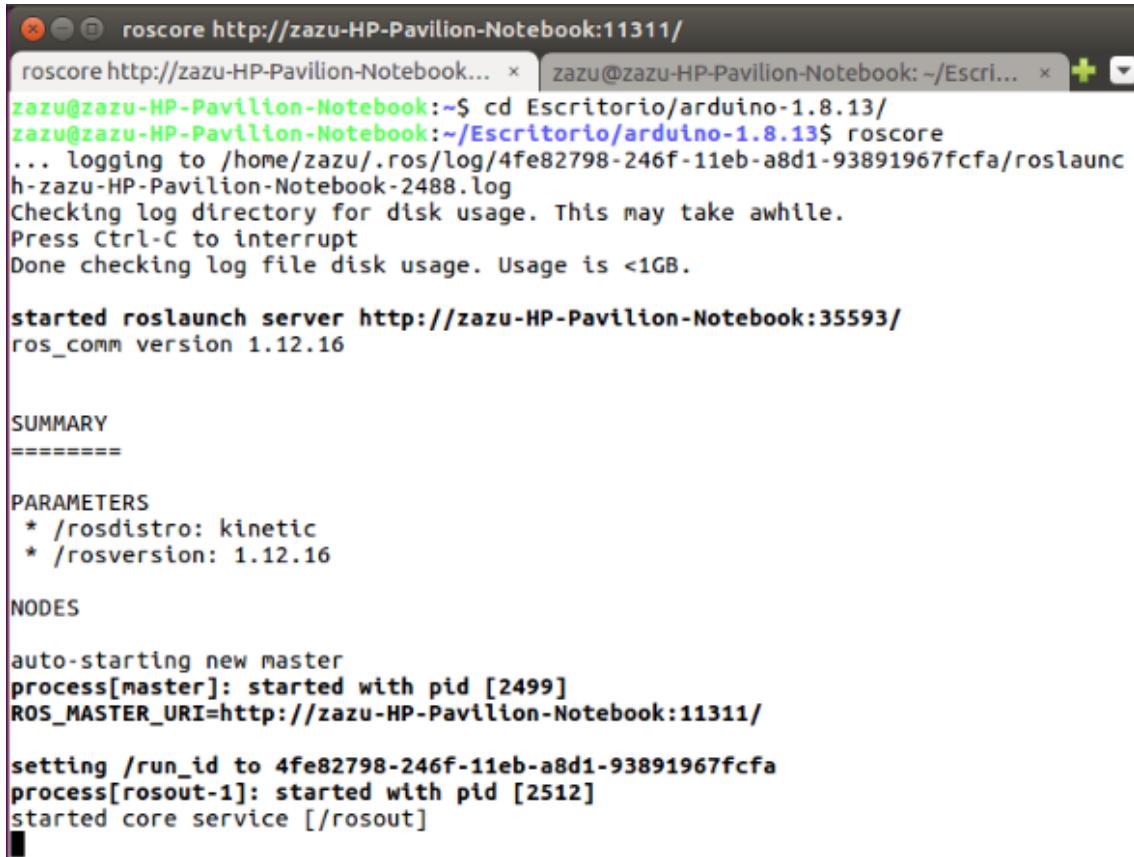
Como siguiente punto, se muestra el código para el uso y control del encendido y apagado del led localizado en la placa Arduino UNO.

El siguiente algoritmo se puede encontrar en el sitio oficial de ROS [http://wiki.ros.org/rosserial\\_arduino/Tutorials/Blink](http://wiki.ros.org/rosserial_arduino/Tutorials/Blink) con una breve explicación del funcionamiento.

En el anexo D se muestra el algoritmo usado para la realización de la práctica que se describe.

## Demostración

Para finalizar, se inicia una ventana de terminal donde se ejecutará el comando para iniciar las librerías **roscore** (Figura 4.6).



```

roscore http://zazu-HP-Pavilion-Notebook:11311/
roscore http://zazu-HP-Pavilion-Notebook... x zazu@zazu-HP-Pavilion-Notebook: ~/Escr... x + ▾
zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ cd Escritorio/arduino-1.8.13/
zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ roscore
... logging to /home/zazu/.ros/log/4fe82798-246f-11eb-a8d1-93891967fcfa/roslaunc
h-zazu-HP-Pavilion-Notebook-2488.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://zazu-HP-Pavilion-Notebook:35593/
ros_comm version 1.12.16

SUMMARY
=====
PARAMETERS
* /rosdistro: kinetic
* /rosversion: 1.12.16

NODES

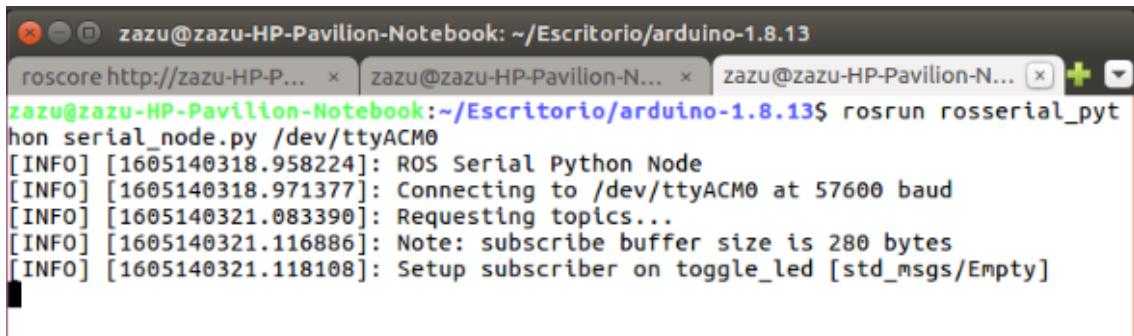
auto-starting new master
process[master]: started with pid [2499]
ROS_MASTER_URI=http://zazu-HP-Pavilion-Notebook:11311/

setting /run_id to 4fe82798-246f-11eb-a8d1-93891967fcfa
process[rosout-1]: started with pid [2512]
started core service [/rosout]

```

Figura 4.6: Inicio de librerías de ROS

Como siguiente paso se realiza la ejecución del nodo **rosserial** (Figura 4.7) de python en su ventana de terminal, esto con el fin de declarar el puerto de la placa Arduino y hacer la comunicación serial entre arduino y ROS.



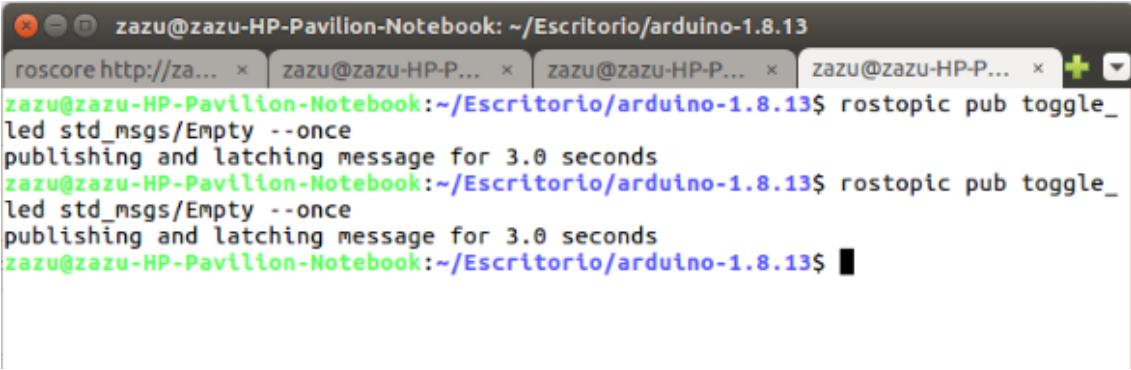
```

zazu@zazu-HP-Pavilion-Notebook: ~/Escritorio/arduino-1.8.13
roscore http://zazu-HP-P... x zazu@zazu-HP-Pavilion-N... x zazu@zazu-HP-Pavilion-N... x + ▾
zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ rosrun rosserial_pyt
hon serial_node.py /dev/ttyACM0
[INFO] [1605140318.958224]: ROS Serial Python Node
[INFO] [1605140318.971377]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1605140321.083390]: Requesting topics...
[INFO] [1605140321.116886]: Note: subscribe buffer size is 280 bytes
[INFO] [1605140321.118108]: Setup subscriber on toggle_led [std_msgs/Empty]

```

Figura 4.7: Configuración de puerto serial

Y por último, se abre una nueva ventana de terminal con el comando **rostopic pub toggle\_led std\_msgs/Empty --once** (Figura 4.8) que es quien permite la inicialización del encendido o apagado del led durante tres segundos.



The screenshot shows a terminal window titled "zazu@zazu-HP-Pavilion-Notebook: ~/Escritorio/arduino-1.8.13". It contains the following command and its execution:

```
zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ rostopic pub toggle_led std_msgs/Empty --once
publishing and latching message for 3.0 seconds
zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$ rostopic pub toggle_led std_msgs/Empty --once
publishing and latching message for 3.0 seconds
zazu@zazu-HP-Pavilion-Notebook:~/Escritorio/arduino-1.8.13$
```

Figura 4.8: Comunicación de ROS, Arduino y Led puerto 13

## 4.2. Control de dispositivo autónomo

Para finalizar con las pruebas emitidas de la comunicación de ROS y Arduino, se ha implementado, en conjunto a ellos, un tercer algoritmo que permite el control más detallado de un dispositivo mediante órdenes previstas.

El mencionado algoritmo esta expuesto y explicado en el anexo E.

# **Capítulo 5**

## **Resultados y conclusiones**

La realización del proyecto llevaba asociada la misión de integrar la plataforma de propósito general Arduino UNO con ROS (Robot Operating System) la cual se ha cumplido de forma exitosa.

Las ventajas y alternativas que ofrece ROS a la hora de abordar el diseño y programación de aplicaciones robot son muy extensas pues al situarse como esa capa de abstracción entre las diferentes plataformas hace mucho más sencillo el establecimiento de conexión, el paso de mensajes y comunicación.

Por otro lado las carencias más importantes que presentaba el dispositivo central de control de ROS para la realización de este proyecto:

- Conflictos de compatibilidad de versiones.
- Conflictos de sistemas operativos en diferentes particiones del disco duro.

Estos inconvenientes se solventan gracias a la adición de mayor información sobre la versión de ROS usada a lo largo del proyecto, así como una reinstalación de ambos sistemas operativos permitiendo una correcta convivencia dentro del disco duro.

En cualquier caso la finalización de este proyecto, más que representar un final como tal, va mucho más allá y abre las puertas a unas más que interesantes líneas de investigación y desarrollo que podrían llegar a ser muy útiles de cara a interés personal pero sobre todo abre un sinfín de posibilidades de cada a su continuación y/o trabajo sobre las bases establecidas para la Universidad:

- Realización de laboratorios y talleres en el ámbito del control y programación de robots.
- Líneas de desarrollo y programación de aplicaciones robot.
- Investigación y desarrollo de aplicaciones más complejas y completas.
- Impartición y aprendizaje de nuevos conocimientos a nivel de software y hardware.

## 5.1. Trabajos a futuro

Como se ha comentado anteriormente la finalización de este proyecto no supone un punto y final a nivel de desarrollo o funcionalidades.

El proyecto realizado se podrían extender de diferentes formas:

- En primer lugar y con el objetivo de obtener una plataforma más compacta que ofrezca viabilidad para su montaje y funcionamiento autónomo y una integración limpia con el hardware y plataformas utilizados, se podrá diseñar una placa PCB mediante **Altium** (Véase la página 47) que actúe como escudo (shield) para Arduino UNO. Lo que se pretendía sería conseguir la integración limpia y clara del hardware que presente características de fiabilidad de las conexiones, estabilidad y robustez, escalabilidad del diseño e integración.
- Añadir componentes extra que doten al sistema de funcionalidades extra fácilmente adaptables:
  - Receptor GPS evitando la perdida constante de la localización del dispositivo.
  - Conjunto de sensores que permitan la función de tomar la medición de diversas variables.
- Implementación de herramientas de navegación de ROS tales como Rviz, que permitiría la representación de obstáculos en un mapa autogenerado, introducción de datos de ubicación y planificación de rutas.
- Examinar la idea de un cambio de sistema embebido, es decir, hacer un cambio de controladora principal por un sistema embebido de mayor capacidad y escalabilidad. Se propone la idea de la ESP32 que es compatible con los algoritmos ya aplicados y que, actualmente, ya cuenta con una amplia gama de información sobre el tema.

# Capítulo 6

## Glosario

- **Altium Designer:** Es un conjunto de programas para el diseño electrónico en todas sus fases y para todas las disciplinas, ya sean esquemas, simulación, diseño de circuitos impresos o desarrollo de código para microprocesadores. [26]
- **Arduino:** Es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso. [33]
- **Arduino UNO:** Es una placa electrónica basada en el microcontrolador ATmega328. La placa incluye todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador. [35]
- **CPU:** La unidad central de procesamiento (CPU) es el hardware dentro de un ordenador u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema. [22]
- **Dispositivo autónomo:** En el campo de la informática un dispositivo autónomo es aquel que no necesita estar conectado al ordenador para funcionar. [29]
- **Framework:** En el desarrollo de Software, un framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, librerías y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. [12]

- **FreeBSD:** Es un Sistema Operativo libre de tipo Unix. [19]
- **Hardware:** Es una palabra inglesa que hace referencia a las partes físicas tangibles de un sistema informático, es decir, todo aquello que podemos tocar con las manos. Dentro del hardware encontramos una gran variedad de componentes eléctricos, electrónicos, electromecánicos y mecánicos. También es definido como el conjunto de los componentes que integran la parte material de una computadora. [30]
- **Linux:** Es un Sistema Operativo como MacOS, DOS o Windows. Es decir, Linux es el software necesario para que el ordenador permita utilizar programas como: editores de texto, juegos, navegadores de Internet, etc. [10]
- **Middleware:** Es software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él. Básicamente, funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas. A veces, se le denomina “lumbing” (tuberías), porque conecta dos aplicaciones para que se puedan pasar fácilmente datos y bases de datos por una “canalización”. [36]
- **Robótica:** Se puede definir como una ciencia que aglutina varias ramas tecnológicas o disciplinas, con el objetivo de diseñar máquinas robotizadas que sean capaces de realizar tareas automatizadas o de simular el comportamiento humano o animal, en función de la capacidad de su software. [27]
- **Robot Operating System:** Es un middleware robótico, es decir, una colección de frameworks para el desarrollo de software de robots. A pesar de no ser un sistema operativo, ROS provee los servicios estándar de uno de estos tales como la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. [34]
- **Servo R/C:** El servo es uno de los elementos más reconocibles del radiocontrol. Este se encarga de transformar las órdenes enviadas desde nuestra emisora en movimientos de rotación. Estos movimientos a su vez se convierten en movimientos de desplazamientos de las diferentes superficies de control. Esto nos sirve tanto para controlar en el aire un avión como para dirigir un coche de R/C. [25]
- **Sistema distribuido:** Se define como una colección de computadores conectados por una red, y con el software distribuido adecuado para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar facilidades de computación. [31]

- **Sistema operativo:** Es un conjunto de programas que mediante órdenes electrónicas, controlan la actividad total del computador. [6] Es un programa (Software) que cuando inicia el ordenador, se encarga de gestionar todos los recursos del sistema informático, tanto del hardware como del software, permitiendo así la comunicación entre el usuario y el ordenador. [7]
- **Software:** Es un término informático que hace referencia a un programa o conjunto de programas de cómputo, así como datos, procedimientos y pautas que permiten realizar distintas tareas en un sistema informático. [28]
- **Unix:** Es un Sistema Operativo, es decir, es una colección de programas que ejecutan otros programas en una computadora. [11]

# Bibliografía

- [1] Plaza, P. (2015). Posicionamiento 3-D Autónomo de Instrumentos Quirúrgicos en Cirugía Laparoscópica (Tesis de fin de grado). Universidad Carlos III de Madrid, Madrid, España. P.15.
- [2] Romero A. (2014). Integración de ROS con Arduino y Raspberry Pi (Tesis de pregrado). Universidad de Sevilla, Sevilla, España.
- [3] Cabalgente, R. (2015). Diseño de un Robot Móvil para estudio de la ocupación en habitaciones. (Tesis de fin de grado). Universidad Carlos III de Madrid, Madrid, España.
- [4] Bergmann, P. (2018). Sistema de supervisión y telemetría para un robot móvil con pila de combustible. (Proyecto fin de grado). Universidad de Sevilla, Sevilla, España.
- [5] Ortego, D. (2017). Qué es ROS (Robot Operating System). Septiembre, 7, 2020, de OpenWebinars Sitio web: <https://openwebinars.net/blog/que-es-ros/>
- [6] (2014). Sistema Operativo. Septiembre 28, 2020, de Concepto Definición. Sitio web: <https://conceptodefinicion.de/sistema-operativo/>
- [7] (2013). Sistemas Operativos. Septiembre 28, 2020, de Área Tecnológica. Sitio web: <https://www.areatecnologia.com/sistemas-operativos.htm>
- [8] Recio, I. (2019). Ros.org: Distributions. Septiembre 27, 2020, de Wiki. Sitio web: <http://wiki.ros.org/Distributions>
- [9] Recio, I. (2018). Ros. org: Documentation. Septiembre 27, 2020, de Wiki. Sitio web: <http://wiki.ros.org/>
- [10] (2018). Que es Linux. Septiembre 28, 2020, de Ciberaula. Sitio web: [http://linux.ciberaula.com/articulo/que\\_es\\_linux/](http://linux.ciberaula.com/articulo/que_es_linux/)

- [11] González, G. (2014). UNIX: uno de los sistemas operativos más importantes en la historia de la computación. Septiembre 28, 2020, de Hipertextual. Sitio web: <https://hipertextual.com/archivo/2014/05/que-es-unix/>
- [12] (2017). Framework. Septiembre 28, 2020, de EcuRed. Sitio web: <https://www.ecured.cu/Framework>
- [13] Cuartielles, D. (2019). Open Source Hardware Association. Septiembre 27, 2020, de OSHWA. Sitio web: <https://www.oshwa.org/definition/spanish/>
- [14] Yúbal, F. (2018). Qué es Arduino, cómo funciona y qué puedes hacer con uno. Septiembre, 13, 2020, de Xataka Sitio web: <https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno>
- [15] (2015). Definición de Open Source Septiembre. 12, 2020, (Tesis de maestría) Sitio web: <http://www.fime.uanl.mx/jcedillo/Definic%EDondeOpenSourceySoftwareLibre.pdf>
- [16] Gómez, G. and Gómez, G. (2016) ¿Qué es Open Source?. Septiembre, 12, 2020 de Tecnología fácil Sitio web: <https://tecnologia-facil.com/que-es/que-es-open-source/>
- [17] Muñoz de Frutos, A. (2016). ¿Qué es Firmware?. Septiembre 12, 2020, de Computer Hoy. Sitio web: <https://computerhoy.com/noticias/software/que-es-firmware-53182>
- [18] SAP Copyright Departmen. <https://www.sap.com/latinamerica/trends/internet-of-things.html>
- [19] (2013). FreeBSD. Septiembre 28, 2020, de EcuRed. Sitio web: <https://www.ecured.cu/FreeBSD>
- [20] Recio, I. (2018). ROS.org. Septiembre 27, 2020, de Wiki. Sitio web: <http://wiki.ros.org/es>
- [21] Guerrero, J. (2014). Arduino Uno: Especificaciones y características. Septiembre, 13, 2020, de Pluselectric Sitio web: <https://pluselectric.wordpress.com/2014/09/21/arduino-uno-especificaciones-y-caracteristicas/>
- [22] (2017). Unidad central de procesamiento. Septiembre, 13, 2020, de Wikipedia Sitio web: [https://es.wikipedia.org/wiki/Unidad\\_central\\_de\\_procesamiento](https://es.wikipedia.org/wiki/Unidad_central_de_procesamiento)
- [23] Zhao, F.(2019). Installing ROS Kinetic on the Raspberry Pi. Septiembre, 13, 2020, de Wiki. Sitio web: <http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi>

- [24] Zhao, F. (2019) Arduino IDE Setup. Septiembre, 13, 2020 de Wiki. Sitio web: [http://wiki.ros.org/rosserial\\_arduino/Tutorials/Arduino%20IDE%20Setup](http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup)
- [25] Anónimo. (2017). ¿Qué es un Servo? Elemento básico para el R/C. Septiembre, 13, 2020, de FpvMax. Sitio web: <http://fpvmax.com/2017/01/30/servo-elemento-basico-rc/>
- [26] Anónimo. (2008). ¿Qué es Altium Designer?. Noviembre, 06, 2020, de Redeweb.com. Sitio web: [https://www.redeweb.com/\\_txt/643/26.pdf](https://www.redeweb.com/_txt/643/26.pdf)
- [27] (2020). ¿Qué es la robótica?. Octubre 04, 2020, de Revista de Robots Sitio web: <https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/>
- [28] (2013). Significado de Software. Octubre 12, 2020, de Significados. Sitio web: <https://www.significados.com/software/>
- [29] (2003). Dispositivo autónomo. Octubre 12, 2020, de Enciclopedia. Sitio web: [http://enciclopedia.us.es/index.php/Dispositivo\\_aut%C3%B3nomo](http://enciclopedia.us.es/index.php/Dispositivo_aut%C3%B3nomo)
- [30] Richard, S. (2016). ¿Qué es el hardware? Para qué sirve y definición. Octubre 12, 2020 de Profesional review. Sitio web: <https://www.profesionalreview.com/hardware/>
- [31] Anónimo (2019). Sistemas distribuidos. Octubre 12, 2020, de Blogspot. Sitio web: <http://sistemasdistribuidosaisseccion1.blogspot.com/p/definicion-de-los-sistemas-distribuidos.html>
- [32] Díaz, J. (2015). Placa Arduino UNO. Octubre 12, 2020, de MiArduino. Sitio web: <http://www.iescamp.es/miarduino/2016/01/21/placa-arduino-uno/>
- [33] Fernández, Y. (2020). Qué es Arduino, cómo funciona y qué puedes hacer con uno. Octubre 12, 2020, de Xataka.com. Sitio web: <https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno>
- [34] Ortego, D. (2017). Qué es ROS (Robot Operating System). Noviembre 17, 2020, de OpenWebinars. Sitio web: <https://openwebinars.net/blog/que-es-ros/>
- [35] Anónimo. (2017). Qué es la programación con arduino y para qué sirve. Octubre 12, 2020, de BeJob. Sitio web: <https://www.bejob.com/que-es-la-programacion-con-arduino-y-para-que-sirve/>
- [36] (2016). ¿Qué es middleware?. Octubre 18, 2020, de Microsoft Azure. Sitio web: <https://azure.microsoft.com/es-es/overview/what-is-middleware/#:text=Middleware>

# Anexo A

## Instalación de ROS

Este anexo se centra en la instalación de ROS Kinetic en Ubuntu 16.04 describiendo detalladamente el proceso. Dicho proceso se encuentra disponible en la comunidad de ROS [23].

La instalación de ROS Kinetic fue realizada en una PC con procesador AMD A8-7410 APU con AMD Radeon R5 Graphics 2.20 GHz, además de tener instalada una memoria RAM de 8.00 GB fragmentada en dos sistemas operativos que son Windows 10 y Ubuntu 16.04 que es el sistema operativo encargado y sugerido para el uso y control de ROS en cualquiera de sus versiones documentadas.

### A.0.1. Prerrequisitos

Antes de comenzar con la instalación, es necesaria la configuración de repositorios y dependencias:

- **Repositorio de ROS:** Para proceder con la instalación es necesario crear un archivo raíz, además de obtener un paquete de autenticación de ROS. Este proceso creará un archivo llamado `ros-latest.list`, el cual contendrá dicha llave de autenticación.

- ```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```
- ```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyserver.net:80 --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Adicionalmente, se debe actualizar *Debian package index*:

- ```
$ sudo apt-get update
```

- **Instalación de librerías y herramientas de Kinetic:** Paquetería con todas las herramientas necesarias para la utilización de ROS Kinetic.

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

- **Inicialización de rosdep:** Herramienta de línea de comandos para la instalación de las dependencias del sistema.

- `$ echo "fuente /opt/ros/kinetic/setup.bash">> / .bashrc`
- `$ fuente / .bashrc`

- **Dependencias para construir paquetes:** Hasta ahora ha instalado lo que necesita para ejecutar los paquetes principales de ROS. Para crear y administrar sus propios espacios de trabajo ROS, existen varias herramientas y requisitos que se distribuyen por separado.

Para instalar esta herramienta y otras dependencias para construir paquetes ROS, se ejecuta:

- `$ sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential`

### A.0.2. Creación de Catkin Workspace

Después de la instalación de las dependencias necesarias, se puede proceder con la instalación. La instalación está basada en la compilación de los núcleos o paquetes esenciales por lo que se necesita un *Catkin Workspace*:

- `$ mkdir -p /ros_catkin_ws`
- `$ cd /ros_catkin_ws`

A continuación, se procede a reunir los paquetes esenciales para su compilación, añadiendo todos los paquetes `catkin` o `wet` de la variante deseada (kinetic) en el entorno de trabajo, específicamente en `/ros_catkin_ws/src`. Actualmente se ofrecen dos variantes de paquetes a instalar por defecto:

- **ROS-Comm:** ROS package, build y librerías de comunicación.
- **Desktop:** ROS Package, rqt, rviz y librerías de robótica.

La principal diferencia entre los dos paquetes es que ROS-Comm no incluye interfaces gráficas de usuario, mientras que Desktop sí las incluye. Debido al alcance del proyecto, se selecciona ROS-Comm por el tipo de aplicaciones que se van a desarrollar (Sistemas embebidos sin interfaz de usuario).

Para la instalación de las variantes de ROS-Comm, se utiliza el comando `wstool`:

- ```
$ rosinstall_generator ros_comm --rosdistro kinetic --deps --wet-only --tar > kinetic-ros_comm-wet.rosinstall
```
- ```
$ wstool init src kinetic-ros_comm-wet.rosinstall
```

En el caso de que se quiera instalar la variante Desktop:

- ```
$ rosinstall_generator desktop --rosdistro kinetic --deps --wet-only --tar > kinetic-desktop-wet.rosinstall
```
- ```
$ wstool init src kinetic-desktop-wet.rosinstall
```

### A.0.3. Compilación de Catkin Workspace

Una vez que se hayan descargado todos los paquetes necesarios y se hayan resuelto las dependencias, se puede proceder a compilar *Catkin Workspace*:

- ```
$ sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/kinetic -j2
```

## Anexo B

# Instalación de la librería ros\_lib en el entorno de Arduino

### B.0.1. Actualización de Debian package index

Como primer punto, es necesario la actualización de las paqueterías para permitir una instalación de versiones correctas y adecuadas.

- `$ sudo apt-get update`

### B.0.2. Compatibilidad ROS y Arduino

Lo siguiente es encontrar la versión Arduino compatible con nuestra versión de ROS así como sus actualizaciones.

- `$ sudo apt-get install ros-kinetic-rosserial-arduino`
- `$ sudo apt-get install ros-kinetic-rosserial`

### B.0.3. Instalación de drivers en Catkin Workspace

Dentro de nuestra carpeta de trabajo deberán quedar instalados y actualizados los drivers para establecer la comunicación entre Arduino y ROS.

- `git clone https://github.com/ros-drivers/rosserial.git`
- `catkin_make install`

#### **B.0.4. Compilación de drivers**

Para finalizar, es necesario que se ejecutar el siguiente comando para permitir que el IDE de Arduino actualize sus librerías donde se han de incluir las requeridas para la comunicación entre ROS y Arduino.

- `rm -rf ros_lib`
- `rosserial_arduino make_libraries.py .`

#### **B.0.5. Reinicio del IDE Arduino**

Por último, es requerido que el dispositivo donde se ha realizado la configuración, actualización e instalación sea reiniciado para finalizar el proceso.

## Anexo C

### Ejemplo de servocontrolador

Código para la manipulación de un servocontrolador con apoyo de ROS y Arduino UNO.

```
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include <WProgram.h>
#endif

#include <Servo.h>
#include <ros.h>
#include <std_msgs/UInt16.h>

ros::NodeHandle nh;

Servo servo;

void servo_cb( const std_msgs::UInt16& cmd_msg){
    servo.write(cmd_msg.data); //set servo angle,
    //should be from 0-180
    digitalWrite(13, HIGH-digitalRead(13)); //toggle led
}

ros::Subscriber<std_msgs::UInt16> sub("servo", servo_cb);

void setup(){
```

```
pinMode(13, OUTPUT);  
  
nh.initNode();  
nh.subscribe(sub);  
  
servo.attach(9); //attach it to pin 9  
}  
  
void loop(){  
nh.spinOnce();  
delay(1);  
}
```

## Anexo D

### Ejemplo de blink

Código para la manipulación del led incluido en la placa Arduino UNO con apoyo de ROS.

```
#include <ros.h>
#include <std_msgs/Empty.h>

ros::NodeHandle nh;

void messageCb( const std_msgs::Empty& toggle_msg){
digitalWrite(13, HIGH-digitalRead(13)); // blink the led
}

ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb);

void setup()
{
pinMode(13, OUTPUT);
nh.initNode();
nh.subscribe(sub);
}

void loop()
{
nh.spinOnce();
delay(1);
}
```

## Anexo E

# Control de dispositivo autónomo

El siguiente algoritmo tiene la funcionalidad principal de enviar y publicar las órdenes deseadas a ROS, es decir, es el encargado de comunicarle a ROS cuales son las funciones que debe emitir para que Arduino UNO las ejecute de forma inmediata.

Es importante mencionar que es un algoritmo hecho en cpp y compilado dentro de la carpeta **catkin\_ws** creada al momento de la instalación de **ROS**.

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <iostream>
#include <sstream>
using namespace std;

int main(int argc, char **argv)
{
    /**
     * La función ros::init() es necesario ejecutarla
     en primer lugar si deseamos interactuar con el sistema ROS
    */

    ros::init(argc, argv, "car_control_pub");

    /**
     * Declaramos un nodehandle denominado nh, encargado
     de gobernar la comunicación.
    */

    ros::NodeHandle nh;
```

```

/**
* Definimos un publicador de dicha comunicaci\'on as\'i
como el tipo de mensaje que env\'ia .
*/
ros::Publisher pub = nh.advertise<std_msgs::String>("car_control"
,1, true);
ros::Rate loop_rate(10);

/**
* Cuenta del n\'umero de mensajes enviados .
*/
int count = 0;
bool inicio=true; //booleano para determinar el mensaje de
espera de subscriptores
char parar= 'N'; // char para opcion de detener la comunicaci\'on
bool cya_subscribers=false; // para detener cuando me quedo
sin subscriptores

while ( ros::ok() && !cya_subscribers )
{
 /**
* cuerpo del mensaje que publicar\'a pub en el topic escogido .
*/
std_msgs::String msg;

// Rellenamos el mensaje
std::string ss;

/*Enviamos los diferentes comandos*/
cout <<" Opciones de control: \n\t--> arranca\n\t-->
acelera \n\t--> reduce\n\t--> stop \n\t--> salir"
<<endl;
cout <<" Introduzca un comando: ";
cin >> ss;

```

```

if ( ss . compare( " salir " )!=0){
msg . data = ss . data ();

/*
* Publicamos el mensaje en el topic .
*/



while ( pub . getNumSubscribers ()<1){
// esperamos subscriptores , en caso de desaparecer
a media ejecucion se da la opcion
// de terminar . Poner 2 si usamos el Scada de Arduino UNO,
2 si no lo estamos usando .
if ( inicio ) {
cout << " Esperando Subscriptores ." << endl ;
inicio =false ;
}
cout << " El subscriptor ha perdido el socket de comunicacion .
\n Se encuentra bloqueado buscandolo . \n Desea parar ( S/N ) " << endl ;
cin >> parar ;
if ( parar =='s' || parar =='S' ) {
cya_subscribers=true ;
system ( " rosnode kill car_control_sub " );
system ( " rosnode kill car_feedback_pub " );
pub . shutdown ();
break ;
}
}
if ( ! cya_subscribers ) pub . publish ( msg );

ros :: spinOnce ();

loop_rate . sleep ();

count++;
if ( count==3)
count=0;
}// fin if
else {
ROS_WARN ( " INICIADO PROTOCOLO DE FINALIZACION " );
}

```

```
ROS_WARN(" CERRANDO EL SISTEMA . . . ");
system(" rosnode kill car_control_sub");
system (" rosnode kill car_feedback_pub");
msg.data = "salir";
pub.publish(msg);
ros::spinOnce();
loop_rate.sleep();
ROS_WARN("FINALIZACION REALIZADA . . . ");
pub.shutdown();
break;
}
} // fin while ros.ok

return 0;
}
```

## Anexo F

# Comunicación Bluetooth

A lo largo del documento se ha hecho mención de la comunicación que se realiza entre Arduino UNO y ROS, en este anexo se muestra el algoritmo encargado de implementar esta comunicación mediante los módulos Bluetooth HC-05 y HC-06. Este procedimiento esta dividido en cuatro: Master, Slave, Control y Orden.

### F.0.1. Algoritmo Master

El primer algoritmo, Master, se encarga de la configuración del módulo Bluetooth por medio de comandos AT desde el monitor serial del IDE Arduino

La implementación de este algoritmo fue hecha en un módulo Bluetooth HC-05.

Utilizando una conexión serie de hardware para la comunicación con la computadora Host y una conexión serie de software para la comunicación con el módulo Bluetooth Slave.

Las conexiones de los pines son:

- Salida BT VCC a Arduino 5V.
- BT GND a GND.
- BT RX al pin 3 de Arduino.
- BT TX al pin 2 de Arduino

Cuando se ingresa un comando en el monitor serial en la computadora el Arduino lo transmitirá al módulo Bluetooth receptor y mostrará el resultado.

```
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(2, 3); // RX | TX
```

```

void setup()
{
Serial.begin(9600);
Serial.println("Arduino with HC-06 is ready");

// HC-06 default baud rate is 9600
BTSerial.begin(9600);
Serial.println("BTserial started at 9600");
}

void loop()
{

// Keep reading from HC-06 and send to Arduino Serial Monitor
if (BTSerial.available())
Serial.write(BTSerial.read());

// Keep reading from Arduino Serial Monitor and send to HC-06
if (Serial.available())
BTSerial.write(Serial.read());
}

```

### F.0.2. Algoritmo Slave

Este algoritmo, como el anterior, se encarga únicamente de la configuración de los módulos Bluetooth a través de comandos AT por medio del monitor serial que tiene como herramienta el IDE Arduino.

El HC-06 pasa por defecto al modo conmutación cuando se enciende por primera vez, necesita ser colocada en modo AT para la configuración de velocidad en baudios predefinida para el modo de comunicación en 38400 baudios.

Estos módulos HC-06 requieren letras mayúsculas y sin final de línea. Las conexiones de los pines son:

- Salida BT VCC a Arduino 5V.
- BT GND a GND.
- BT RX al pin 3 de Arduino.

- BT TX al pin 2 de Arduino.

```
#include <SoftwareSerial.h>
SoftwareSerial BTserial(2, 3); // RX | TX

char c = ' ';

void setup()
{
// start th serial communication with the host computer
Serial.begin(9600);
Serial.println("Arduino with HC-05 is ready");

// start communication with the HC-05 using 38400
BTserial.begin(38400);
Serial.println("BTserial started at 38400");
}

void loop()
{

// Keep reading from HC-05 and send to Arduino Serial Monitor
if (BTserial.available())
{
c = BTserial.read();
Serial.write(c);
}

// Keep reading from Arduino Serial Monitor and send to HC-05
if (Serial.available())
{
c = Serial.read();

// mirror the commands back to the serial monitor
// makes it easy to follow the commands
Serial.write(c);
BTserial.write(c);
}
```

```
}
```

### F.0.3. Algoritmo Control

El siguiente algoritmo se implementa en la placa con el módulo Maestro (HC-05). Este algoritmo establece la comunicación entre ROS, Rosserial y Arduino, permitiendo el envío de caracteres desde bluetooth maestro con ROS hacia el bluetooth esclavo.

```
#if (ARDUINO >= 100)
#include <Arduino.h>
#else
#include <WProgram.h>
#endif

#include <ros.h>
#include <std_msgs/String.h>

#include <SoftwareSerial.h>

ros::NodeHandle nh;
SoftwareSerial BTSerial(2, 3);

void comandos( const std_msgs::String& comando){
String C="";
C=comando.data;

if(C.equals("arranca"))
{
digitalWrite(13,HIGH);
BTSerial.write('1');
BTSerial.flush();
} else
if(C.equals("acelera"))
{
digitalWrite(13,LOW);
BTSerial.write('2');
BTSerial.flush();
}
```

```

}

ros :: Subscriber<std_msgs::String> sub("car_control", comandos);

void setup(){

pinMode(13,OUTPUT);
digitalWrite(13,LOW);

BTSerial.begin(9600);
nh.initNode();
nh.subscribe(sub);
}

void loop(){
nh.spinOnce();
delay(1);
}

```

#### F.0.4. Algoritmo Orden

Como última implementación se encuentra el algoritmo cargado en la placa con el bluetooth esclavo. En este se realiza la recepción de los caracteres que le indican el encendido de los servomotores para que el dispositivo comience el funcionamiento dependiendo de cada una de las órdenes emitidas por el algoritmo anterior.

```

#include <SoftwareSerial.h>

SoftwareSerial BTSerial(2, 3); // RX | TX
int m1 = 4;
int m2 = 5;

void setup()
{
BTSerial.begin(9600);

pinMode(13,OUTPUT);
pinMode(m1,OUTPUT);

```

```
pinMode(m2,OUTPUT);

void loop()
{
char c = BTSerial.read();
if(c == '1'){
digitalWrite(13,HIGH);

digitalWrite(m1,HIGH);
digitalWrite(m2,HIGH);

BTSerial.flush();

}

if(c == '2'){
digitalWrite(13,LOW);

digitalWrite(m1,LOW);
digitalWrite(m2,LOW);

BTSerial.flush();

}
}
```