



Introducción a ROS en Raspberry Pi

Javier Gutiérrez Pérez

Máster Universitario Ingeniería de Telecomunicación
TFM Electrónica

Nombre Consultor/a: Aleix López Antón

Nombre Profesor/a responsable de la asignatura: Carlos Monzo

07/06/2017



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Copyright © 2017-Javier Gutiérrez Pérez

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

© (Javier Gutiérrez Pérez)

Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Introducción a ROS en Raspberry Pi</i>
Nombre del autor:	<i>Javier Gutiérrez Pérez</i>
Nombre del consultor/a:	Aleix López Antón
Nombre del PRA:	<i>Carlos Monzo</i>
Fecha de entrega (mm/aaaa):	06/2017
Titulación:	<i>Máster Universitario Ingeniería de Telecomunicación</i>
Área del Trabajo Final:	<i>TFM Electrónica</i>
Idioma del trabajo:	<i>Español</i>
Palabras clave	<i>ROS, Robótica, Sistema Operativo</i>
Resumen del Trabajo (máximo 250 palabras): <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i>	
<p>Se presenta un marco introductorio al sistema operativo ROS dirigido al público interesado en robótica usando Raspberry Pi. Debido a que la comunidad de la robótica ha realizado grandes progresos en los últimos años, es pertinente realizar un estudio sobre dicha plataforma de software utilizada en el control de los robots.</p> <p>Para tales efectos, el proyecto aborda tres cuestiones principales. Primero, se presenta una visión general del papel de ROS en el mundo de la robótica, ya que este sistema operativo no sustituye a los OS tradicionales, sino que trabaja conjuntamente con ellos. Asimismo, se especifica qué es ROS, cuáles son sus ventajas, cuando se usa y qué podemos esperar de este sistema operativo. Segundo, se procede al estudio en detalle de este sistema operativo. Para ello, se describe en profundidad su funcionamiento, estructura y forma de trabajar. A fin de que este estudio pueda servir como base para futuros proyectos en robótica, se realiza un montaje de ROS en Raspberry Pi. Esto se lleva a cabo mediante una ilustración detallada de ROS, una guía de instalación del sistema operativo en Raspberry Pi y una serie de ejemplos básicos para comprobar el correcto funcionamiento de ROS en la plataforma Raspberry Pi. Tercero, para aumentar el impacto de este proyecto, se procede al diseño de un sistema de comunicación basado en SPI para aumentar la interoperabilidad de este</p>	

proyecto. Posteriormente, se describe el protocolo utilizado, así como su implementación y despliegue en la plataforma descrita.

Abstract (in English, 250 words or less):

This paper presents an introductory framework to the ROS operating system aimed at the general public with an interest in robotics using Raspberry Pi. Given that the robotics community has made great progress in recent years, it is pertinent to carry out a study of this software platform used to control robots.

To this end, the project addresses three main issues. First, it presents an overview of ROS' role in the world of robotics, since this operating system does not replace traditional OS, but works with them. It also specifies what ROS is, its advantages, when it is used and what we can expect from this operating system. Second, we proceed to the detailed study of this operating system. For this, its operation, structure and way of working is described in depth. In order for this study to serve as a basis for future robotic projects, a ROS assembly is performed on Raspberry Pi. This is done through a detailed ROS illustration, an operating system installation guide in Raspberry Pi and a series of basic examples to verify the correct operation of ROS on the Raspberry Pi platform. Third, to increase the impact of this project, we proceed to design a communication system based on SPI to increase the interoperability of this project. Subsequently, the protocol used is described, as well as its implementation and deployment in the described platform.

Índice

1. Introducción	1
1.1 Contexto y justificación del Trabajo	1
1.2 Objetivos del Trabajo	1
1.3 Enfoque y método seguido	1
1.4 Planificación del Trabajo	2
1.5 Breve descripción de los otros capítulos de la memoria	2
2. Estado del arte	3
2.1 Sistemas operativos compatibles con ROS	3
2.2 Distribuciones de ROS	5
2.3 Sistemas empotrados adecuados para ROS	6
2.4 Protocolos de comunicación entre sistemas empotrados	9
3. Introducción a ROS	11
3.1 ¿Qué es ROS y por qué debe utilizarse?	11
3.2 Historia	12
4. ROS en Raspberry Pi	13
4.1 Raspberry Pi	13
4.1.1 Hardware	13
4.1.2 Software	14
4.2 Instalación de ROS en Raspberry Pi	15
4.2.1 Requisitos de instalación	15
4.2.2 Instalación de Raspbian Jessie	16
4.2.3 Instalación de ROS	17
4.2.3.1 Prerrequisitos	17
4.2.3.2 Creación de Catkin Workspace	18
4.2.3.3 Resolver dependencias	19
4.2.3.4 Compilación de Catkin Workspace	20
5. ROS en profundidad	21
5.1 Descripción de Filesystem level	21
5.2 Packages	22
5.2.1 Meta Packages	24
5.2.2 Messages	24
5.2.3 Services	25
5.3 Descripción de Computation Graph level	25
5.3.1 Nodes	27
5.3.2 Topics	28
5.3.3 Services	29
5.3.4 Messages	29
5.3.5 Bags	30
5.3.6 Master	30
5.4 Descripción de Community Level	30
5.5 Tutoriales de ROS en Raspberry Pi	31
5.5.1 Instalación de tutoriales	31
5.5.2 Tutorial - Navigating the ROS Filesystem	32
5.5.3 Tutorial - Understanding ROS Nodes	33
5.5.4 Tutorial - Understanding ROS Topics	37
5.5.5 Tutorial - Understanding ROS Services	40

6. Protocolo de comunicación en Raspberry Pi + ROS	42
6.1 Arquitectura de la aplicación	43
6.2 Implementación básica de robot.....	45
6.2.1 Solución adoptada.....	45
6.2.2 Preparación del entorno	46
6.2.2.1 Nodo controlador	47
6.2.2.2 Nodo master_spi.....	49
6.2.3 Compilación y ejecutables	52
6.2.4 Prueba de funcionamiento	52
7. Conclusiones	55
8. Bibliografía	56

Lista de figuras

Ilustración 1. Entregables (wbs) a nivel de producto	2
Ilustración 2. Logo Ubuntu.....	3
Ilustración 3 .Logo Debian.....	3
Ilustración 4. Logo HomeBrew	4
Ilustración 5. Logo Gentoo	4
Ilustración 6. Logo OpenEmbedded	4
Ilustración 7. Logo Android NDK	4
Ilustración 8. Logo ROS Lunar.....	5
Ilustración 9. Logo ROS Kinetic.....	5
Ilustración 10. Logo ROS ade	5
Ilustración 11. Logo ROS Indigo.....	6
Ilustración 12. Placa Raspberry Pi 3.....	6
Ilustración 13. Placa BeagleBone Blue	7
Ilustración 14. Placa ODROID-XU4	8
Ilustración 15. Placa Jetson TK1	8
Ilustración 16. Comunicación SPI	9
Ilustración 17. Comunicación I2C	10
Ilustración 18. Comunicación UART	10
Ilustración 19. Diagrama de Pines Raspberry Pi 3.....	14
Ilustración 20. SDCARD	15
Ilustración 21. Imagen Raspbian Jessie.....	15
Ilustración 22. Wind32 Disk Manager	15
Ilustración 23. Versión Raspbian Jessie	16
Ilustración 24. Escritura con Win32 Disk Manager	16
Ilustración 25. Filesystem Level ROS	22
Ilustración 26. Datos estándar ROS	25
Ilustración 27. Computation Level ROS	26
Ilustración 28. Comando rospak	32
Ilustración 29. Comando roscd	33
Ilustración 30. Comando rosls	33
Ilustración 31. roscore	34
Ilustración 32. lista de nodos, rosnode list	34
Ilustración 33. Información de nodos, rosout.....	35
Ilustración 34. Nodo tortuga	35
Ilustración 35. Lista de nodos. rosnode list.....	36
Ilustración 36. Ping a nodos	36
Ilustración 37. Ejemplo turtle_teleop_key	38
Ilustración 38. Información nodo teleop_turtle	39
Ilustración 39. Información nodo turtlesim	39
Ilustración 40. Lista de tópicos.....	40
Ilustración 41. Información topico cmd_vel.....	40
Ilustración 42. Lista de servicios.....	41
Ilustración 43. Lista de parámetros del servicio	41
Ilustración 44. Creación de nueva tortuga	42
Ilustración 45. Diagrama de Robot basado en ROS	42
Ilustración 46. Diagrama de nodos con tópicos	44

Ilustración 47. Diagrama de nodos con servicios	44
Ilustración 48. Montaje de Robot (Raspberry Pi 3 + LED)	46
Ilustración 49. Comunicación entre nodo controlador y nodo master_spi	53
Ilustración 50. Lista de nodos en robot.....	54
Ilustración 51. Publicación de nodos en robot.....	54

Lista de tablas

Tabla 1. Estructura de mensajes.....	24
Tabla 2. Conexión entre Raspberry Pi y Serial 7-seg.....	46

1. Introducción

1.1 Contexto y justificación del Trabajo

Debido a que el mundo de la robótica ha evolucionado considerablemente en los últimos años, es necesario introducirlo en la educación. En este proyecto se pretende dar una idea general del funcionamiento de ROS en Raspberry PI para su utilización en el aprendizaje y el desarrollo de personas interesadas en este tema. Así, este proyecto podrá ser utilizado tanto de una forma teórica como práctica en la introducción a la robótica, sin necesidad de conocimientos previos.

1.2 Objetivos del Trabajo

Los objetivos principales del proyecto son los siguientes:

- Descripción general de ROS y su aplicación en la robótica.
- Estudio detallado de ROS incluyendo su funcionamiento, estructura y forma de trabajar.
- Implementación de protocolo de comunicación basado en SPI usando ROS como sistema operativo.

1.3 Enfoque y método seguido

Con el objetivo de la realización de un proyecto que sea reutilizable y sirva para proporcionar conocimientos sobre el sistema operativo ROS, se procede primeramente a la descripción de dicho sistema operativo, entrando en detalle y profundizando a medida que el proyecto avanza para suministrar una guía de aprendizaje para aquellas personas que estén interesadas en la robótica. Asimismo, se detallan paso a paso las acciones a tomar para la instalación o ejecución de todas las acciones descritas en este proyecto para facilitar la reproducción de entornos y tutoriales por parte del lector.

1.4 Planificación del Trabajo

La planificación del trabajo se ha realizado a través de entregables a nivel de producto:

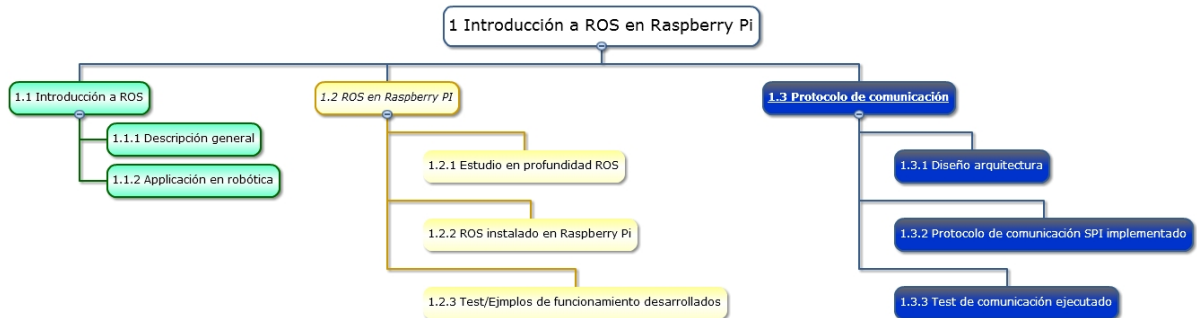


Ilustración 1. Entregables (wbs) a nivel de producto

1.5 Breve descripción de los otros capítulos de la memoria

- Capítulo 2. Estado del Arte. Conocimiento general de la tecnología relacionada o usada.
- Capítulo 3. Introducción a ROS. Descripción del sistema operativo ROS, así como las ventajas ofrecidas por el mismo.
- Capítulo 4. ROS en Raspberry Pi. Análisis de Raspberry Pi en lo que se refiere a Hardware y Software. Además, se proporciona una guía de instalación de ROS en Raspberry Pi.
- Capítulo 5. ROS en profundidad. Descripción detallada del sistema operativo ROS junto con tutoriales basado en Raspberry Pi para facilitar la comprensión del sistema operativo.
- Capítulo 6. Protocolo de comunicación en Raspberry Pi + ROS. Implementación de un protocolo de comunicación basado en SPI, para establecer la transferencia de datos entre Raspberry Pi y la placa STM32F4.

2. Estado del arte

Robot Operative System (ROS) es un *framework* usado de manera generalizada en el mundo de la robótica. Su modularidad ofrece a los usuarios la capacidad de reutilizar código en distintas plataformas sin la necesidad de grandes cambios en el código, permitiendo la transferencia de funcionalidades entre distintos robots.

2.1 Sistemas operativos compatibles con ROS

Debido a que ROS no es un sistema operativo, sino un *framework* 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:

- **Ubuntu**

Sistema operativo de código abierto 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



Ilustración 2. 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



Ilustración 3 .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.



Ilustración 4. 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 empujados. Provee sistemas de compilación e instalación, además de actualizaciones automáticas.



Ilustración 5. Logo Gentoo

- **OpenEmbedded/Yocto**

Framework de código abierto usado para la creación de distribuciones basadas en Linux, mantenido por *Yocto Project* y *OpenEmbedded Project*. Principalmente utilizado en sistemas empujados, aunque no está restringido a dichas plataformas. Está basado en distintas capas de aplicación y librerías que forman un conjunto de metadatos.



Ilustración 6. 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*).



Ilustración 7. Logo Android NDK

2.2 Distribuciones de ROS

Una distribución de ROS es un conjunto 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:

- **ROS Lunar:**

- Publicada: 23/05/2017
- Plataformas:
 - Ubuntu: Willy; Xenial
 - Debian: Stretch
- Cambios:
<http://wiki.ros.org/lunar/Migration>



Ilustración 8. Logo ROS Lunar

- **ROS Kinetic**

- Publicada: 23/05/2016
- Plataformas:
 - Ubuntu: Willy; Xenial
 - Debian: Jessie
 - OS X (Homebrew)
 - Gentoo
 - OpenEmbedded/Yocto
- Cambios:
<http://wiki.ros.org/kinetic/Migration>



Ilustración 9. Logo ROS Kinetic

- **ROS Jade**

- Publicada: 23/05/2015
- Plataformas:
 - Ubuntu
 - OS X (Homebrew)
 - Gentoo
 - Android (NDK)
- Cambios:
<http://wiki.ros.org/jade/Migration>



Ilustración 10. Logo ROS Jade

- **ROS Indigo**

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



Ilustración 11. Logo ROS Indigo

- Cambios:
<http://wiki.ros.org/indigo/Migration>

2.3 Sistemas empotrados adecuados para ROS

En el mercado se puede encontrar una gran variedad de placas de sistemas empotrados, aunque no todas ellas son compatibles con ROS. A continuación, se presenta una lista de las plataformas más usadas compatibles con ROS, cualquiera de las cuales permitiría cumplir los objetivos de este proyecto:

- **Raspberry Pi**

Provee las siguientes versiones compatibles con ROS:

- Raspberry Pi 3
- Raspberry Pi Zero

Cuenta con las siguientes características (Raspberry Pi 3):

- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 4 puertos USB
- 40 GPIO pins
- Salidas vídeo: Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD.
- Salidas Audio: 3.5 mm jack, HDMI.
- Almacenamiento: SD / MMC / ranura para SDIO.
- Red: 10/100 Ethernet (RJ45) via hub USB.
- Alimentación: 5V vía Micro USB o GPIO header.
- Dimensiones: 85.60mm x 53.98mm (3.370 x 2.125 inch).



Ilustración 12. Placa Raspberry Pi 3

- **Beaglebone**

Provee las siguientes versiones compatibles con ROS:

- BeagleBone Blue
- BeagleBone Black
- BeagleBoard-X15

Cuenta con las siguientes características (BeagleBone Blue):

- Octavo Systems OSD3358 1GHz ARM® Cortex-A8
- 512MB DDR3 RAM integrado
- Sistema de gestión de energía integrado
- Unidades programables a tiempo real (PRUs) 2x32-bit 200-MHz
- Almacenamiento en flash *on-board* 4GB 8-bit eMMC
- Inalámbrico: 802.11bgn, Bluetooth 4.1 y BLE
- Control motor: 8 6V servo out, 4 DC motor out, 4 codificador en cuadratura in
- Sensores: 9 axis IMU, barómetro
- Conectividad: USB 2.0 de alta velocidad cliente y servidor
- Interfaz de usuario: 11 LEDs programables por el usuario, 2 botones programables por el usuario
- Interfaces Easy Connect para agregar sensores adicionales tales como: GPS, DSM2 radio, UARTs, SPI, I2C, 1.8V analog, 3.3V GPIOs

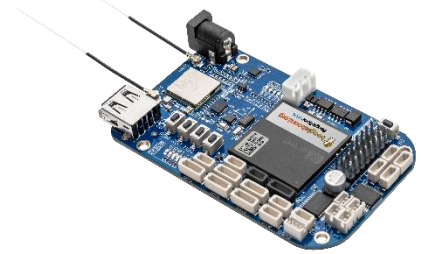


Ilustración 13. Placa BeagleBone Blue

- **Odroids**

Provee las siguientes versiones compatibles con ROS:

- ODROID-XU4
- ODROID-C2
- ODROID-C1+

Cuenta con las siguientes características (ODROID-XU4):

- Cortex™-A15 2Ghz y Cortex™-A7 Octa core CPUs
- 2Gbyte LPDDR3 RAM PoP *stacked*
- Almacenamiento en flash eMMC5.0 HS400
- 2 x USB 3.0 Host, 1 x USB 2.0 Host
- 30 GPIO *pins*
- Puerto Gigabit Ethernet
- HDMI 1.4a para *display*



Ilustración 14. Placa ODROID-XU4

- **Nvidia Jetson TK1**

Características de Nvidia Jetson TK1:

- NVIDIA Kepler GPU con 192 CUDA Cores
- 2 GB x16 con memoria de ancho de 64-bit
- Memoria 16 GB 4.51 eMMC
- 1 Half Mini-PCIE Slot
- 40 GPIO *pins*
- 1 conector de tamaño completo SD/MMC
- 1 puerto HDMI de tamaño completo
- 1 puerto USB 2.0, Micro AB
- 1 puerto USB 3.0, A
- 1 puerto serie RS232
- 1 ALC5639 Realtek Audio Codec con Mic
- In y Line Out
- 1 RTL8111GS Realtek GigE LAN
- 1 puerto de datos SATA



Ilustración 15. Placa Jetson TK1

2.4 Protocolos de comunicación entre sistemas empujados

A la hora de elegir los protocolos de comunicación utilizados en las plataformas compatibles con ROS, deben considerarse las limitaciones de las plataformas con las que se quiere establecer dicha comunicación. En el caso particular de este proyecto, se prevé establecer un sistema de comunicación con la plataforma STM32F4DISCOVERY. Las posibilidades de protocolos de comunicación entre circuitos integrados de baja velocidad son:

- **SPI**

Serial Peripheral Interface (SPI) es una interfaz bus usada principalmente para enviar información entre microcontroladores y circuitos periféricos integrados.

En SPI, la señal de reloj (normalmente llamada CLK o SCK para Serial Clock) se genera por un dispositivo con el rol de “maestro”. Mientras que el otro dispositivo recibe el rol de “esclavo”. Siempre hay un solo maestro, pero puede haber varios esclavos. Cuando los datos se envían desde el maestro a un esclavo, se envía en una línea de datos llamada MOSI, para “Master Out / Slave In”. Si el esclavo necesita enviar una respuesta al maestro, el maestro continuará generando un número preestablecido de ciclos de reloj y el esclavo colocará los datos en una tercera línea de datos llamada MISO, para “Master In / Slave Out”.

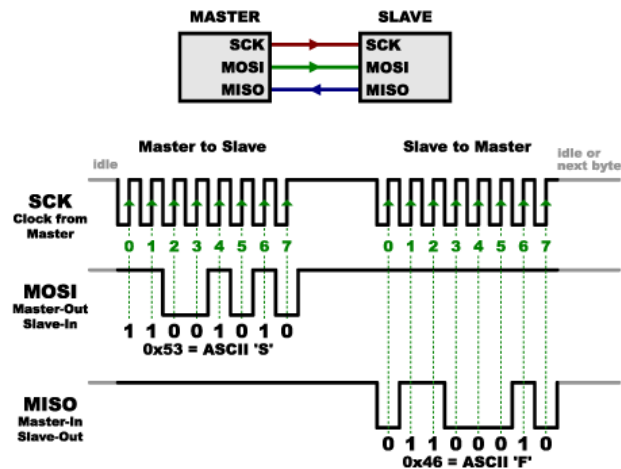


Ilustración 16. Comunicación SPI

- **I2C**

Inter-integrated Circuit (I2C) es un bus de datos principalmente utilizado para la comunicación entre microcontroladores y circuitos periféricos integrados.

EN I2C, la transferencia de datos es realizada desde un dispositivo “maestro” a otro dispositivo “esclavo”. La comunicación se realiza a través de dos conexiones, aunque permite la conexión de hasta 1008 esclavos. Además, se permite la conexión de varios elementos maestros, aunque la comunicación directa entre ellos no es posible.

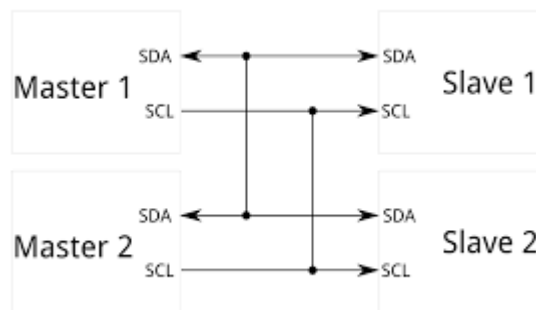


Ilustración 17. Comunicación I2C

- **UART**

Universal Asynchronous Receiver/Transmitter (UART) solo requiere de dos conexiones: una línea de transmisión (Tx) y una línea de recepción (Rx).

En el lado de transmisión, se debe de crear el paquete de datos añadiendo bits de sincronización y de paridad. En el extremo de recepción, se debe de obtener la información de acuerdo con la tasa de transmisión y seleccionar el bit de sincronización.

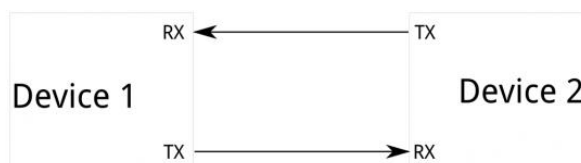


Ilustración 18. Comunicación UART

3. Introducción a ROS

3.1 ¿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, conforme a su página oficial es la siguiente:

“ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers”.¹

Como se puede observar en la definición, ROS no es un sistema operativo 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 a continuación:

- **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

¹ “ROS es un sistema de código abierto, meta-operativo, para su robot. Proporciona los servicios que usted esperaría de un sistema operativo, incluyendo la abstracción de hardware, el control de dispositivos de bajo nivel, la implementación de la funcionalidad comúnmente utilizada, el paso de mensajes entre procesos y la administración de paquetes. También proporciona herramientas y bibliotecas para obtener, construir, escribir y ejecutar código en varios equipos” (Traducción no oficial al español). ROS, *Introduction*, <http://wiki.ros.org/ROS/Introduction> (última visita, mayo de 2017).

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 al 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.

3.2 Historia

ROS forma parte de un proyecto originalmente desarrollado por la universidad de Stanford, específicamente por el laboratorio de inteligencia artificial (Stanford AI Robot) en colaboración con Personal Robots (PR) para la creación de un sistema operativo flexible y dinámico. En 2007, bajo el amparo del instituto de investigación Willow Garage y numerosos investigadores se originó la primera versión del núcleo y módulos de software de ROS con licencia de código abierto.

Desde el comienzo, el desarrollo de ROS no fue centralizado, ya que fue desarrollado por distintas instituciones y para distintos robots. Cada desarrollador puede desarrollar nuevas librerías para ROS y compartirlas con los demás miembros de la comunidad sin la necesidad de permisos de publicación o control alguno por parte de la comunidad. Esto permite un desarrollo mucho más dinámico, la capacidad de recibir retroalimentación y mejorar las librerías aportadas.

Actualmente, la comunidad ROS está formada por miles de usuarios, teniendo un amplio abanico de proyectos que van desde proyectos de aficionados hasta sistemas complejos de automatización industrial.

4. ROS en Raspberry Pi

4.1 Raspberry Pi

4.1.1 Hardware

Ante la necesidad de seleccionar un sistema empotrado capaz de soportar ROS con un bajo coste, un bajo consumo y facilitar el aprendizaje, la familia de sistemas empotrados Raspberry Pi es una de las mejores elecciones que se puede realizar. Raspberry Pi ofrece las siguientes placas compatibles con ROS:

- Raspberry Pi 3
- Raspberry Pi Zero
- Raspberry Pi Zero W

La placa seleccionada para la realización de este proyecto es Raspberry Pi 3 con las siguientes características técnicas:

- Procesador a 1,2 GHz de 64 bits con cuatro núcleos ARMv8
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- puertos USB
- 40 pines GPIO
- Puerto Full HDMI
- Puerto Ethernet
- Conector combo compuesto de audio y vídeo de 3,5 mm
- Interfaz de la cámara (CSI)
- Interfaz de pantalla (DSI)
- Ranura para tarjetas microSD (push-pull en lugar de push-push)
- Núcleo de gráficos VideoCore IV 3D
- Dimensiones de placa de 8.5 por 5.3 cm

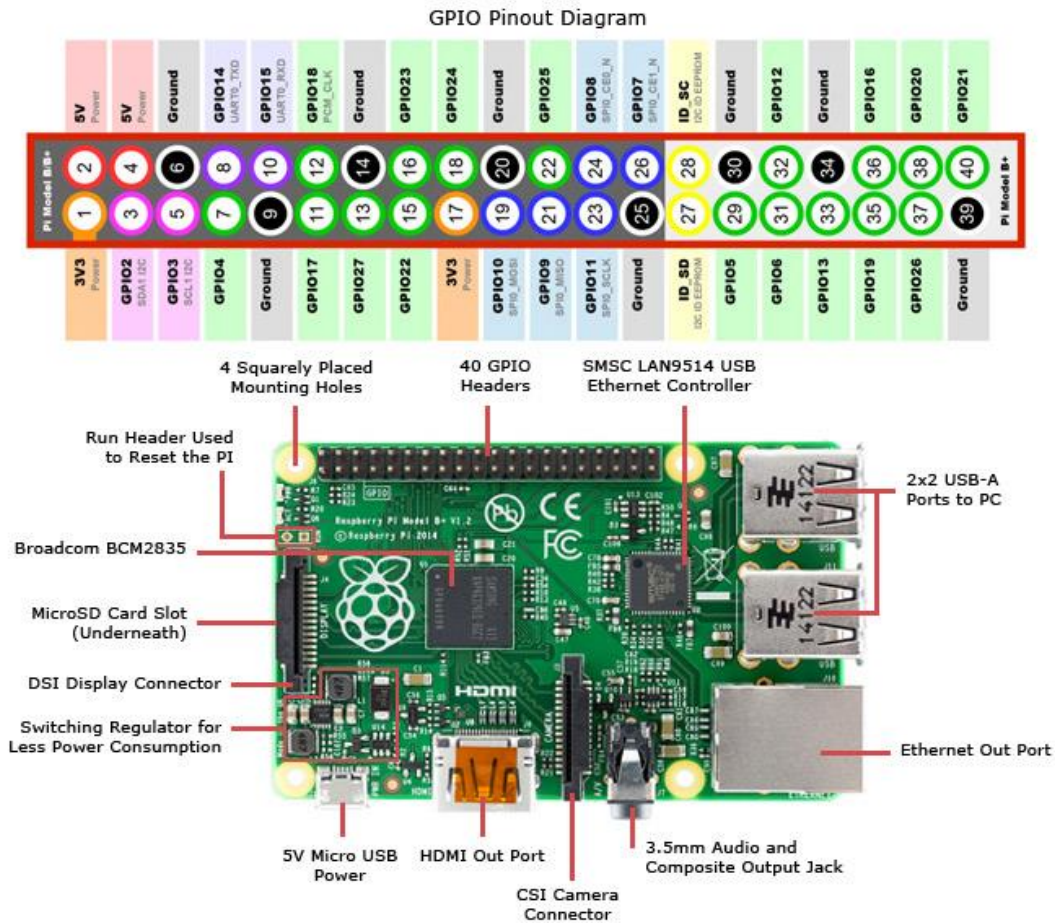


Ilustración 19. Diagrama de Pines Raspberry Pi 3

4.1.2 Software

Raspberry Pi tiene compatibilidad con los siguientes sistemas operativos:

- Noobs
- Raspbian
- Ubuntu Mate
- Windows 10
- OSMC

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 índigo, solo el sistema operativo *Raspbian Jessie* y *Ubuntu Mate 16.04* son compatibles con dicha distribución². En este caso se ha seleccionado *Raspbian Jessie* por ser el

² Installing ROS Kinetic on the Raspberry Pi, <http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi> (última visita, mayo de 2017).

sistema operativo oficial de Raspberry Pi y tener la mayor cantidad de documentación proporcionada por ROS.

4.2 Instalación de ROS en Raspberry Pi

4.2.1 Requisitos de instalación

- **SDCARD**

Raspberry Pi utiliza como sistema de almacenamiento permanente tarjetas de memoria del tipo SDCARD. Se ha utilizado una tarjeta SanDisk micro de 32 GB para este propósito.



Ilustración 20.
SDCARD

- **Imagen Raspbian Jessie**

Sistema operativo sobre el cual se realizará la instalación de ROS. La *Imagen* se puede obtener de la página oficial de Raspberry³.



Ilustración 21.
Imagen Raspbian
Jessie

- **Win32 Disk Imager**

Software gratuito destinado a copiar/grabar imágenes en USB o SDCARD. Es necesario para copiar/instalar la imagen del sistema operativo *Raspbian Jessie* en la tarjeta SD.



Ilustración 22.
Win32
Disk
Manager

³ Jessie Raspbian, <https://www.raspberrypi.org/downloads/raspbian/> (última visita, mayo de 2017).

4.2.2 Instalación de Raspbian Jessie

La instalación del sistema operativo comienza por la descarga del mismo. Como ya se ha comentado anteriormente, se puede encontrar la última versión oficial de Raspbian Jessie en la página de descargas de Raspberry⁴, además de una guía de instalación⁵. En nuestro caso:

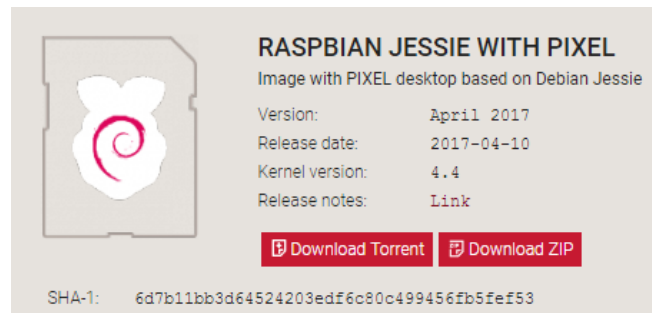


Ilustración 23. Versión Raspbian Jessie

Será necesario el uso de una herramienta para instalar imágenes en SD, en nuestro caso *Win32 Disk Imager*. A la hora de proceder con la instalación, se debe seleccionar la imagen a instalar y el dispositivo donde se va a realizar la instalación. Después de haber seleccionado la imagen y su destino, se procede a su instalación:

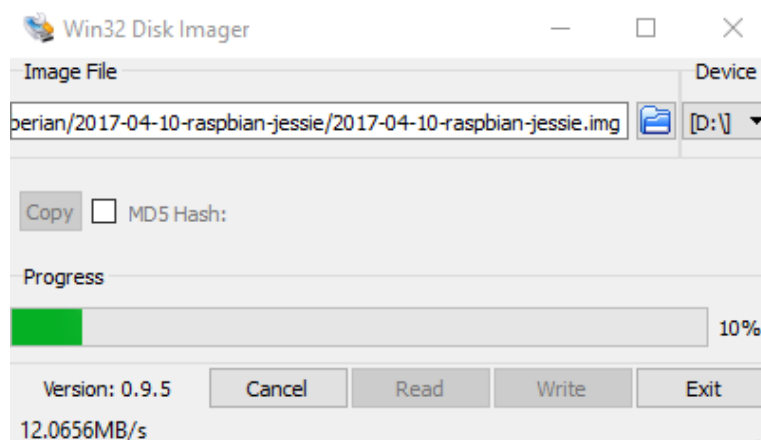


Ilustración 24. Escritura con Win32 Disk Manager

Al finalizar, se retira la tarjeta SD del ordenador donde se ha realizado la instalación y se introduce en el espacio designada para tarjetas SD de Raspberry Pi.

⁴ Jessie Raspbian, <https://www.raspberrypi.org/downloads/raspbian/> (última visita, mayo de 2017).

⁵ Installation Guide, <https://www.raspberrypi.org/documentation/installation/installing-images/README.md> (última visita, mayo de 2017).

4.2.3 Instalación de ROS

Este capítulo se centra en la instalación de ROS Kinetic en Raspbian Jessie describiendo detalladamente el proceso. Dicho proceso se encuentra disponible en la comunidad de ROS⁶.

4.2.3.1 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-keyservers.net:
80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

Adicionalmente, se debe actualizar *Debian package index*:

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

- **Dependencias Bootstrap:** Herramientas necesarias para facilitar la descarga y la gestión de los paquetes de ROS y sus dependencias, entre otras cosas.

```
$ sudo apt-get install -y python-rosdep python-rosinstall-genera
tor python-wstool python-rosinstall build-essential cmake
```

⁶ ROSberryPi/Installing ROS Kinetic on the Raspberry Pi, <http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi> (última visita, mayo de 2017).

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

```
$ sudo rosdep init
$ rosdep update
```

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

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

```
$ rosinstall_generator ros_comm --rostdistro 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
```

4.2.3.3 Resolver dependencias

Antes de que se pueda compilar *Catkin Workspace*, es necesario que las dependencias requeridas estén disponibles. Para llevar a cabo esta tarea, se usa la herramienta `rosdep`; sin embargo, si alguna de las dependencias no está disponible, deben de ser instaladas manualmente.

Para el caso de Kinetic en Raspberry Pi, solo se necesita la instalación manual de *Assimp* (*Open Asset Import Library*):

```
mkdir -p ~/ros_catkin_ws/external_src  
cd ~/ros_catkin_ws/external_src  
wget http://sourceforge.net/projects/assimp/files/assimp-3.1/assimp-3.  
1.1_no_test_models.zip/download -O assimp-3.1.1_no_test_models.zip  
unzip assimp-3.1.1_no_test_models.zip  
cd assimp-3.1.1  
cmake.  
make  
sudo make install
```

Para resolver el resto de dependencias se debe de usar `rosdep`:

```
$ cd ~/ros_catkin_ws  
$ rosdep install -y --from-paths src --ignore-src --rosdistro kinetic  
-r --os=debian:jessie
```

Esta herramienta se encarga de comprobar las dependencias necesarias de todos los paquetes e instala recursivamente las dependencias.

4.2.3.4 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
```

Es posible que durante la compilación se produzca un error como “virtual memory exhausted: Cannot allocate memory.” Esto se debe a que el sistema termina el proceso de compilación debido a la falta de memoria RAM o SWAP. Para mitigar este error, es necesario aumentar el tamaño de la memoria SWAP durante el proceso de compilación.

En el caso de esta instalación, se realizará una ampliación momentánea de 3072 durante la instalación.

```
sudo dd if=/dev/zero of=swapfile bs=1M count=3072
sudo mkswap swapfile
sudo swapon swapfile
```

Después de realizar la ampliación de la memoria SWAP, 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
```

Si el resultado de la instalación es satisfactorio (sin errores), se deshabilita y elimina la memoria SWAP:

```
sudo swapoff swapfile
sudo rm swapfile
```

Adicionalmente, se debe de copiar el archivo `setup.bash` en `~/.bashrc`, para que las variables de entorno de ROS estén disponibles automáticamente:

```
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```

5. ROS en profundidad

Una vez que la instalación se ha realizado con éxito, se puede proceder a realizar un estudio de ROS más detallado. En este capítulo, se describe la arquitectura de ROS, además de proveer una introducción a sus principales conceptos usando ejemplos basados en TurtleSim⁷.

La arquitectura de ROS se puede dividir en tres niveles de conceptos⁸:

- **Filesystem level:** Grupo de conceptos que son usados para explicar cómo está formado ROS, estructura de carpetas y archivos necesarios para funcionar.
- **Computation Graph level:** Lugar donde sucede la comunicación entre procesos y sistemas. Grupo de conceptos para configurar sistemas, manejar procesos y los sistemas de comunicación usados.
- **Community level:** Grupo de herramientas y conceptos usados para compartir conocimientos, algoritmos y código de cualquier desarrollador.

5.1 Descripción de Filesystem level

ROS, como cualquier sistema operativo, está dividido en carpetas, y cada una de esas carpetas tiene archivos que definen sus funcionalidades:⁹

⁷ TurtleSim, <http://wiki.ros.org/turtlesim> (última visita, mayo de 2017).

⁸ ROS concepts, <http://wiki.ros.org/ROS/Concepts> (última visita, mayo de 2017).

⁹ Lentin Joseph, *Mastering ROS for robotics Programming* (Packt Publishing, 2015).

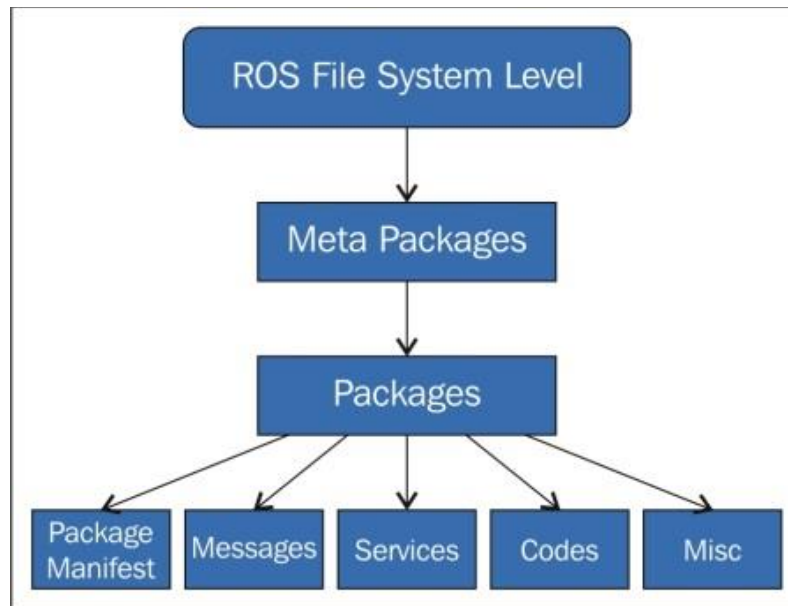


Ilustración 25. Filesystem Level ROS

- **Packages:** Nivel básico de paquetes. Un paquete es la estructura y contenido mínimo necesario para crear un programa en ROS.
- **Package Manifest:** Un manifiesto proporciona información relativa al paquete, información de licencias, dependencias y configuración del compilador. Esta información se puede encontrar en archivos llamados manifest.xml.
- **Meta Packages:** Cuando se juntan varios paquetes se obtiene un meta paquete.
- **Messages (msg) types:** Un mensaje es la información que un proceso envía a otro proceso. Por defecto, ROS tiene un gran número de mensajes estandarizados.
- **Service (svr) types:** Define las estructuras de petición y respuesta usadas por los servicios en ROS.

5.2 Packages

Un paquete contiene habitualmente una estructura formada por archivos y carpetas, normalmente compuestas por los siguientes elementos¹⁰:

¹⁰ Packages ROS, <http://wiki.ros.org/Packages> (última visita, mayo de 2017).

- **bin/**: Carpeta donde los programas son almacenados después de su compilación.
- **include/package_name/**: Carpeta donde se encuentran las cabeceras (.h) de las librerías usadas.
- **msg/**: Carpeta donde se encuentran los mensajes estándar y los creados.
- **src/package_name/**: Carpeta donde se aloja el código fuente del programa.
- **srv/**: Carpeta que contiene los tipos de servicios.
- **scripts/**: Carpeta que contiene scripts.
- **CMakeLists.txt**: Archivo para compilar con CMake.

Además, ROS provee herramientas para facilitar la creación, modificación y manejos de los paquetes:

- **rospack**: Comando para buscar u obtener información de paquetes.
- **roscrcat**: Comando para crear nuevos paquetes.
- **rosmake**: Comando para compilar un paquete.
- **rosclean**: Comando para instalar dependencias necesitadas por el paquete.
- **rosclean**: Comando para mostrar dependencias usadas por el paquete.

Asimismo, ROS proporciona una serie de comandos adicionales para moverse entre paquetes y entre sus carpetas y archivos que se pueden encontrar dentro del paquete `rosclean`. Estos comandos son muy parecidos a sus homólogos en Linux. A continuación, podemos encontrar algunos ejemplos:

- **roscd**: Comando para cambiar de directorio.
- **roscd**: Comando para editar archivos.
- **roscd**: Comando para copiar archivos.

- **rosls**: Comando para listar archivos de un paquete.

5.2.1 Meta Packages

Los paquetes en ROS están organizados en Meta-Paquetes (*Meta-Packages*). Mientras que el objetivo de un paquete es crear pequeñas colecciones de código para su reusabilidad, los Meta-Paquetes están diseñados para mejorar el proceso de compartir código.

Un Meta-Paquete es una estructura básica formada por carpetas y archivos que puede ser creado manualmente o con la ayuda de `roscrate-stack`.

5.2.2 Messages

ROS utiliza un sistema de descripción de mensajes para el valor de los datos que los nodos publican. Con este sistema de descripción, ROS puede generar el código fuente correcto para ese tipo de mensajes en diferentes lenguajes de programación.

Un mensaje está compuesto por dos partes principales: Campo con el tipo de dato que va a ser transmitido en el mensaje, por ejemplo, `uint8`, `bool` o tipos de datos creados por el usuario, constantes que definen el nombre del campo. Por ejemplo:

<i>Campo</i>	<i>Constante</i>
<i>int16</i>	time
<i>bool</i>	active
<i>float</i>	result
<i>string</i>	name

Tabla 1. Estructura de mensajes

Los tipos de datos estándar en Ros son los siguientes¹¹:

Primitive type	Serialization	C++	Python
bool	Unsigned 8-bit int	uint8_t	bool
int8	Signed 8-bit int	int8_t	int
uint8	Unsigned 8-bit int	uint8_t	int
int16	Signed 16-bit int	int16_t	int
uint16	Unsigned 16-bit int	uint16_t	int
int32	Signed 32-bit int	int32_t	int
uint32	Unsigned 32-bit int	uint32_t	int
int64	Signed 64-bit int	int64_t	long
uint64	Unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ASCII string (4-bit)	std::string	string
time	Secs/nsecs signed 32-bit ints	ros::Time	rospy. Time
duration	Secs/nsecs signed 32-bit ints	ros::Duration	rospy. Duration

Ilustración 26. Datos estándar ROS

5.2.3 Services

ROS utiliza un lenguaje de descripción de servicios simplificado para describir los tipos de servicios ofertados por ROS. Esto permite a crear un nivel sobre los tipos de mensaje estándar para habilitar un sistema de comunicación basado en petición/respuesta entre los nodos. Dichos servicios se encuentran en los archivos .srv en la carpeta srv/ de cada paquete.

Adicionalmente, ROS proporciona una herramienta rossrv que proporciona información sobre los servicios, los paquetes que contienen dicho servicio y permite localizar archivos que usan ese servicio.

5.3 Descripción de Computation Graph level

La red creada por ROS conecta todos los procesos de forma tal que cualquier nodo de esta red puede interactuar con otros nodos, obtener la información respecto a la información que envían y transmitir información a dicha red.

¹¹ Standart Types, <http://wiki.ros.org/msg> (última visita, mayo de 2017).

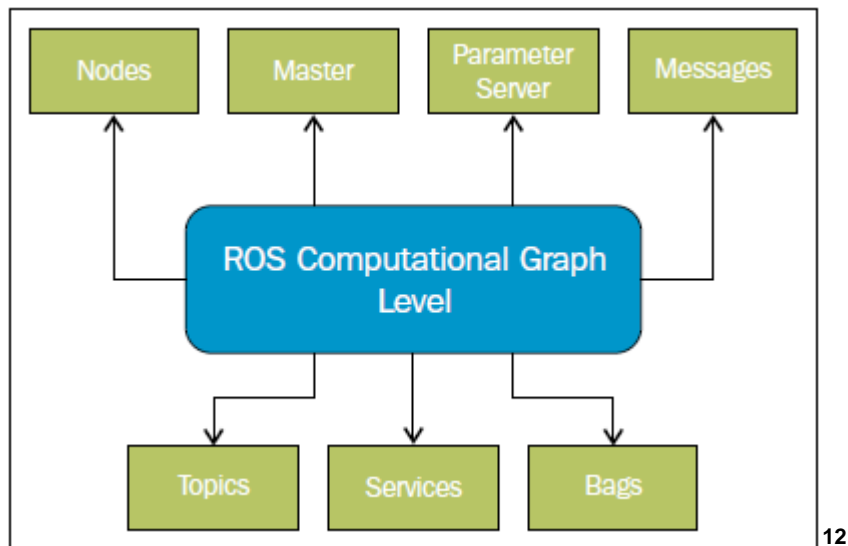


Ilustración 27. Computation Level ROS

En este nivel podemos encontrar los siguientes conceptos:

- **Nodes:** Los procesos donde se realiza el trabajo deseado en ROS se denominan Nodos. Para que un proceso pueda comunicarse con otros nodos, es necesario la creación de un nodo con este proceso y conectarlo a la red de ROS. Normalmente un sistema tendrá varios nodos para controlar distintas funciones, aunque es posible contar solo con un nodo que controle todas las funcionalidades (no recomendado).
- **Master:** Es un elemento esencial en la red de ROS que permite el registro y la búsqueda de nodos. Sin este elemento la comunicación con otros nodos, servicios y mensajes no es posible.
- **Parameter Server:** Este parámetro da la posibilidad de tener información almacenada en una ubicación central a través de llaves. Además, permite modificar la configuración de los nodos sin la necesidad de compilar.
- **Messages:** La comunicación entre nodos es realizada con mensajes. Un mensaje contiene la información que se quiere enviar a otros nodos.
- **Topics:** Cada mensaje debe de tener asignado un nombre para que pueda ser canalizado por ROS. Cuando un nodo envía información, se dice que ese nodo está publicando un tópico. Un nodo puede estar suscrito a ese tópico para recibir un mensaje enviado por otro nodo. Es muy importante que el nombre del tópico sea único para evitar problemas de comunicación con tópicos con el mismo nombre.

¹² Lentin Joseph, *Mastering ROS for robotics Programming* (Packt Publishing, 2015).

- **Services:** Cuando se publica un tópico, dicho mensaje va dirigido a varios nodos, lo cual no es lo más eficiente si se quiere hacer una petición a un nodo en específico. Este servicio otorga la posibilidad de interactuar específicamente entre ciertos nodos. De la misma manera que con los tópicos, el nombre del servicio debe ser único para evitar confusiones.
- **Bags:** ROS permite guardar y reproducir mensajes de información a través de Bags. Este es un elemento muy importante a la hora de desarrollar aplicaciones, ya que hace posible la simulación de información sin tener los elementos que la generan.

5.3.1 Nodes

Los nodos son procesos/ejecutables que pueden comunicarse con otros procesos usando tópicos y/o servicios. El uso de nodos permite tener una cierta modularidad en lo que a separación de funcionalidades se refiere, permitiendo una mayor reusabilidad de código o de los nodos en sí mismo.

El nombre de los nodos debe ser único en el sistema para permitir la comunicación con otros nodos sin tener ambigüedades. A la hora de implementar un nodo se puede utilizar distintas librerías como `roscpp` para C++ o `rospy` para Python.

ROS proporciona las siguientes herramientas para el manejo de los nodos:

- **`roscpp info node`:** Proporciona información del nodo.
- **`roscpp kill node`:** Termina la ejecución de un nodo.
- **`roscpp list`:** Proporciona una lista de los nodos activos.
- **`roscpp machine hostname`:** Proporciona una lista de los nodos activos en un ordenador en específico.
- **`roscpp ping node`:** Prueba la conectividad con el nodo.
- **`roscpp cleanup`:** Elimina la información relativa a nodos los cuales son inaccesibles.

Adicionalmente, ROS proporciona la posibilidad de cambiar parámetros/información del nodo, incluyendo el nombre del nodo, el nombre de los tópicos y nodos de los parámetros. Esta información es modificable sin necesidad de compilar el código de nuevo. Esta acción se puede llevar a cabo con `roscrun`.

5.3.2 Topics

Los tópicos son buses de datos usados por los nodos para transmitir información. Como ya se ha mencionado anteriormente, un tópico puede tener varios suscriptores, sin la necesidad de que haya una conexión directa entre los nodos ya que el consumo y la producción de información está desvinculada.

La transmisión de información se realiza usando TCP/IP y UDP. Cuando la comunicación está basada en TCP/IP se conoce como TCPROS. En cambio, cuando la comunicación está basada en UDP se conoce como UDPROS.

ROS proporciona `rostopic`, una herramienta que ofrece información sobre los nodos o la información publicada en la red de ROS. Esta herramienta se puede usar con los siguientes parámetros:

- **`rostopic bw /topic`**: Muestra el ancho de banda usado por el tópico.
- **`rostopic echo /topic`**: Muestra los mensajes de ese tópico.
- **`rostopic find messag_type`**: Busca tópicos por tipo.
- **`rostopic hz /topic`**: Muestra la frecuencia de publicación del tópico.
- **`rostopic info /topic`**: Muestra información relativa al tópico.
- **`Rostopic list`**: Muestra información de tópicos activos.
- **`rostopic type /topic`**: Muestra información del tipo del nodo y del tipo de mensajes que publica.

5.3.3 Services

Los servicios surgen de la necesidad de recibir respuesta cuando se envía un mensaje, algo que no es proporcionado por los tópicos. Los servicios deben ser desarrollados por los usuarios, ya que no existen servicios estándar. Dichos servicios deben asociarse a un tipo que es fijo para todos los servicios `srv` y un nombre que identifica al servicio.

ROS proporciona dos herramientas para trabajar con servicios `rossrv` y `rosservice`. En el caso de `rossrv`, proporciona información sobre los servicios, exactamente como hace `rosmmsg`. Mientras que `rosservice` puede mostrar y almacenar servicios, soportando los siguientes comandos:

- ***Rosservice call /service args***: Invoca el servicio con los argumentos proporcionados.
- ***Rosservice find msg-type***: Encuentra servicios por tipo.
- ***Rosservice info /service***: Muestra información del servicio.
- ***Rosservice list***: Muestra lista de servicios activos.
- ***Rosservice type /service***: Muestra el tipo de servicio.

5.3.4 Messages

Un nodo envía información a otro nodo a través de mensajes, los cuales son publicados por tópicos. El mensaje tiene una estructura básica formada por tipos de dato estándar o tipos de datos creados por el usuario.

ROS proporciona la herramienta `rosmmsg` para obtener información de los mensajes. Puede ser usada con los siguientes parámetros:

- ***rosmmsg show***: Muestra información del mensaje.
- ***rosmmsg list***: Muestra lista de mensajes.
- ***rosmmsg package***: Muestra lista de mensajes del paquete.
- ***rosmmsg packages***: Muestra paquetes que contienen mensajes.

- **rosmg users:** Muestra archivos que usan el tipo de dato message.

5.3.5 Bags

Un “bag” es un archivo creado por ROS en el que se puede guardar mensajes, tópicos y servicios, entre otros. Esta información se puede visualizar y reproducir a gusto del usuario. Para usar “bags”, ROS proporciona las siguientes herramientas:

- **rosbab:** Usado para reproducir, grabar u otras operaciones.
- **rxbag:** Usado para visualizar información.
- **rostopic:** Muestra los tópicos enviados a los nodos.

5.3.6 Master

El ROS Master proporciona el nombre y registra los servicios del resto de los nodos del sistema, además de supervisar el tráfico de tópicos y servicios. Su función principal es la de habilitar los nodos y localizarse unos a otros hasta que exista una comunicación de pares.

5.4 Descripción de *Community Level*

La comunidad de ROS está basada en distintos recursos que permiten a distintas comunidades el intercambio de software y conocimientos. Estos recursos incluyen:

- **Distribuciones:** Las distribuciones son colecciones de Meta-paquetes que se pueden instalar. Principalmente utilizadas para mantener versiones estables de software.
- **Repositorios:** ROS se basa en una red de repositorios de código, donde diferentes instituciones pueden desarrollar y aportar su propio software.
- **ROS Wiki:** ROS Wiki es el principal foro para documentación e información sobre ROS. Cualquier persona puede aportar nueva documentación, corregir errores o escribir tutoriales.
- **Lista de email:** Es el principal canal de comunicación sobre actualizaciones de ROS, así como un método de debate sobre temas relacionados con ROS.

5.5 Tutoriales de ROS en Raspberry Pi

En este capítulo, se pretende dar un ejemplo práctico de algunos de los componentes descritos anteriormente. Estos tutoriales están basados en TurtleSim¹³.

- *Navigating the ROS Filesystem*
- *Understanding ROS Nodes*
- *Understanding ROS Topics*
- *Understanding ROS Services and Parameters*

5.5.1 Instalación de tutoriales

Antes de proceder con los tutoriales, es necesario añadir el paquete `ros_tutorial`¹⁴ a nuestro entorno de trabajo, el cual ofrece un gran abanico de posibilidades a la hora de aprender con ROS.

El primer paso es la creación de una instalación de ROS que contenga los nuevos paquetes. En nuestro caso la instalación contendrá `ros_com` y `ros_tutorial`:

```
$ cd ~/ros_catkin_ws
$ rosinstall_generator ros_comm ros_tutorial --rostdistro kinetic --dep
s --wet-only --tar > kinetic-custom_ros.rosinstall
```

A continuación, se actualizará el entorno de trabajo y se instalarán o actualizarán las dependencias requeridas:

```
$ wstool merge -t src kinetic-custom_ros.rosinstall
$ wstool update -t src
$ rosdep install --from-paths src --ignore-src --rostdistro kinetic -y
-r --os=debian:jessie
```

¹³ TurtleSim, <http://wiki.ros.org/turtlesim> (última visita, mayo de 2017).

¹⁴ Tutoriales Ros, <http://wiki.ros.org/ROS/Tutorials> (última visita, mayo de 2017).

Finalmente, se procederá a la compilación del entorno de trabajo, dando por finalizada la instalación de los tutoriales:

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

5.5.2 Tutorial - Navigating the ROS Filesystem

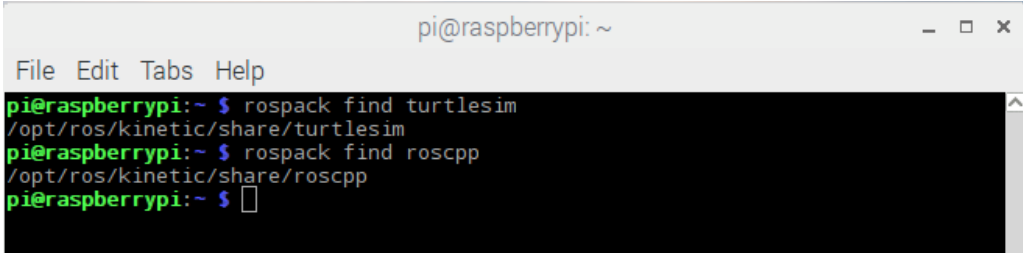
Anteriormente se han descrito varios comandos para navegar en el sistema de archivos. En este tutorial se ofrece una descripción de los siguientes comandos/herramientas:

- **rospack**

Este comando permite obtener información de los paquetes.

```
$ rospack find [package_name]
```

En el siguiente ejemplo buscaremos los paquetes `roscpp` y `turtlesim` para obtener el directorio de los paquetes:

A screenshot of a terminal window titled "pi@raspberrypi: ~". The window has a menu bar with "File", "Edit", "Tabs", and "Help". The terminal shows the following commands and output:

```
pi@raspberrypi:~ $ rospack find turtlesim
/opt/ros/kinetic/share/turtlesim
pi@raspberrypi:~ $ rospack find roscpp
/opt/ros/kinetic/share/roscpp
pi@raspberrypi:~ $
```

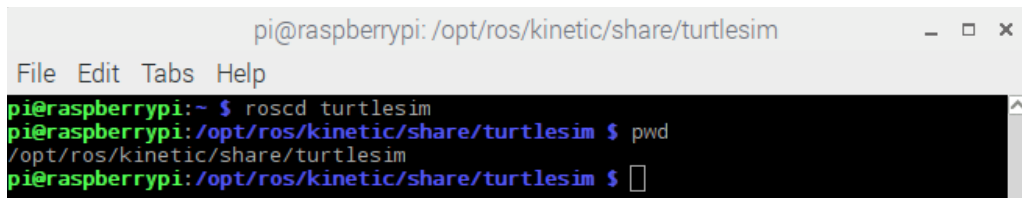
Ilustración 28. Comando `rospack`

- **roscd**

Este comando permite moverse por los directorios.

```
$ roscd [locationname[/subdir]]
```

En el siguiente ejemplo se procede al cambio de directorio del paquete seleccionado.



```
pi@raspberrypi: /opt/ros/kinetic/share/turtlesim
File Edit Tabs Help
pi@raspberrypi:~ $ roscd turtlesim
pi@raspberrypi:/opt/ros/kinetic/share/turtlesim $ pwd
/opt/ros/kinetic/share/turtlesim
pi@raspberrypi:/opt/ros/kinetic/share/turtlesim $
```

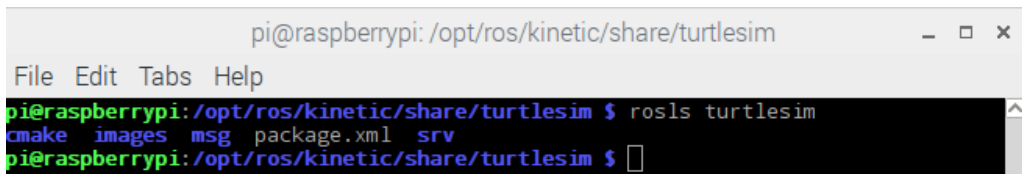
Ilustración 29. Comando roscd

- **rosls**

Este comando lista los archivos que se encuentran en un paquete.

```
$ rosls [locationname[/subdir]]
```

En el siguiente ejemplo se lista los archivos contenidos en el paquete selección.



```
pi@raspberrypi: /opt/ros/kinetic/share/turtlesim
File Edit Tabs Help
pi@raspberrypi:/opt/ros/kinetic/share/turtlesim $ rosls turtlesim
cmake images msg package.xml srv
pi@raspberrypi:/opt/ros/kinetic/share/turtlesim $
```

Ilustración 30. Comando rosls

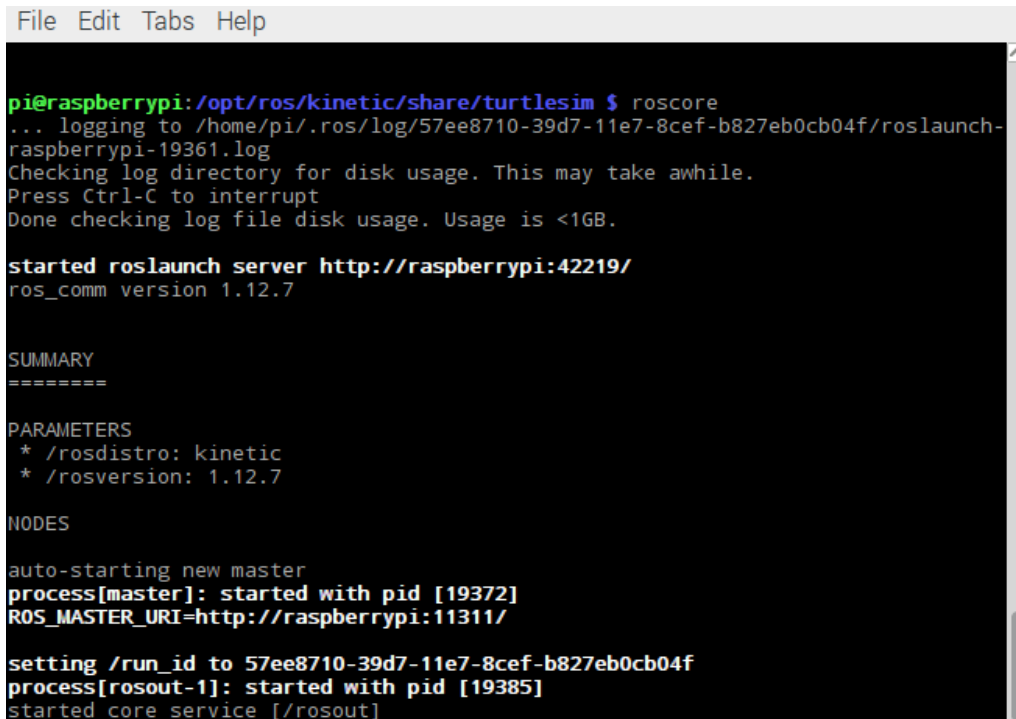
5.5.3 Tutorial - Understanding ROS Nodes

En los capítulos anteriores se ha procedido a realizar una descripción de los Nodos, el comando/herramienta usado para obtener información de los nodos es rosnode.

La primera acción que se debe de realizar cuando se está usando ROS es inicializar roscore:

```
$ roscore
```

Generando la siguiente información:



```
File Edit Tabs Help
pi@raspberrypi:/opt/ros/kinetic/share/turtlesim $ roscore
... logging to /home/pi/.ros/log/57ee8710-39d7-11e7-8cef-b827eb0cb04f/roslaunch-
raspberrypi-19361.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://raspberrypi:42219/
ros_comm version 1.12.7

SUMMARY
=====

PARAMETERS
* /rostdistro: kinetic
* /rosversion: 1.12.7

NODES

auto-starting new master
process[master]: started with pid [19372]
ROS_MASTER_URI=http://raspberrypi:11311/

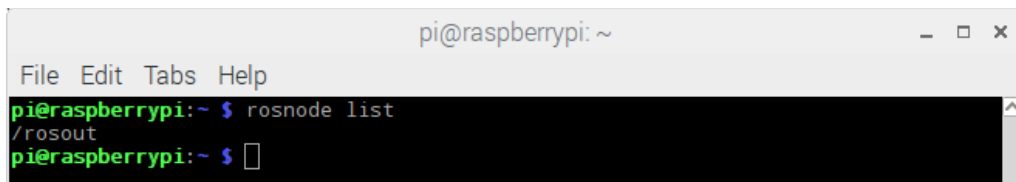
setting /run_id to 57ee8710-39d7-11e7-8cef-b827eb0cb04f
process[rosout-1]: started with pid [19385]
started core service [/rosout]
```

Ilustración 31. roscore

Una vez iniciado roscore, podemos usar el comando rosnode para obtener información de los nodos activos:

```
$ rosnode list
```

Obteniendo:



```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ rosnode list
/rosout
pi@raspberrypi:~ $
```

Ilustración 32. lista de nodos, rosnode list

Como se puede observar solo un nodo está activo, el nodo perteneciente a roscore. Para obtener información adicional de este nodo, se puede usar el siguiente comando:

```
$ rosnode info /rosout
```

Proporcionando información sobre rosout:



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ rosnode info /rosout  
-----  
Node [/rosout]  
Publications:  
* /rosout_agg [rosgraph_msgs/Log]  
  
Subscriptions:  
* /rosout [unknown type]  
  
Services:  
* /rosout/set_logger_level  
* /rosout/get_loggers  
  
contacting node http://raspberrypi:45038/ ...  
Pid: 19385
```

Ilustración 33. Información de nodos, rosout

Adicionalmente, vamos a comenzar otro nodo con rosrun. En este caso, iniciaremos un nodo perteneciente al paquete turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

Como se puede observar, una nueva ventana aparece con una tortuga:

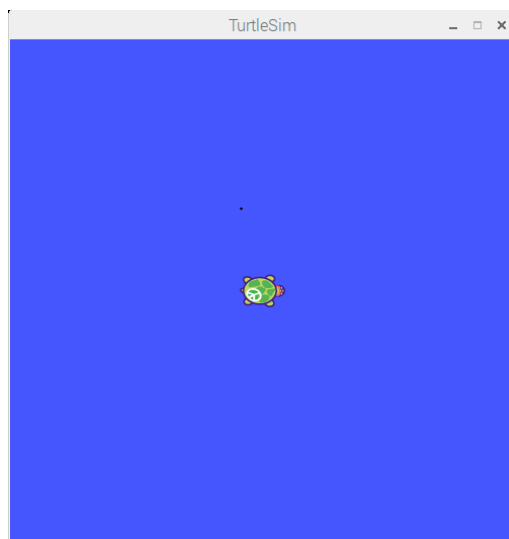
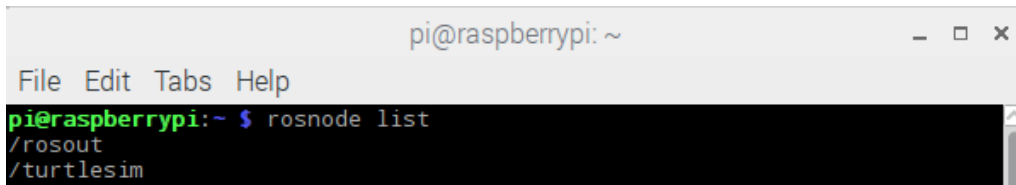


Ilustración 34. Nodo tortuga

Ahora tenemos dos nodos activos, lo cual puede comprobarse con `rostopic`:



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ rostopic list  
/rostopic  
/turtlesim
```

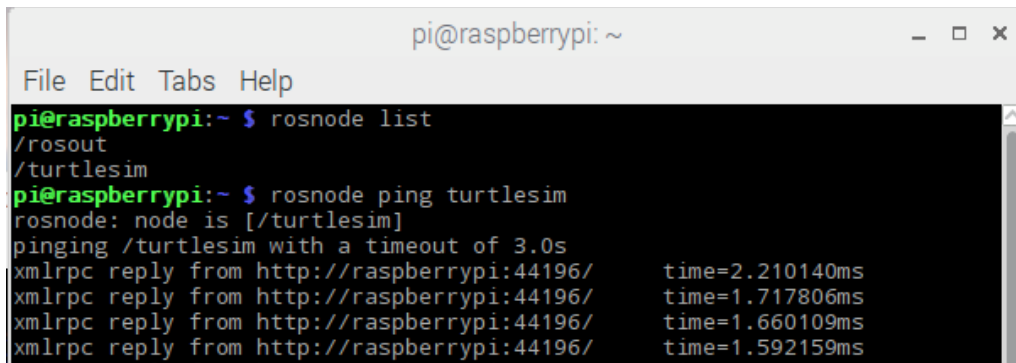
Ilustración 35. Lista de nodos. `rostopic list`

Aparte de listar nodos, `rostopic` permite cambiar el nombre de los nodos sin necesidad de compilar:

```
$ rostopic rename turtlesim turtlesim_node __name:=my_turtle
```

O comprobar la conectividad con un nodo:

```
rostopic ping turtlesim
```



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ rostopic list  
/rostopic  
/turtlesim  
pi@raspberrypi:~ $ rostopic ping turtlesim  
rostopic: node is [/turtlesim]  
pinging /turtlesim with a timeout of 3.0s  
xmlrpc reply from http://raspberrypi:44196/      time=2.210140ms  
xmlrpc reply from http://raspberrypi:44196/      time=1.717806ms  
xmlrpc reply from http://raspberrypi:44196/      time=1.660109ms  
xmlrpc reply from http://raspberrypi:44196/      time=1.592159ms
```

Ilustración 36. Ping a nodos

Finalmente, podemos obtener toda la información relativa al nodo como sus tópicos (suscripciones o publicaciones) y servicios con `rostopic /info`.

```
$ rostopic info /turtlesim
```

```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ rosnodetool info /turtlesim
-----
Node [/turtlesim]
Publications:
* /turtle1/color_sensor [turtlesim/Color]
* /rosout [roscpp_msgs/Log]
* /turtle1/pose [turtlesim/Pose]

Subscriptions:
* /turtle1/cmd_vel [unknown type]

Services:
* /turtle1/teleport_absolute
* /turtlesim/get_loggers
* /turtlesim/set_logger_level
* /reset
* /spawn
* /clear
* /turtle1/set_pen
* /turtle1/teleport_relative
* /kill

contacting node http://raspberrypi:33222/ ...
Pid: 19722
Connections:
* topic: /rosout
  * to: /rosout
  * direction: outbound
  * transport: TCPROS
```

5.5.4 Tutorial - Understanding ROS Topics

En este capítulo, interactuaremos con nodos y tópicos usando la herramienta `rostopic`. Basándonos de nuevo en `turtlesim`, ejecutamos lo siguiente:

```
$ roslaunch turtlesim turtle_teleop_key
```

La ejecución de este nodo junto con el nodo `turtlesim` permite mover la tortuga que aparece en `turtlesim` con las flechas del teclado:

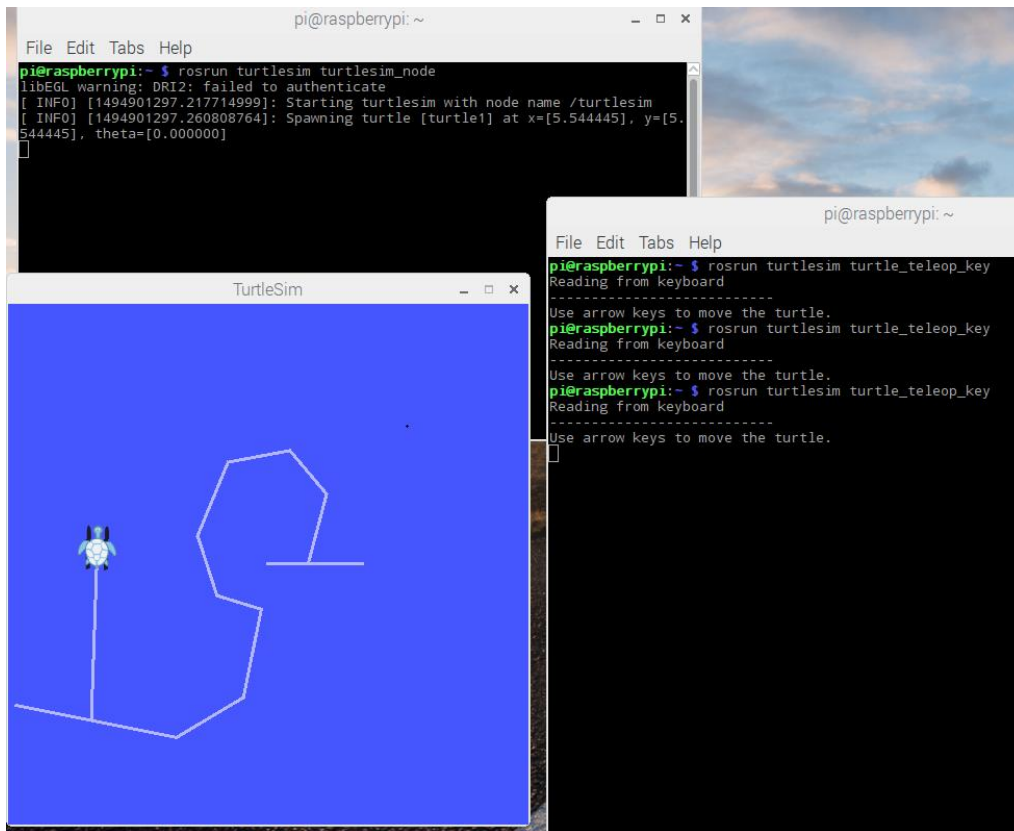


Ilustración 37. Ejemplo turtle_teleop_key

Como se puede observar, cada vez que se presiona una flecha del teclado, la tortuga se mueve en la dirección de la flecha presionada. Esto se debe que el nodo turtle_teleop_key publica un tópico con información relacionada a la velocidad (tecla presionada) /cmd_vel:

```
$ rosnode inf /teleop_turtle
```



```
pi@raspberrypi:~ $ rosnode info /teleop_turtle
-----
Node [/teleop_turtle]
Publications:
* /turtle1/cmd_vel [geometry_msgs/Twist]
* /rosout [roscpp_msgs/Log]

Subscriptions: None

Services:
* /teleop_turtle/get_loggers
* /teleop_turtle/set_logger_level

contacting node http://raspberrypi:38450/ ...
Pid: 20100
Connections:
* topic: /rosout
  * to: /rosout
  * direction: outbound
  * transport: TCPROS
* topic: /turtle1/cmd_vel
  * to: /turtlesim
  * direction: outbound
  * transport: TCPROS
```

Ilustración 38. Información nodo teleop_turtle

Mientras que el nodo que contiene a la tortuga turtlesim, está suscrito a dicha publicación:

```
$ rosnode info /turtlesim
```

```
pi@raspberrypi:~ $ rosnode info /turtlesim
-----
Node [/turtlesim]
Publications:
* /turtle1/color_sensor [turtlesim/Color]
* /rosout [roscpp_msgs/Log]
* /turtle1/pose [turtlesim/Pose]

Subscriptions:
* /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
* /turtle1/teleport_absolute
* /turtlesim/get_loggers
* /turtlesim/set_logger_level
* /reset
* /spawn
* /clear
* /turtle1/set_pen
* /turtle1/teleport_relative
* /kill
```

Ilustración 39. Información nodo turtlesim

A través del comando rostopic podemos obtener la información de todas las publicaciones de los nodos:

```
$ rostopic list
```

```
File Edit Tabs Help
pi@raspberrypi:~ $ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

Ilustración 40. Lista de tópicos

Además, se puede monitorear la información que el tópico está publicando:

```
$ rostopic echo /turtle1/cmd_vel
```

```
pi@raspberrypi:~ $ rostopic echo /turt
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Ilustración 41. Información tópico cmd_vel

5.5.5 Tutorial - Understanding ROS Services

Los servicios constituyen otra manera en que los nodos pueden comunicarse entre ellos, permitiendo el envío de peticiones y respuestas. Con el comando/herramienta rosservice, se puede obtener la lista de servicios disponibles:

```
$ rosservice list
```

```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

Ilustración 42. Lista de servicios

A la hora de usar un servicio se debe usar la siguiente estructura:

```
$ rosservice call [service] [args]
```

Como ejemplo podemos usar el servicio `/spawn`. Este servicio creará otra tortuga con una localización y orientación determinada. Para empezar, vamos a ver su tipo de mensaje:

```
rosservice type /spawn | rossrv show
```

```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ rosservice type /spawn | rossrv show
float32 x
float32 y
float32 theta
string name
---
string name
```

Ilustración 43. Lista de parámetros del servicio

Una vez que se tiene la información de los parámetros necesarios que debemos proveer, se procede a crear otra tortuga:

```
rosservice /spawn 5 5 0.5 "nueva_tortuga_2"
```

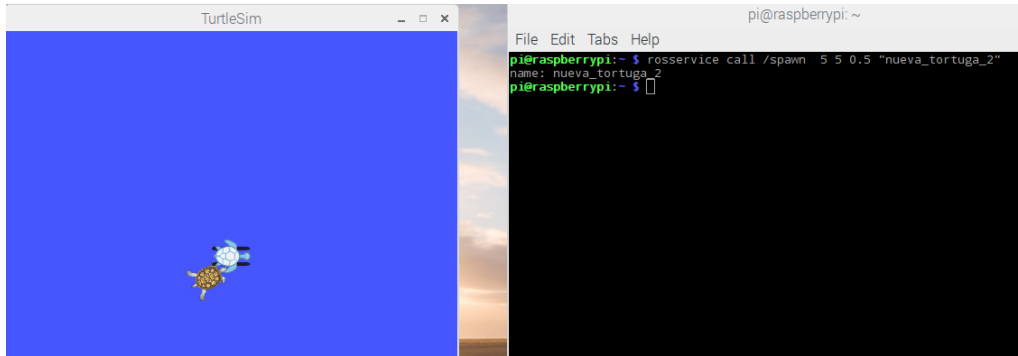


Ilustración 44. Creación de nueva tortuga

6. Protocolo de comunicación en Raspberry Pi + ROS

Como parte del desarrollo de un robot para fines educativos, es necesaria la implementación de un sistema que permita proporcionar conocimientos básicos a las personas interesadas en el mundo de la robótica, así como desarrollar aplicaciones de mayor complejidad. Teniendo este objetivo en mente, se ha diseñado un robot basado en los siguientes elementos:

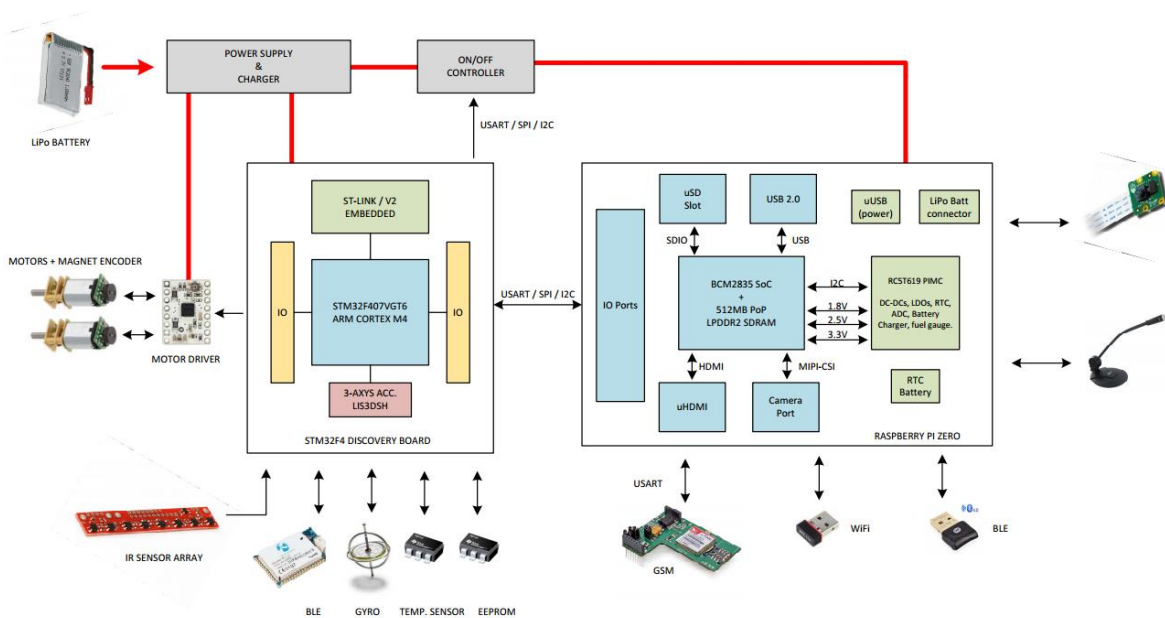


Ilustración 45. Diagrama de Robot basado en ROS

Se observa en el diagrama que el robot propuesto está basado en una computadora de placa reducida STM32F4¹⁵ junto a otra computadora de placa reducida Raspberry Pi. La placa STM32F4 se encarga del control de los motores y sensores, mientras que Raspberry Pi actuará como el cerebro del robot. Esto quiere decir que Raspberry Pi tomará decisiones de un cierto nivel de complejidad y enviará peticiones de ejecución a los sensores y/o motores a través de un protocolo de comunicación.

El alcance de este capítulo es utilizar Raspberry Pi + ROS para sentar las bases de este robot y permitir la comunicación con la placa STM32F4.

6.1 Arquitectura de la aplicación

La arquitectura de ROS está basada en nodos, los cuales, como ya hemos comentado anteriormente son ejecutables que se comunican entre ellos para intercambiar información. El robot basado en Raspberry PI podrá tener múltiples nodos dependiendo de las funcionalidades que soporte. En el caso de este proyecto, se limita a la versión básica de un nodo central (Controlador) más otro nodo que se encargue de la comunicación con la placa ST a través de SPI (master_spi).

La comunicación entre el nodo controlador y el nodo master_spi, será necesaria de dos formas:

- A través de **Tópicos**: Los tópicos cubrirán la comunicación que va unidireccionalmente desde el nodo controlador hasta el nodo master_spi. Se puede tomar como ejemplo la situación en que el módulo controlador decide que hay que parar el motor controlado por la placa ST. En este caso, el controlador publica un tópico al cual el nodo master_spi está suscrito. Cuando el nodo master_spi recibe la notificación, transmitirá a través de SPI el comando de parar el motor. En este caso el nodo controlador no espera ninguna respuesta del nodo master_spi.

¹⁵ STM32F4, <http://www.st.com/en/microcontrollers/stm32f4-series.html?querycriteria=productId=SS1577> (última visita, mayo de 2017).

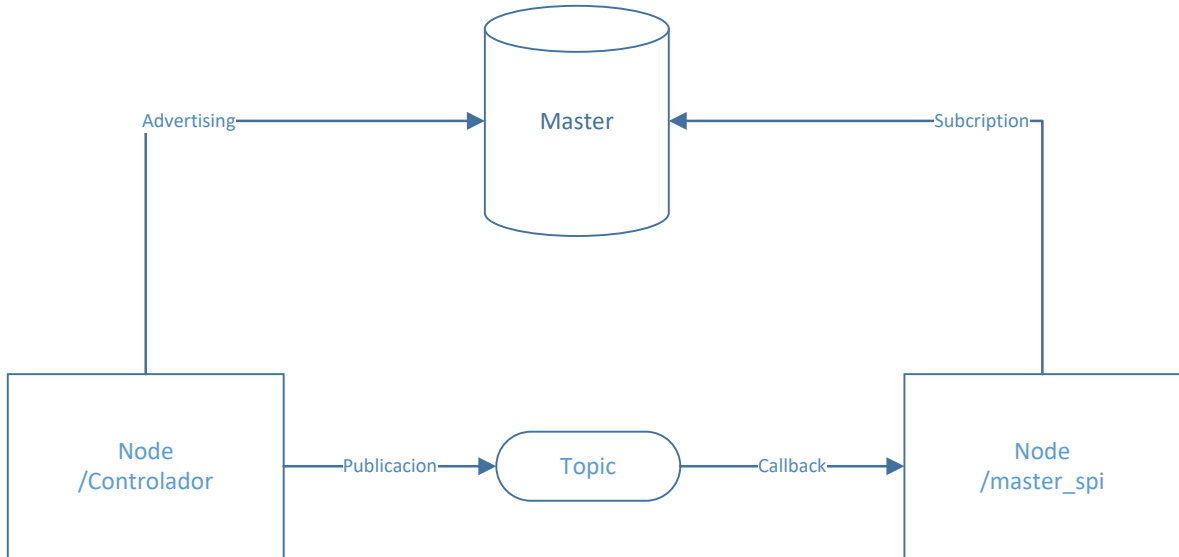


Ilustración 46. Diagrama de nodos con tópicos

- A través de **Servicios**: Los servicios cubrirán la comunicación que va bidireccionalmente desde el nodo controlador hasta el nodo master_spi. Se puede tomar como ejemplo la situación en la que el módulo controlador necesite la posición del motor controlado por la placa ST. En este caso, el controlador publica un servicio al cual el nodo master_spi está suscrito. Cuando el nodo master_spi recibe la notificación, transmitirá el comando para leer la posición del motor a través de SPI. Una vez recibida, el servicio transmitirá el valor del nodo master_spi al nodo controlador.

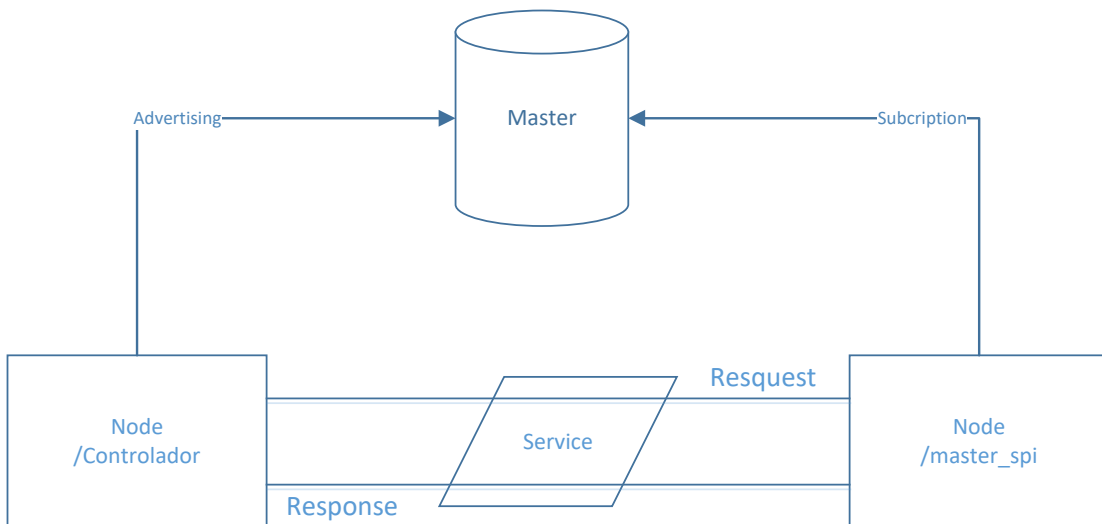


Ilustración 47. Diagrama de nodos con servicios

6.2 Implementación básica de robot

6.2.1 Solución adoptada

Debido a las limitaciones en lo que se refiere a disponibilidad de la placa STM32F4DISCOVERY, se ha decidido sustituir la misma por un display de 7 segmentos¹⁶ controlado por SPI. Este display permitirá comprobar que las librerías usadas para la comunicación SPI funcionan correctamente. Para poder usar SPI en Raspberry Pi, se debe de habilitar esta interfaz¹⁷.

Los materiales requeridos para implementación del robot, son los siguientes:

- 40-pin Pi Wedge¹⁸
- Raspberry
- Protoboard¹⁹
- Cables conectores
- Display de 7 segmentos

El display de 7 segmentos debe de quedar conectado a Raspberry Pi a través del Pi Wedge de la siguiente manera:

Rabpberry Pi Signal	Serial 7-seg Signal
GND	GND
3.3V	VCC
CE1	SS (Shift Select)
SCK	SCK
MOSI	SDI
MISO	SDO

¹⁶ 7- Segment Serial Display, https://www.sparkfun.com/products/11441?_ga=2.133104782.1424294466.1495072057-1921775219.1494198110 (última visita, mayo de 2017).

¹⁷ Habilitar SPI, <http://www.raspberrypi-spy.co.uk/2014/08/enabling-the-spi-interface-on-the-raspberry-pi/> (última visita, mayo de 2017).

¹⁸ 40-pin PI Wedge, https://www.sparkfun.com/products/13717?_ga=2.87467864.1386560376.1495071753-1921775219.1494198110 (última visita, mayo de 2017).

¹⁹ BreadBoard, https://www.sparkfun.com/products/12615?_ga=2.113457671.1491397261.1495071952-1921775219.1494198110 (última visita, mayo de 2017).

Tabla 2. Conexión entre Raspberry Pi y Serial 7-seg

Quedando las conexiones de la siguiente manera:

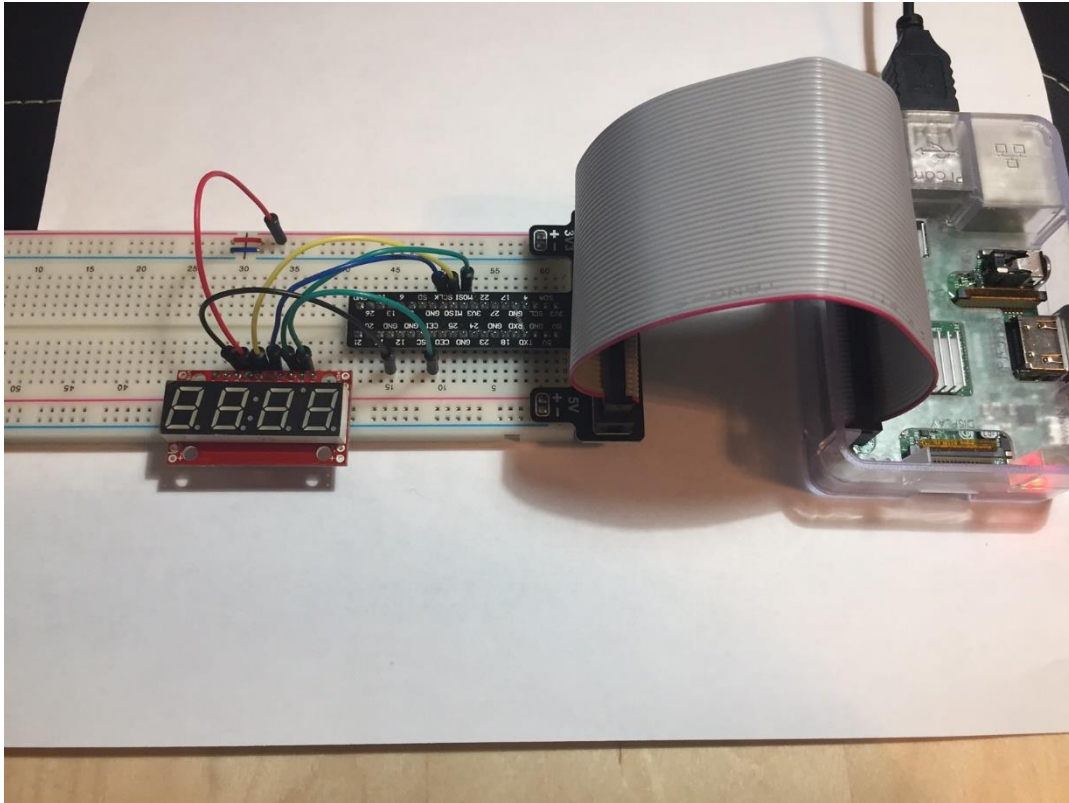


Ilustración 48. Montaje de Robot (Raspberry Pi 3 + LED)

6.2.2 Preparación del entorno

Una vez configurado el entorno se procede a la fase de desarrollo. Se aprovechará el entorno creado en la instalación de ROS para crear los paquetes adicionales del robot:

- **Nodo controlador:** Nodo encargado de tomar decisiones. Será capaz de publicar un tópico llamado motor cada 5 segundos y junto con el número de veces que se ha publicado el tópico. El número de veces varía de 0 a 255.
- **Nodo master_spi:** Nodo encargado de controlar las comunicaciones con el esclavo (Display de 7 segmentos) basado en los tópicos recibidos. Recibirá el tópico motor, junto con el número de veces que el nodo se ha publicado.

6.2.2.1 Nodo controlador

Se ha de crear un nodo que contenga el programa que permita el envío de un tópico con información sobre el número de publicaciones. Lo primero que se debe de realizar es la creación de un paquete:

```
cd ~/ros_catkin_ws/src
catkin_create_pkg controlador_test std_msgs roscpp
```

Con los siguientes elementos:

```
package_n/
  CMakeLists.txt           -- CMakeLists.txt para compilar
  package.xml              -- Información del paquete
  README.md                -- Nombre del paquete
  include/                 -- Carpeta para .h
  src/                     -- Carpeta para .cpp
  controlador_test.cpp     -- Archivo controlador
```

La generación de archivos es automática, excepto por el controlador _test.cpp. A continuación, podemos encontrar el código que permite publicar el tópico motor cada 5 segundos:

```
#include <ros/ros.h>
#include <std_msgs/UInt8.h>
#include <sstream>

int main(int argc, char** argv)
{
  //Inicializa controlador_test
  ros::init(argc, argv, "controlador_test");
  ros::NodeHandle n;
  //Topico motor
  ros::Publisher motor_pub = n.advertise<std_msgs::UInt8>("motor", 1000);
  //Retraso de 5 segundos
```

```

ros::Rate loop_rate(5);
//Velocidad del motor
std_msgs::UInt8 motorSpeed;
//Contador de numero de publicaciones
uint count = 0;
while (ros::ok())
{
//Simulamos velocidad del motor con numero de publicaciones
motorSpeed.data = count;
//Muestra por pantalla numero de publicaciones/velocidad motor
ROS_INFO("%d", motorSpeed.data);
//Publicacion de topico
motor_pub.publish(motorSpeed);
//Retraso y incremento de publicacion
ros::spinOnce();
loop_rate.sleep();
count++;
}
}

```

Además, es necesario modificar el archivo CMakeLists.txt para configurar las opciones del compilador:

```

cmake_minimum_required(VERSION 2.8.3)
project(controlador_node)

find_package(catkin REQUIRED COMPONENTS
  roscpp
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

catkin_package(
  INCLUDE_DIRS include
)

```

```

include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  ${PROJECT_NAME}/include
)

add_executable(controlador_test src/controlador_test.cpp)
add_dependencies(controlador_test wiringPi)
target_link_libraries(controlador_test
  ${catkin_LIBRARIES}
  ${WIRINGPI_LIBRARY}
)

```

6.2.2.2 Nodo master_spi

Se ha de crear un nodo que contenga el programa que permita la suscripción a un tópico con información sobre el número de publicaciones. Lo primero que se debe de realizar es la creación de un paquete:

```

cd ~/ros_catkin_ws/src
catkin_create_pkg master_spi std_msgs roscpp

```

Con los siguientes elementos:

```

package_n/
  CMakeLists.txt           -- CMakeLists.txt para compilar
  package.xml              -- Información del paquete
  README.md                -- Nombre del paquete
  include/                 -- Carpeta para .h
  src/                     -- Carpeta para .cpp
  master_spi.cpp           -- Archivo controlador

```

La generación de archivos es automática, excepto por `master_spi.cpp`. A continuación, podemos encontrar el código para suscribir un tópico y enviar la información a un display de 7 segmentos a través de SPI:

```
#include "ros/ros.h"
#include <errno.h>
#include <wiringPiSPI.h>
#include <unistd.h>
#include <std_msgs/UInt8.h>

static const int CHANNEL = 1;

void motorCallback(const std_msgs::UInt8::ConstPtr& msg)
{
    unsigned char buffer[100];
    int result;

    // Deciamles y putnos
    buffer[0] = 0x77;
    buffer[1] = msg->data;
    result = wiringPiSPIDataRW(CHANNEL, buffer, 2);

    // Primer digito
    buffer[0] = 0x7b;
    buffer[1] = msg->data;
    result = wiringPiSPIDataRW(CHANNEL, buffer, 2);

    // Segundo digito
    buffer[0] = 0x7c;
    buffer[1] = msg->data;
    result = wiringPiSPIDataRW(CHANNEL, buffer, 2);

    // Tercer digito
    buffer[0] = 0x7d;
    buffer[1] = msg->data;
    result = wiringPiSPIDataRW(CHANNEL, buffer, 2);
}
```

```

    // Cuarto digito
    buffer[0] = 0x7e;
    buffer[1] = msg->data;
    result = wiringPiSPIDataRW(CHANNEL, buffer, 2);

    ROS_INFO("I heard: [%d]", msg->data);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "master_spi");
    ros::NodeHandle n;

    int fd;

    // CHANNEL insicates chip select,
    // velocidad de transmision - 500000 .
    fd = wiringPiSPISetup(CHANNEL, 500000);

    ros::Subscriber sub = n.subscribe("motor", 1000, motorCallback);

    ros::spin();

    return 0;
}

```

Además, es necesario modificar el archivo CMakeLists.txt para configurar las opciones del compilador:

```

cmake_minimum_required(VERSION 2.8.3)
project(controlador_node)
find_package(catkin REQUIRED COMPONENTS
    roscpp
)
## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

```

```
catkin_package(  
  INCLUDE_DIRS include  
#  CATKIN_DEPENDS roscpp  
)  
include_directories(  
  include  
  ${catkin_INCLUDE_DIRS}  
  ${PROJECT_NAME}/include  
)  
add_executable(controlador_test src/controlador_test.cpp)  
add_dependencies(controlador_test wiringPi)  
target_link_libraries(controlador_test  
  ${catkin_LIBRARIES}  
  ${WIRINGPI_LIBRARY}  
)
```

6.2.3 Compilación y ejecutables

Una vez que los nodos están listos, se puede proceder a la compilación y creación de los ejecutables. Para compilar los paquetes en el entorno `catkin` se deben de ejecutar los siguientes comandos:

```
$ cd ~/ros_catkin_ws  
$ catkin_make
```

Si el proceso de compilación del entorno resulta exitoso, se generan los ejecutables de los nodos correspondientes a `controlador_test` y `master_spi` en el directorio `ros_catkin_ws/devel/lib`.

6.2.4 Prueba de funcionamiento

Terminado el proceso de compilación y generación de ejecutables, se puede proceder a la fase de pruebas y comprobaciones.

Se lleva a cabo el conexionado de todos los elementos tal y como se ha descrito con anterioridad y se procede a la inicialización de los nodos:

- **Inicialización de roscore:** Nueva sesión de terminal.

```
$ roscore
```

- **Inicialización del nodo master_spi:** Nueva sesión de terminal a través del ejecutable o creando un archivo launch.

```
$ ./master_spi_node
```

- **Inicialización del nodo controlador_test:** Nueva sesión de terminal a través del ejecutable o creando un archivo launch.

```
$ ./controlador_test
```

Una vez que el servicio roscore está correctamente inicializado, el nodo controlador_test empieza a publicar el tópico motor, mientras que el nodo master_spi consume dicho tópico, como se puede observar en la siguiente imagen:

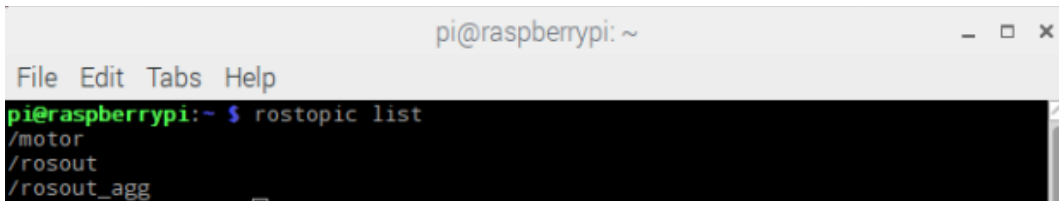
The image shows four terminal windows illustrating the ROS environment setup and communication:

- Top-left window:** Shows the output of the `roscout` command, displaying a list of INFO messages for various nodes, including `controlador_test` and `master_spi`.
- Top-right window:** Shows the output of the `rostopic echo /motor` command, displaying a series of numerical values (e.g., 26, 27, 28, 29, 30, 31, 32, 33, 34, 35) received from the `/motor` topic.
- Bottom-left window:** Shows the output of the `roslaunch` command, displaying the launch configuration for the `roscout` node, including the master URL and the path to the `roscout` executable.
- Bottom-right window:** Shows the output of the `rostopic echo /motor` command, displaying a series of numerical values (e.g., 26, 27, 28, 29, 30, 31, 32, 33, 34, 35) received from the `/motor` topic.

Ilustración 49. Comunicación entre nodo controlador y nodo master_spi

Si se ejecuta el comando para listar los tópicos, se obtienen los tópicos activos, incluyendo en nuestro caso /motor:

```
$ rostopic list
```

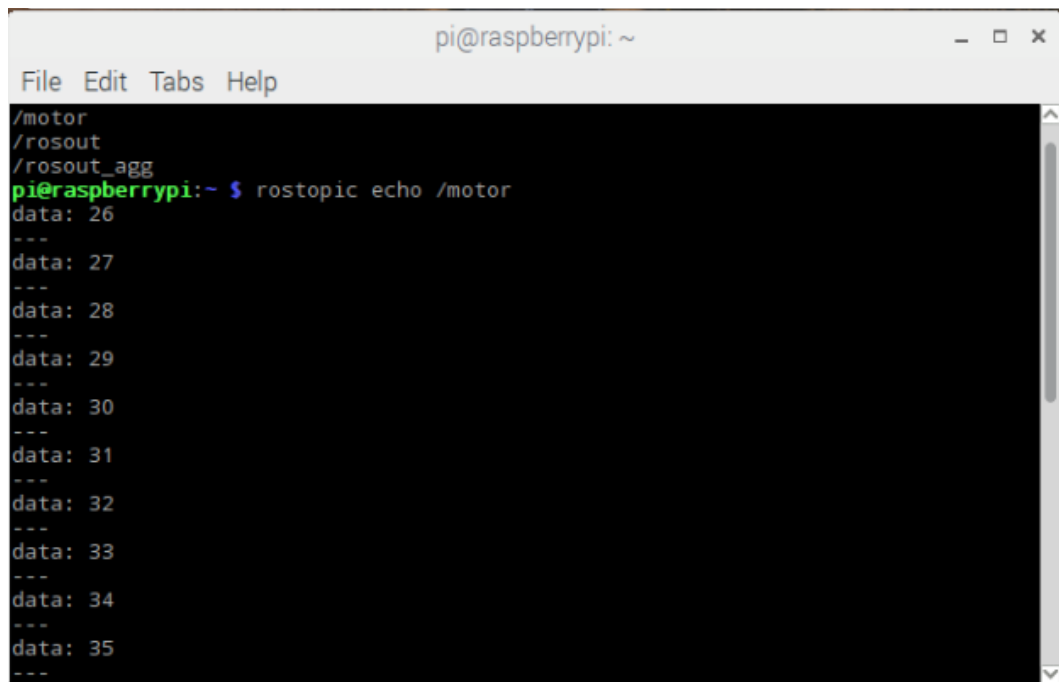


```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ rostopic list  
/motor  
/rosout  
/rosout_agg
```

Ilustración 50. Lista de nodos en robot

Si se ejecuta el comando para ver qué está publicando un tópico, se obtiene la información que dicho nodo está publicando, en nuestro caso /motor:

```
$ rostopic echo /motor
```



```
pi@raspberrypi: ~  
File Edit Tabs Help  
/motor  
/rosout  
/rosout_agg  
pi@raspberrypi:~ $ rostopic echo /motor  
data: 26  
---  
data: 27  
---  
data: 28  
---  
data: 29  
---  
data: 30  
---  
data: 31  
---  
data: 32  
---  
data: 33  
---  
data: 34  
---  
data: 35  
---
```

Ilustración 51. Publicación de nodos en robot

Adicionalmente, se puede ver que los dígitos de los 4 displays de 7 segmentos cambian cada 5 segundos con la información que proviene del tópico /motor.

7. Conclusiones

Como se puede observar a lo largo del proyecto, las ventajas y alternativas que ofrece ROS son muy extensas a la hora de implementar aplicaciones de robótica, debido a que actúa conjuntamente con el sistema operativo, situándose en la capa de abstracción. Esto permite la portabilidad de componentes y librerías entre distintas plataformas y proyectos.

Raspberry Pi por un lado presenta ciertas limitaciones a la hora de llevar a cabo tareas que requieren más memoria como cálculos muy complejos o tratamiento de imágenes en 3D necesitando un aumento de la memoria SWAP. Este aumento reduce considerablemente el tiempo de vida de las tarjetas SCARD. Por otro lado, Raspberry Pi permite el desarrollo de aplicaciones de robótica a un precio muy asequible, siendo especialmente recomendable para:

- Desarrollo de aplicaciones robóticas para sistemas empujados
- Desarrollo de aplicaciones en el ámbito de la enseñanza
- Aprendizaje de nuevos conceptos basado en ROS
- Control de placas con más interfaces digitales o analógicas procesando su información a través de los protocolos de comunicaciones como SPI, UART or I2C

El proyecto realizado se puede extender de las siguientes formas:

- Implementación de servicios en ROS para la comunicación entre futuros nodos
- Instalación de ROS en distintas plataformas en caso de que se necesite más potencia de procesamiento.
- Desarrollo completo de librería para comunicación con la placa STM32F4 basado en SPI.
- Desarrollo de nodos para el control de WiFi, BLE, GSM y control de voz.

Finalmente se puede recalcar que Raspberry pi + ROS es una combinación perfecta para realizar una inmersión al mundo de la robótica en sistemas empujados con un coste muy bajo pero con unas prestaciones muy altas.

8. Bibliografía

- [1]. ROS, *Introduction*, <http://wiki.ros.org/ROS/Introduction> (última visita, mayo de 2017).
- [2]. *Installing ROS Kinetic on the Raspberry Pi*, <http://wiki.ros.org/ROSBerryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi> (última visita, mayo de 2017).
- [3]. *Jessie Raspbian*, <https://www.raspberrypi.org/downloads/raspbian/> (última visita, mayo de 2017).
- [4]. *Jessie Raspbian*, <https://www.raspberrypi.org/downloads/raspbian/> (última visita, mayo de 2017).
- [5]. *Installation Guide*, <https://www.raspberrypi.org/documentation/installation/installing-images/README.md> (última visita, mayo de 2017).
- [6]. *ROSBerryPi/Installing ROS Kinetic on the Raspberry Pi*, <http://wiki.ros.org/ROSBerryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi> (última visita, mayo de 2017).
- [7]. *TurtleSim*, <http://wiki.ros.org/turtlesim> (última visita, mayo de 2017).
- [8]. *ROS concepts*, <http://wiki.ros.org/ROS/Concepts> (última visita, mayo de 2017).
- [9]. Lentin Joseph, *Mastering ROS for robotics Programming* (Packt Publishing, 2015).
- [10]. *Packages ROS*, <http://wiki.ros.org/Packages> (última visita, mayo de 2017).

- [11]. *Standart Types*, <http://wiki.ros.org/msg> (última visita, mayo de 2017).
- [12]. Lentin Joseph, *Mastering ROS for robotics Programming* (Packt Publishing, 2015).
- [13]. *TurtleSim*, <http://wiki.ros.org/turtlesim> (última visita, mayo de 2017).
- [14]. *Tutoriales Ros*, <http://wiki.ros.org/ROS/Tutorials> (última visita, mayo de 2017).
- [15]. *STM32F4*, <http://www.st.com/en/microcontrollers/stm32f4-series.html?querycriteria=productId=SS1577> (última visita, mayo de 2017).
- [16]. *7- Segment Serial Display*,
https://www.sparkfun.com/products/11441?_ga=2.133104782.1424294466.1495072057-1921775219.1494198110 (última visita, mayo de 2017).
- [17]. *Habilitar SPI*, <http://www.raspberrypi-spy.co.uk/2014/08/enabling-the-spi-interface-on-the-raspberry-pi/> (última visita, mayo de 2017).
- [18]. *40-pin PI Wedge*,
https://www.sparkfun.com/products/13717?_ga=2.87467864.1386560376.1495071753-1921775219.1494198110 (última visita, mayo de 2017).
- [19]. *BreadBoard*,
https://www.sparkfun.com/products/12615?_ga=2.113457671.1491397261.1495071952-1921775219.1494198110 (última visita, mayo de 2017).