

Raspberry pi – ROS



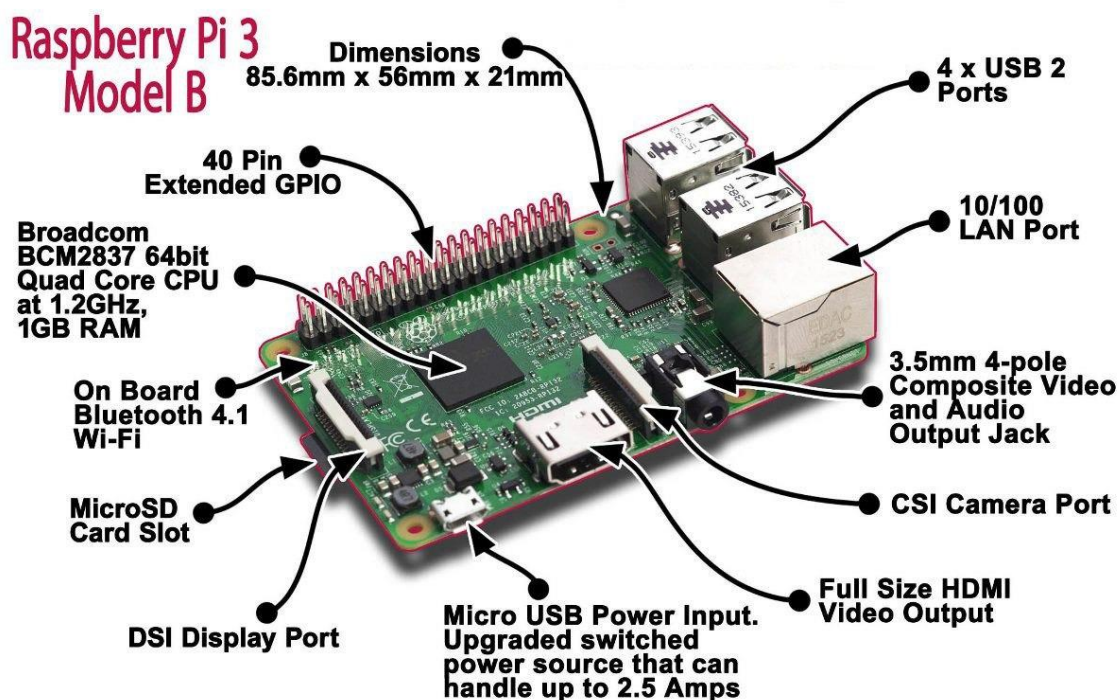
Que es la raspberry pi?

La Raspberry Pi (RPI) es una minicomputadora de bajo coste, cuyos primeros modelos fueron lanzados en abril de 2012. Generalmente este tipo de mini computadoras ejecutan sistemas operativos basados en Linux y están íntimamente relacionados con el Open Software. No obstante, el desarrollo en sí de Raspberry Pi no es Open Hardware.

La RPI incorpora funciones de electrónica como pines GPIO (General Purpose Input/Output), y de comunicación como UART (Universal Asynchronous Receiver-Transmitter), y SPI (Serial Peripheral Interface), I²C (Inter-Integrated Circuit). Estas funciones hacen que pueda ser empleado en proyectos de electrónica y robótica interactuando con sensores (temperatura, luz, aceleración...) y actuadores (motores, servos, relés...). En este sentido, podríamos decir que “comparte” ciertas capacidades con dispositivos como Arduino, aunque con mucha más potencia de computación y el beneficio de correr un sistema operativo on-board, más la posibilidad de comunicación WIFI y Bluetooth.

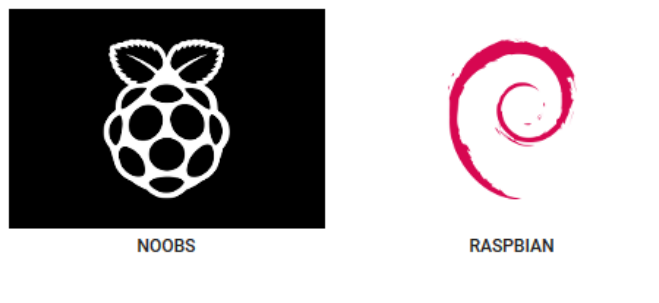
Las características técnicas de la última versión de la RPI son las siguientes:

Raspberry Pi 3 Modelo B+	
Procesador	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC
Frecuencia de reloj	1,4 GHz
GPU	VideoCore IV 400 MHz
Memoria	1GB LPDDR2 SDRAM
Conectividad inalámbrica	2.4GHz / 5GHz IEEE 802.11.b/g/n/ac Bluetooth 4.2, BLE
Conectividad de red	Gigabit Ethernet over USB 2.0 (300 Mbps de máximo teórico)
Puertos	GPIO 40 pines HDMI 4 x USB 2.0 CSI (cámara Raspberry Pi) DSI (pantalla tácil) Toma auriculares / vídeo compuesto Micro SD Micro USB (alimentación) Power-over-Ethernet (PoE)

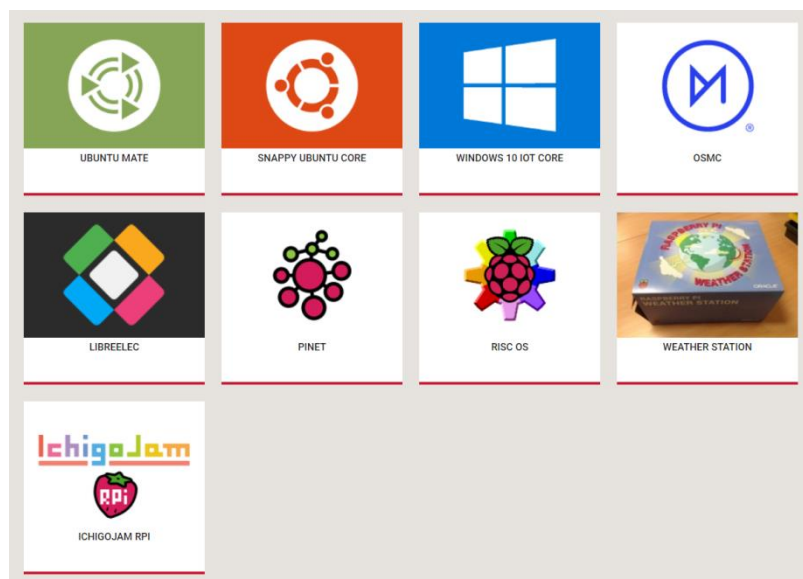


En el sitio oficial de la RPI <https://www.raspberrypi.org/> podemos encontrar proyectos desarrollados, información de utilización y software útil para la misma.

Respecto al sistema operativo que es posible instalar en ella, sus fabricantes recomiendan el sistema operativo RASPBIAN y su instalador NOOBS:



Sin embargo la RPI soporta una variedad de sistemas operativos que pueden ser descargados de la página oficial:



Si bien ROS puede correr en Raspbian, por cuestiones de compatibilidad con la instalación de Ubuntu en la máquina de escritorio que utilizamos hasta aquí, elegiremos el sistema operativo **Ubuntu Mate** para nuestras RPI.

El proceso de instalación del sistema operativo y configuración de la RPI consiste en descargar la imagen del sistema operativo deseado, grabarla sobre una tarjeta SD (esta



será el equivalente a memoria solididad de nuestra RPI), colocar la memoria en la RPI y conectarla. Terminado esto tendremos boteando la RPI como una computadora normal.

Para nuestro caso la imagen la descargamos de:

<https://ubuntu-mate.org/raspberry-pi/>

Una vez nuestro sistema operativo funcione, instalaremos **ROS Kinetic** siguiendo los pasos indicados en:

<http://wiki.ros.org/kinetic/Installation/Ubuntu>

Dependiendo de que queremos hacer con nuestra RPI podremos elegir que instalación de ROS realizar, en general como no correremos aplicación de visualización sobre la RPI la versión Desktop será suficiente, si necesitamos algún paquete específico podremos descargarlo e instalarlo en forma independiente.

Como usar la RPI?

- Como una **computadora tradicional** (conectando pantalla, teclado y mouse): esta forma de trabajo será similar a una máquina de escritorio con el sistema operativo elegido en nuestro caso Ubuntu Mate.
- **Vía red usando SSH o escritorio virtual**: en esta forma no necesitaremos tener nada conectado a nuestra RPI, solo que esta se encuentre configurada y conectada a una red.

Trabajando en la RPI en forma remota vía SSH:

Ya que esta será la forma más común de uso, en especial si pensamos en montar nuestra RPI sobre un robot, daremos más detalles sobre como configurarla y como trabajar con ella.

Para poder operar en forma remota la RPI primero deberemos configurarla, para esto necesitaremos conectar teclado y monitor a ella.

Iniciaremos la configuración chequeando si la RPI y la computadora desde donde la comandaremos se “vean” en la red, para esto primero obtendremos las direcciones IP de ambos dispositivos, para esto en cada máquina en un terminal usamos el comando:

```
ifconfig
```

En la PC obtendríamos algo similar a:

```
alumno@alumno: ~  
alumno@alumno:~$ ifconfig  
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:04:19:dd  
            inet addr:10.0.12.137  Bcast:10.0.12.255  Mask:255.255.255.0  
            inet6 addr: fe80::bdeb:bde8:2049:ed9f/64 Scope:Link  
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
            RX packets:105600 errors:0 dropped:0 overruns:0 frame:0  
            TX packets:14408 errors:0 dropped:0 overruns:0 carrier:0  
            collisions:0 txqueuelen:1000  
            RX bytes:128641126 (128.6 MB)  TX bytes:976479 (976.4 KB)  
  
lo          Link encap:Local Loopback  
            inet addr:127.0.0.1  Mask:255.0.0.0  
            inet6 addr: ::1/128 Scope:Host  
            UP LOOPBACK RUNNING  MTU:65536  Metric:1  
            RX packets:286 errors:0 dropped:0 overruns:0 frame:0  
            TX packets:286 errors:0 dropped:0 overruns:0 carrier:0  
            collisions:0 txqueuelen:1000  
            RX bytes:30646 (30.6 KB)  TX bytes:30646 (30.6 KB)  
  
alumno@alumno:~$
```

En la RPI obtendríamos:

```
rpi@rpi-desktop: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
rpi@rpi-desktop:~$ ifconfig  
enxb827eb6d6594 Link encap:Ethernet  HWaddr b8:27:eb:6d:65:94  
                UP BROADCAST MULTICAST  MTU:1500  Metric:1  
                RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
                TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
                collisions:0 txqueuelen:1000  
                RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)  
  
lo          Link encap:Local Loopback  
            inet addr:127.0.0.1  Mask:255.0.0.0  
            inet6 addr: ::1/128 Scope:Host  
            UP LOOPBACK RUNNING  MTU:65536  Metric:1  
            RX packets:252 errors:0 dropped:0 overruns:0 frame:0  
            TX packets:252 errors:0 dropped:0 overruns:0 carrier:0  
            collisions:0 txqueuelen:1  
            RX bytes:24816 (24.8 KB)  TX bytes:24816 (24.8 KB)  
  
wlan0      Link encap:Ethernet  HWaddr b8:27:eb:38:30:c1  
            inet addr:10.0.12.238  Bcast:10.0.12.255  Mask:255.255.255.0  
            inet6 addr: fe80::c07e:ebf3:4530:89a4/64 Scope:Link  
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
            RX packets:302591 errors:0 dropped:0 overruns:0 frame:0  
            TX packets:159876 errors:0 dropped:0 overruns:0 carrier:0  
            collisions:0 txqueuelen:1000  
            RX bytes:425002184 (425.0 MB)  TX bytes:14938170 (14.9 MB)  
  
rpi@rpi-desktop:~$
```

Con este comando encontramos los nombres y detalles de todas las interfaces de red activas en el sistema, usualmente enXX (Ethernet interfaz), lo (loopback interface – este tipo especial de interfaz en general es usada en tareas de diagnóstico por el sistema para comunicarse consigo mismo), wlanX (interfaz inalámbrica).



- Si nos encontramos en Windows para conocer nuestra IP desde cmd usamos: ipconfig



En el caso que venimos analizando, estamos corriendo una máquina virtual con Ubuntu 16 (PC) conectada mediante Ethernet a una red con un punto de acceso WIFI al cual está conectada nuestra RPI, con esto en cuenta podemos ver cuáles serán las direcciones IP que nos interesarán para comunicar ambas maquinas:

PC: 10.0.12.137

RPI: 10.0.12.238

Para comprobar la conexión realizaremos un ping desde la PC a la RPI y viceversa, donde si la conexión es correcta obtendremos algo como lo siguiente:

En la PC:

```
alumno@alumno: ~  
alumno@alumno:~$ ping 10.0.12.238  
PING 10.0.12.238 (10.0.12.238) 56(84) bytes of data.  
64 bytes from 10.0.12.238: icmp_seq=1 ttl=64 time=3.07 ms  
64 bytes from 10.0.12.238: icmp_seq=2 ttl=64 time=5.13 ms  
64 bytes from 10.0.12.238: icmp_seq=3 ttl=64 time=3.46 ms  
64 bytes from 10.0.12.238: icmp_seq=4 ttl=64 time=4.96 ms  
64 bytes from 10.0.12.238: icmp_seq=5 ttl=64 time=5.48 ms  
64 bytes from 10.0.12.238: icmp_seq=6 ttl=64 time=6.62 ms  
64 bytes from 10.0.12.238: icmp_seq=7 ttl=64 time=4.79 ms  
64 bytes from 10.0.12.238: icmp_seq=8 ttl=64 time=4.28 ms  
64 bytes from 10.0.12.238: icmp_seq=9 ttl=64 time=3.49 ms
```

En la RPI:

```
rpi@rpi-desktop: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
rpi@rpi-desktop:~$ ping 10.0.12.137  
PING 10.0.12.137 (10.0.12.137) 56(84) bytes of data.  
64 bytes from 10.0.12.137: icmp_seq=1 ttl=64 time=2.60 ms  
64 bytes from 10.0.12.137: icmp_seq=2 ttl=64 time=7.73 ms  
64 bytes from 10.0.12.137: icmp_seq=3 ttl=64 time=7.31 ms  
64 bytes from 10.0.12.137: icmp_seq=4 ttl=64 time=4.60 ms  
64 bytes from 10.0.12.137: icmp_seq=5 ttl=64 time=4.01 ms  
64 bytes from 10.0.12.137: icmp_seq=6 ttl=64 time=9.80 ms
```



IMPORTANTE:

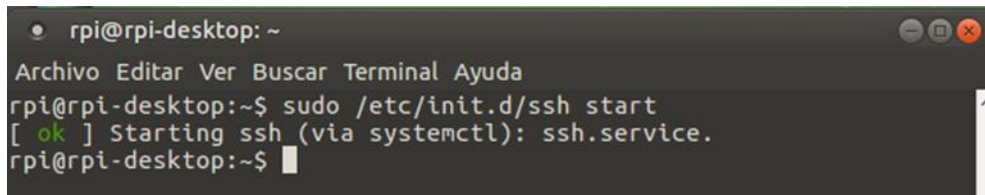
- Algunas veces el firewall no permite la conexión mediante ping, por lo que podemos desconectarlo momentáneamente.
- Si estamos usando una máquina virtual deberemos conectar el adaptador de red de la misma, en la forma de adaptador puente de red (de esta forma quedamos conectados en la misma red que la PC anfitriona, y en nuestro ejemplo que la RPI).
- Es conveniente si vamos a trabajar a distancia con nuestra RPI configurar una IP fija para la misma, esto lo realizamos configurando el archivo `/etc/network/interfaces` de Ubuntu.



Comprobada la conexión, pasamos a la **conexión mediante SSH** (o Secure Shell) es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. Para esto necesitamos que el SSH esté instalado en la maquina a donde deseamos conectarnos, en Ubuntu mate viene preinstalado pero no activado por lo que en la RPI corremos el siguiente comando para activarlo:

```
sudo /etc/init.d/ssh start
```

Veremos algo como:



```
rpi@rpi-desktop: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
rpi@rpi-desktop:~$ sudo /etc/init.d/ssh start  
[ ok ] Starting ssh (via systemctl): ssh.service.  
rpi@rpi-desktop:~$
```



- Si usaremos nuestra RPI a distancia seria conveniente que el SSH arranque solo al inicio de sección, esto lo podemos lograr con el siguiente comando: `sudo systemctl enable ssh`

Terminada la configuración probaremos la conexión, la forma de conexión genérica será:

```
ssh usuario@ip
```

Dónde:

- **usuario** es el usuario que deseamos utilizar perteneciente al equipo al cual nos conectamos.
- **ip** es la dirección IP del equipo al que deseamos conectarnos.



Por ejemplo, para el caso que venimos trabajando:

```
rpi@rpi-desktop: ~  
alumno@alumno:~$ ssh rpi@10.0.12.238  
The authenticity of host '10.0.12.238 (10.0.12.238)' can't be established.  
ECDSA key fingerprint is SHA256:oyJ834y14PBF8VZTLUGaEbxfermoJyPKKQeanQo02zM.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '10.0.12.238' (ECDSA) to the list of known hosts.  
rpi@10.0.12.238's password:  
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.9.80-v7+ armv7l)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
Pueden actualizarse 484 paquetes.  
2 actualizaciones son de seguridad.  
  
*** Es necesario reiniciar el sistema ***  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
  
"source: command not found  
rpi@rpi-desktop:~$
```

Vemos que la primera vez que nos conectemos nos preguntara si deseamos agregar a la IP de conexión a la lista de host conocidos, y luego nos solicitara la contraseña del usuario del dispositivo al que nos conectaremos.

Finalmente vemos que la conexión está establecida, cualquier comando que ejecutemos en esta consola será ejecutado en la maquina servidor, en nuestro caso la RPI.

Si bien aquí conectamos a la RPI a distancia, lo desarrollado es válido para la conexión inversa.

Finalmente si deseamos terminar la conexión podemos utilizar el comando:

exit

```
alumno@alumno: ~  
rpi@rpi-desktop:~$ exit  
logout  
Connection to 10.0.12.238 closed.  
alumno@alumno:~$
```



- Una opción interesante de SSH es la `-X` la cual permite ejecutar aplicaciones graficas corriendo en la maquina servidor, pero vistas en la maquina conectada: `ssh -X usuario@ip`
- Con esta versión podemos correr el gestor de ventanas terminator (ver primer apunte), y de esta forma iniciar solo una terminal con la comunicación atreves del SSH.



Bien tenemos nuestra RPI funcionando con una instalación de Ubuntu Mate, y sobre ella ROS, para comprobar la instalación como siempre podríamos correr el master y ver si funciona. Lo que sigue será intentar trabajar con ROS tanto en la RPI como en la PC al mismo tiempo.

Como correr ROS entre múltiples maquinas?

Como habíamos dicho ROS se encuentra diseñado con la idea de poder distribuir los procesos que se ejecutan en distintas plataformas. En teoría una buena programación con nodos, no debería importarle a través de que red se esté ejecutando, esto nos permitiría relocalizar procesos de acuerdo a los recursos disponibles (por supuesto hay excepciones, un nodo que accede a un driver local debe correr en la maquina donde se encuentre este hardware). Para correr ROS a través de varias máquinas deben cumplirse los siguientes requerimientos:

- Solo existirá un master
- Todos los nodos deben ser configurados para usar el mismo master, usando **ROS_MASTER_URI**
- Debe existir una conectividad completa bidireccional entre las maquinas.

Lo que haremos aquí será correr nodos de ROS desde distintas maquinas, para hacer esto deberemos seleccionar donde se encontrará el maestro corriendo y en función de ello configuraremos el sistema.

Partiremos con la idea de ejecutar el maestro sobre la RPI y un nodo sobre la máquina de escritorio. Presentamos los comandos a ejecutar en orden secuencial:

Paso	En RPI:	En PC:	Descripción
1	roscore		Iniciamos el maestro
2	hostname -I		Consultamos cual es la dirección de la maquina maestro
3	export ROS_IP=ip_maquina_maestro		Establecemos dirección de conexión para todos los nodos en esta máquina.
4		export ROS_MASTER_URI=http://ip_maquina_maestro:11311	Establecemos dirección de conexión al maestro
5		hostname -I	Consultamos cual es la dirección de la maquina con nodos
6		export ROS_IP=ip_maquina_nodo	Establecemos dirección de conexión para todos los nodos en esta máquina.



En la PC tendremos algo similar a:

```
alumno@alumno: ~
alumno@alumno: ~ 65x6
alumno@alumno:~$ export ROS_MASTER_URI=http://10.0.12.238:11311
alumno@alumno:~$ hostname -I
10.0.12.137
alumno@alumno:~$ export ROS_IP=10.0.12.137
alumno@alumno:~$
```

Y en la RPI:

```
rpi@rpi-desktop: ~
Archivo Editar Ver Buscar Terminal Ayuda
rpi@rpi-desktop:~$ hostname -I
10.0.12.238
rpi@rpi-desktop:~$ export ROS_IP=10.0.12.238
rpi@rpi-desktop:~$
```

Para continuar con el ejemplo probamos correr el simulador **turtlesim** en la RPI y un nodo de teleoperación en la PC (similar a lo que hicimos en el primer apunte, pero ahora los nodos están distribuidos):

7	roslaunch turtlesim turtlesim_node		Lanzamos el nodo del simulador turtlesim
8		roslaunch turtlesim turtle_teleop_key	Lanzamos el nodo del simulador turtlesim

Si todo funciona, deberíamos comandar el movimiento de la tortuga del simulador corriendo en la RPI, mediante un nodo corriendo en la PC. De esta forma hemos arribado a todo lo necesario para la comunicación entre múltiples máquinas en ROS, por ejemplo un nodo cuyo código se encuentre en el espacio de trabajo de la PC (catkin_ws, en nuestro ejemplo) al ejecutarse en la PC intercambiara información con todo los nodos registrados en el master de la RPI.



Actividad

Pruebe la conexión en sentido inverso.



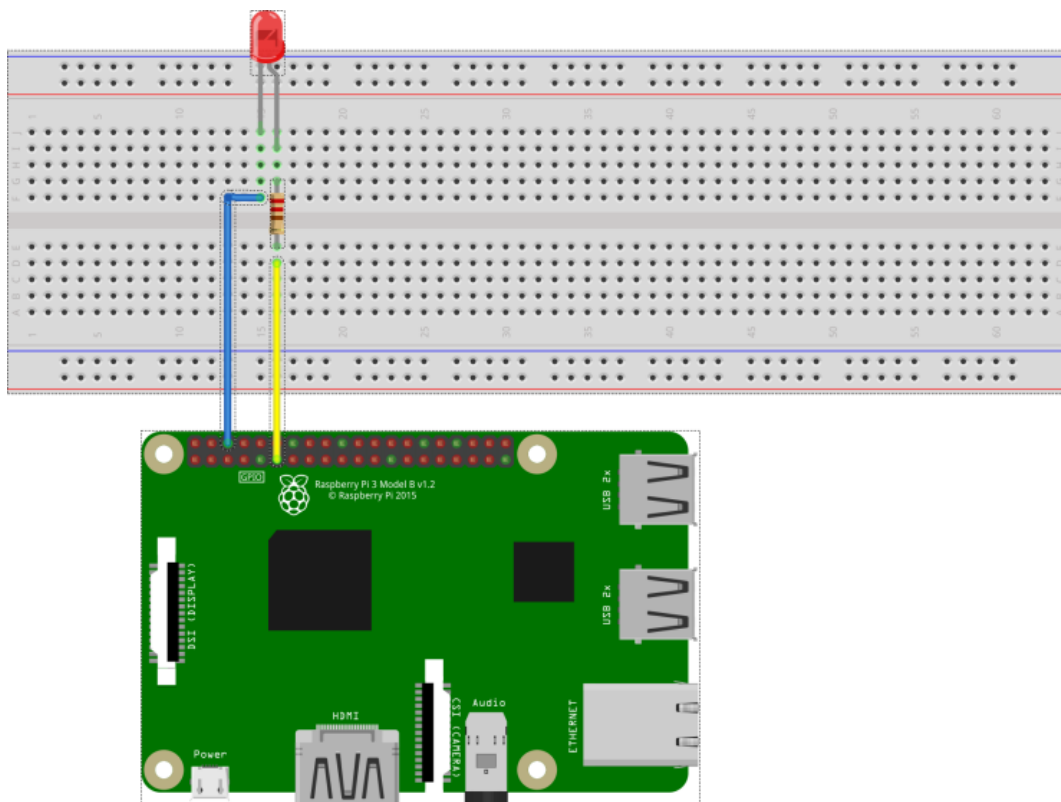
- Si bien aquí la RPI puede ser considerada como una computadora de escritorio ya que bootea con un sistema Ubuntu, la conexión a través de distintas plataformas no es exclusiva de arquitecturas de gran procesamiento. Por ejemplo el interesado podría investigar el siguiente link donde se puede encontrar información de como configurar un nodo sobre un arduino y ejemplos de su aplicación:

http://wiki.ros.org/roserial_arduino/Tutorials

Ejemplo: encendiendo un Led

La RPI dispone de un puerto de entradas y salidas digitales llamado **GPIO**, existen varias bibliotecas con funciones para su manejo. Aquí utilizaremos la biblioteca **Wiring Pi**, esta tiene entre otras cosas funciones de escritura y lectura de pines digitales, comunicación SPI, UART, etc. Para más información de esta librería podemos visitar su web oficial: <http://wiringpi.com/>

Aquí iniciaremos su uso con un ejemplo simple mediante el encendido de un nodo activaremos el parpadeo de un led conectado al pin 11 del puerto GPIO:



fritzing



Para esto deberíamos realizar:

- 1) Crear un espacio de trabajo en la RPI (para las RPI del club ya está hecho, **catkin_workspace**).
- 2) Instalar la librería Wiring Pi en la RPI (para las RPI del club ya está hecho)
Para esto seguir los siguientes pasos (<http://wiringpi.com/download-and-install/>):

```
sudo apt-get purge wiringpi  
hash -r  
sudo apt-get install git-core  
cd  
git clone git://git.drogon.net/wiringPi  
cd ~/wiringPi  
git pull origin  
cd ~/wiringPi  
./build
```

- 3) Crear un nodo que genere el parpadeo de un led:
 - a) Como ya vimos, primero deberemos crear un paquete dentro de la carpeta **src** del espacio de trabajo:

```
catkin_create_pkg cdr_rpi
```

- b) Luego la carpeta **src** dentro de este:

```
cd cdr_rpi  
mkdir src
```



c) Luego creamos el archivo .cpp que contendrá nuestro código:

```
// led.cpp          Parpadeo led en ROS con raspberry pi

// Cargamos libreria ROS
#include <ros/ros.h>
#include <wiringPi.h>

int main(int argc, char **argv)
{
    // inicializamos ROS
    ros::init(argc, argv, "led");
    // establecemos este programa como un node de ROS
    ros::NodeHandle nh;
    // enviamos un mensaje de salida
    ROS_INFO_STREAM("Deberia parpadear");
    wiringPiSetup();          //inicializamos wiring pi
    pinMode(0,OUTPUT);        // definimos pin 0 (de wiring pi) como salida
    while(ros::ok())
    {
        digitalWrite (0, HIGH) ;    // pin 0 a estado alto (3.3V)
        delay (500) ;
        digitalWrite (0, LOW) ;     // pin 0 a estado bajo (0V)
        delay (500) ;
    }
}
```



d) Finalmente agregamos las dependencias en el CMakeList.txt y package.xml

```
cmake_minimum_required(VERSION 2.8.3)
# nombre del paquete
project(cdr_rpi)
# indicamos que busque paquetes de los cuales dependemos
find_package(catkin REQUIRED COMPONENTS roscpp)
#declaramos nuestro paquete catkin
catkin_package()
#especificamos archivos a incluir
include_directories(include ${catkin_INCLUDE_DIRS})
#agregamos ejecutable, los parametros son (nombre_nodo, direccion_archivo.cpp)
add_executable(led src/led.cpp)
#especificamos librerias con las cuales linkear
#target_link_libraries(led ${catkin_LIBRARIES} ${WIRINGPI_LIBRARY})
target_link_libraries(led ${catkin_LIBRARIES} wiringPi )
```

```
<?xml version="1.0"?>
<package>
  <name>cdr_rpi</name>
  <version>0.0.1</version>
  <description>The cdr package</description>
  <maintainer email="alumno@todo.todo">alumno</maintainer>
  <license>TODO</license>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <run_depend>roscpp</run_depend>

</package>
```

4) Finalmente compilamos y ejecutamos (esto lo podemos hacer desde la RPI o bien a distancia mediante SSH).

Algo interesante es la distribución de pines de la **gpio**, su numeración de acuerdo con la librería y sus funciones. Esto lo podemos consultar directamente por terminal con el comando:

```
gpio readall
```

Obteniendo:

```
rpi@rpi-desktop: ~
Archivo Editar Ver Buscar Terminal Ayuda

rpi@rpi-desktop:~$ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2   |  8   |  3.3v    |      |   | 1   | 2   |      |  5v      |     |     | |
| 3   |  9   |  SDA.1    | IN    | 1 | 3   | 4   |      |  5v      |     |     |
| 4   |  7   |  SCL.1    | IN    | 1 | 5   | 6   |      |  0v      |     |     |
|     |     |  GPIO. 7  | IN    | 1 | 7   | 8   | 0   | IN    |  TxD   | 15  | 14  |
|     |     |  0v       |      |   | 9   | 10  | 1   | IN    |  RxD   | 16  | 15  |
| 17  |  0   |  GPIO. 0  | IN    | 0 | 11  | 12  | 0   | IN    |  GPIO. 1 | 1   | 18  |
| 27  |  2   |  GPIO. 2  | IN    | 0 | 13  | 14  | 0   | IN    |  0v     |     |     |
| 22  |  3   |  GPIO. 3  | IN    | 0 | 15  | 16  | 0   | IN    |  GPIO. 4 | 4   | 23  |
|     |     |  3.3v     |      |   | 17  | 18  | 0   | IN    |  GPIO. 5 | 5   | 24  |
| 10  | 12   |  MOSI     | IN    | 0 | 19  | 20  | 0   | IN    |  0v     |     |     |
| 9   | 13   |  MISO     | IN    | 0 | 21  | 22  | 0   | IN    |  GPIO. 6 | 6   | 25  |
| 11  | 14   |  SCLK     | IN    | 0 | 23  | 24  | 1   | IN    |  CE0    | 10  | 8   |
|     |     |  0v       |      |   | 25  | 26  | 1   | IN    |  CE1    | 11  | 7   |
| 0   | 30   |  SDA.0    | IN    | 1 | 27  | 28  | 1   | IN    |  SCL.0  | 31  | 1   |
| 5   | 21   |  GPIO.21  | IN    | 1 | 29  | 30  | 0   | IN    |  0v     |     |     |
| 6   | 22   |  GPIO.22  | IN    | 1 | 31  | 32  | 0   | IN    |  GPIO.26 | 26  | 12  |
| 13  | 23   |  GPIO.23  | IN    | 0 | 33  | 34  | 0   | IN    |  0v     |     |     |
| 19  | 24   |  GPIO.24  | IN    | 0 | 35  | 36  | 0   | IN    |  GPIO.27 | 27  | 16  |
| 26  | 25   |  GPIO.25  | IN    | 0 | 37  | 38  | 0   | IN    |  GPIO.28 | 28  | 20  |
|     |     |  0v       |      |   | 39  | 40  | 0   | IN    |  GPIO.29 | 29  | 21  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
rpi@rpi-desktop:~$
```



Actividad

- 1) Modifique el nodo que prende y apaga el led en la RPI, de forma que se prenda al detectar un '1' en un tópico que usted defina y se apague si se envía cualquier otro dato por el mismo tópico.
Recuerde: ¿este será un programa publicador o suscriptor? ¿si usamos un tipo de mensaje de ROS debemos agregar su librería asociada?
- 2) Cree un nodo en la PC, que publique con una frecuencia de .1 Hz en el tópico del punto 1, haciendo que el led se prenda y apague.
Recuerde: Si ejecutamos el nodo a distancia recuerde seguir los pasos de conexión antes listados.



Ejemplo: tomando una imagen

En este ejemplo veremos como tomar imágenes desde una webcam conecta a la RPI. El objetivo será instalar paquetes pre hechos y adaptarlos para nuestra aplicación.

Para comenzar necesitamos un programa que reconozca la salida de este periférico y la transforme en algo manejable dentro de ROS, buscando en la web un poco encontramos el paquete **usb_cam** el cual está diseñado para la tarea que buscamos. Podemos chequear que nodos existen en este paquete y que tópicos utilizan en: http://wiki.ros.org/usb_cam

Para instalarlo utilizamos: (ya hecho en las RPI del club)

```
sudo apt install ros-kinectic-usb-cam
```



- Existen dos formas genericas de instalar paquetes en ROS:
 - Mediante apt get, que en forma generica podria escribirse como :
`sudo apt install ros-<distribucionderos>-<nombrepaquete>`
 - Mediante git descargando el paquete en nuestro espacio de trabajo, que en forma genérica podría escribirse como:
`git clone <branch> <direcciongit>`
- Las anteriores son formas genéricas y no todos los paquetes funcionaran con ellas, una cuestión importante, sobre todo al usar git, es que los paquetes sean instalados con la estructura correcta en el espacio de trabajo.
- Otro punto a mencionar es que si bien existe cierta portabilidad en los paquetes de ROS, es posible que debamos adaptarlos para su funcionamiento en nuestro código. Lo mismo sucede más frecuentemente con paquetes de una distribución distinta a la que estamos trabajando.

Luego de instalado compilamos nuevamente nuestro espacio de trabajo, y probamos nuestro nuevo paquete mediante:

```
roscore  
roslaunch usb_cam usb_cam_node
```



Podemos chequear que todo funcione mediante **rviz**, agregando una imagen (add->image) y en el campo topic seleccionamos el tópico de salida del nodo usb_cam: **/usb_cam/image_raw**

```
rviz
```

O podemos ejecutar:

```
roslaunch image_view image_view image:=/usb_cam/image_raw
```

Ejemplo: reconociendo contornos de una imagen

Para este ejemplo utilizaremos el paquete **canny_edge_my_face**, el cual puede obtenerse de: https://github.com/tronium/canny_edge_my_face

Si observamos las especificaciones de este paquete vemos que posee un tópico de entrada (de donde tomara las imágenes a analizar):

/cv_camera/image_raw

Y dos de salida (donde mostrara las imágenes de salida):

/camera/canny_edge

/camera/canny_edge_overlay

Para descargar este paquete utilizaremos los comandos de **git**, cuidando de que al descargarlo quede en la forma correcta en el espacio de trabajo. Para el caso de la RPI serían necesarios los siguientes comandos: (esto ya esta hecho en las RPI del club)

```
cd catkin_workspace/src
mkdir canny_edge_my_face      (debemos crear esta carpeta por que el creador del
                               paquete dejo los archivos sueltos, en general esto no es
                               asi)
git clone https://github.com/tronium/canny_edge_my_face.git
```

En este punto podríamos compilar nuestro espacio de trabajo, sin obtener errores, y al ejecutar el nodo correspondiente al paquete descargado nada pasaría. ¿Por qué?

En la configuración que venimos armando el nodo **usb_cam** publica la imagen de la webcam en el tópicos:

/usb_cam/image_raw

Mientras que el nodo **cannyEdge** estará esperando recibir las imágenes a tratar en el tópicos llamado:

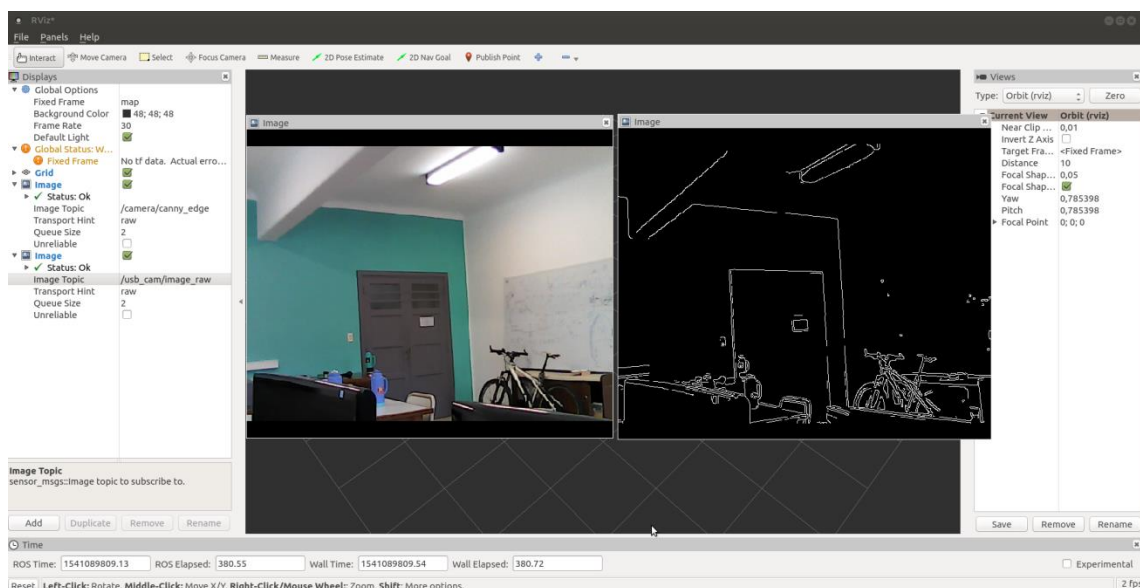
/cv_camera/image_raw

Para solucionar este problema simplemente accediendo al código del nodo **cannyEdge** (catkin_workspace/src/canny_edge_my_face/src/cannyedge.cpp), modificamos el tópicos en el cual debe recibir las imágenes reemplazándolo por el de nuestra aplicación. (esto ya esta hecho en las RPI del club)

Luego de esto compilamos nuestro espacio de trabajo, y podremos ejecutar mediante las siguientes líneas:

```
roscore
roslaunch usb_cam usb_cam_node
roslaunch canny_edge_my_face cannyEdge
rviz
```

Con Rviz podemos agregar varios objetos imagen y observar los tópicos de salida de los distintos nodos. Si todo funciona deberíamos ver algo similar a lo siguiente:





Bibliografía

- A gentle Introduction to ROS. Jason M. O,Kane
- ROS Tutorials. <http://wiki.ros.org/ROS/Tutorials>
- ROS Robotics By Example. Carol Fairchild
- http://wiki.ros.org/usb_cam
- https://github.com/tronium/canny_edge_my_face