

Desempolva esa falda que esta en el closet, y prepárate para usarla!!!

Comenzando con ROS

Arduino Interfaz para Proyectos de Robótica

Introducción a ROS - Arduino Interface

Ya que están usando las boards de Arduino con el profesor Torres, pues, aprovechando y colocándose par de falditas, podrán realizar la siguiente practica (también podrán cambiarse de falditas y usar mbed con alguna otra board más poderosa, como la FRDM-KL25z, BluePill-STM32, PSoC6, LPCXpresso, etc)

Los Arduinos se están convirtiendo en un ingrediente esencial en los robots de bricolaje y los aparatos electrónicos para todos aquellos que no saben programar ni pensar, y buscan una solución rápida a un problema ya establecido y resuelto por alguien más. Algunos boards están basados en microcontroladores ARM disponibles en el mercado que pueden funcionar mejor que los Arduino, no creo que puedan ganarse a los aficionados no pensantes y aficionados al bricolaje push and play.

Centrémonos en la robótica. Ya sabemos que los Arduinos se utilizan en robots de bricolaje, pero ¿crees que estas placas pueden manejar todas las diferentes funciones en un robot complejo? Veamos un ejemplo. Imagina que vamos a construir un robot móvil autónomo que pueda mapear sus alrededores y navegar de forma autónoma. ¿Crees que el Arduino puede hacer este trabajo solo? ¡No! Debido a que Arduino es solo una plataforma de microcontrolador basada en el controlador AVR / ARM (que ejecuta 8 MHz-84 MHz), es básicamente una placa de E / S que puede realizar solo una cantidad mínima de cómputo. No podemos hacer una aplicación de visión de computadora usándola.

Entonces, ¿cómo podemos usar esta placa en un robot de gama alta? Yo diría que podemos usarlo como una placa de E / S básicamente para conectar sensores robóticos (como sensores de sonido ultrasónicos), unidades de medida inercial (IMU) y accionadores como motores de CC, servos, etc. Para realizar un procesamiento de alta calidad en robots, es posible que tengamos que mirar las PC con frameworks ROS para programar robots.

ROS (Sistema operativo de robot) es un frameworks de software de robótica popular para trabajar con robots complejos como PR2, Robonaut, TurtleBot, etc. Estos robots de gama alta tienen toneladas de sensores, por lo que el procesamiento de datos es una tarea incómoda. ROS proporciona un middleware de paso de mensajes (por así decirlo) que se puede usar para comunicarse con diferentes procesos / nodos. Por ejemplo, puede tener un nodo para leer y escribir en un Arduino, y un nodo diferente para obtener imágenes de una cámara. Cada uno de estos nodos puede comunicarse e intercambiar datos entre sí. Ahora la pregunta es ¿cómo conectamos un Arduino a ROS? ¿Cómo intercambiamos datos de sensores y mensajes de control desde Arduino al

frameworks de comunicación de ROS? Antes de discutir la interfaz, es una buena idea repasar algunos conceptos básicos de ROS. Entonces, ¡hagamos precisamente eso!

ROS en pocas palabras

Como se mencionó, ROS es un sistema operativo meta, lo que significa que le brinda funcionalidad, pero necesita un sistema operativo host para ejecutar. Las principales características de ROS son:

- **Middleware de comunicación:** este middleware permite la comunicación entre procesos entre nodos / procesos ROS para intercambiar datos. La comunicación se realiza mediante un mecanismo de publicación / suscripción, es decir, un nodo está enviando datos y otro está recibiendo. Los principales paradigmas de comunicación en ROS son Temas, Mensajes, Servicios y Parámetros.
- **Herramientas:** Cuenta con una amplia variedad de herramientas de línea de comandos y GUI para visualizar y depurar datos ROS. Algunas de las herramientas son Rviz (ROS Visualizer) y rqt.
- **Capacidades:** Además del middleware de comunicación, ROS proporciona una amplia variedad de capacidades que se pueden usar en cualquier robot sin tener un conocimiento profundo de ello. Algunas de las capacidades de ROS son la estimación de pose de robot, localización, mapeo y navegación.
- **Ecosistema:** existe una comunidad mundial activa para el desarrollo y apoyo de ROS.

Realmente empezando

Antes de comenzar con ROS, primero debemos entender su terminología. Echemos un vistazo a las terminologías y conceptos básicos de ROS:

1. **Distribuciones ROS:** al igual que las distribuciones de Linux (por ejemplo, Ubuntu), una distribución ROS es un conjunto de paquetes ROS con versiones. La última distribución de ROS es Kinect Kame. Las versiones anteriores incluían Jade Turtle y Indigo Igloo. El propósito de las distribuciones de ROS es permitir a los desarrolladores trabajar contra una base de código relativamente estable hasta que estén listos para avanzar todo. Si desea explorar las últimas funciones de ROS, puede usar Kinect; Si quieres paquetes estables, usa ROS Indigo.
2. **Sistema operativo compatible:** Como se indicó anteriormente, el frameworks de ROS necesita un sistema operativo host para su funcionamiento; Un sistema operativo favorito para ROS es el mismo Ubuntu. La última versión de ROS es compatible con Ubuntu 16.04 LTS y 15.10. El ROS Indigo es soportado principalmente en Ubuntu 14.04 LTS. Echa un vistazo a este enlace para explorar varias distribuciones de ROS: <http://wiki.ros.org/Distributions> .
3. **Nodos ROS:** Los nodos son procesos que realizan algún tipo de cálculo con la ayuda del frameworks de comunicación ROS. Por ejemplo, un nodo puede

recibir datos de los buscadores de rango láser, y uno puede enviar y recibir datos a un Arduino.

4. **Temas de ROS:** Los temas se *denominan* buses que intercambian datos entre nodos ROS mediante mensajes ROS. El método de envío de datos usando temas se llama publicación; la recepción de datos utilizando temas se llama suscripción. Para una comunicación libre de errores sobre temas ROS, el nodo editor y el nodo suscriptor deben manejar el mismo tipo de mensaje ROS.
5. **Mensajes ROS:** el nodo intercambia datos en forma de mensajes sobre temas. Los mensajes ROS son estructuras de datos que pueden contener tipos primitivos (por ejemplo, enteros, puntos flotantes, booleanos, etc.); arrays también son compatibles. También podemos escribir mensajes personalizados para nuestras aplicaciones.
6. **Servidor de parámetros ROS:** este servidor es un diccionario compartido de múltiples variantes para almacenar principalmente datos estáticos no binarios. Cada nodo puede almacenar y recuperar parámetros a este servidor en tiempo de ejecución. Los parámetros de ROS son visibles a nivel mundial y accesibles a todos los nodos, y se utilizan principalmente para almacenar parámetros de configuración. Esto puede almacenar casi todos los tipos de tipos de datos primitivos y está diseñado principalmente para aplicaciones de bajo ancho de banda.
7. **Servicios ROS:** Los servicios ROS son un tipo de paradigma de comunicación RPC (Llamada a procedimiento remoto) / solicitud. Usando servicios, un nodo proporciona un servicio, y el otro nodo cliente puede llamar a este servicio. El cliente debe esperar hasta que el nodo del proveedor de servicios devuelva el resultado al cliente. Entonces, solo el cliente puede enviar la siguiente solicitud.
8. **Bibliotecas de cliente de ROS:** La biblioteca de cliente de ROS ayuda a crear un nodo de ROS al proporcionar un conjunto de API (interfaces de programación de aplicaciones) para implementar conceptos de ROS, como temas, servicios, parámetros, etc., en nodos. El programador simplemente puede usar las API de la biblioteca cliente para usar los conceptos de ROS. Las principales bibliotecas de clientes son roscpp y rospy. Los programadores pueden usar roscpp junto con su código C ++ también (rospy para programadores de Python). Vaya a este enlace para obtener una lista de las bibliotecas cliente disponibles en ROS: <http://wiki.ros.org/Client%20Libraries> .
9. **Paquetes ROS:** El software en ROS se organiza como paquetes. Cada paquete puede contener nodos, archivos de configuración, software de terceros o cualquier cosa que podamos llamar como un módulo. El paquete en ROS hace que el código sea reutilizable, modular y fácil de distribuir. Los meta paquetes de ROS son un tipo de paquete virtual que significa que no tiene nodos u otros archivos, excepto *package.xml* y *manifest.xml* . Esta es una referencia para uno o más paquetes que pertenecen a un grupo en particular.

10. **Sistema de compilación ROS:** El sistema de compilación es responsable de compilar el ejecutable o la biblioteca de destino desde el código fuente sin procesar dentro de un paquete ROS. El sistema de construcción utilizado en las nuevas distribuciones de ROS es *catkin*. Catkin usa macros CMake y script Python para hacer la construcción del código fuente dentro de un paquete.
11. **Área de trabajo ROS:** El área de trabajo es un lugar donde organizamos los paquetes ROS. Podemos crear, modificar e instalar un paquete ROS desde un espacio de trabajo ROS. Tenemos que establecer un espacio de trabajo de catkin antes de crear un paquete basado en catkin.

Cómo conectar un dispositivo serie (como un Arduino) a ROS

En la mayoría de los robots, los sensores básicos se conectarán a placas de E / S como Arduino, STM32 o Tiva C Launchpad. Entonces, ¿cómo podemos alimentar los datos de este sensor a ROS que se ejecuta en una PC?

Esta es la función de los paquetes *rosteriales* (<http://wiki.ros.org/rosterial>). El meta paquete *rosterial* consiste en conjuntos de paquetes para recibir datos en serie de un microcontrolador o cualquier otro dispositivo en serie utilizando un protocolo *rosterial* estándar. Así es como funciona el protocolo *rosterial*.

Roserial es un protocolo estándar para la comunicación entre ROS y un dispositivo serie. La comunicación se realiza a través de una línea de transmisión en serie y utiliza técnicas de serialización / des-serialización para transmitir mensajes ROS. El dispositivo serie está enviando mensajes ROS como un paquete que tiene un encabezado y una cola que permiten múltiples temas y servicios desde un solo dispositivo de hardware. El paquete también contiene indicadores para sincronizar la comunicación entre la PC y el dispositivo, y viceversa. **La Figura 1** muestra el formato del paquete utilizando el protocolo *rosterial*.

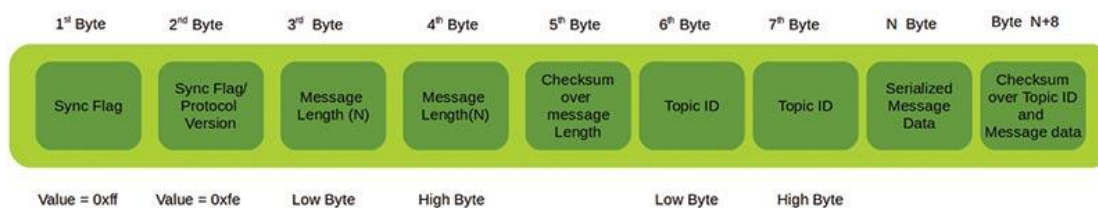


Figura 1. Estructura del paquete de Roserial.

El primer byte se llama *Sync Flag*, que se usa para sincronizar la comunicación entre ROS y el dispositivo. Su valor será siempre *0xff*. El segundo byte o la *Versión de protocolo* / Indicador de *sincronización* le indica si la versión de ROS que estamos utilizando es anterior a ROS Groovy o posterior a Groovy. El valor será *0xfe* después de ROS Groovy y será *0xff* hasta Groovy. Los bytes tercero y cuarto representan la longitud del mensaje que está en el paquete, y el quinto byte es una suma de comprobación de la longitud del mensaje. Los bytes sexto y séptimo están dedicados a la identificación del tema. El ID de tema de 0-100 está reservado para las funciones del sistema. Los bytes restantes se utilizan para datos en serie y sus sumas de comprobación.

Las sumas de comprobación de la longitud del paquete y los datos se calculan utilizando la siguiente ecuación:

$$255 - ((ID \text{ de tema Low Byte} + ID \text{ de tema High Byte} + \text{valores de bytes de datos}) \% 256)$$

La comunicación entre el dispositivo serie y la PC se iniciará desde el lado de la PC, que enviará un paquete de consulta para obtener la cantidad de temas, nombres y tipos de temas desde el lado del dispositivo Arduino / serie. Cuando el Arduino obtiene este paquete de consulta, responderá a la PC con una serie de paquetes de respuesta. El paquete de respuesta constará de las siguientes entradas:

uint16 topic_id
string nombre_tema
string message_type
cadena md5sum
int32 buffer_size

Esta serie de respuestas estará en los mensajes `rosterial_msgs / TopicInfo`. Cuando una respuesta no recibe respuesta del dispositivo serie, la consulta se reenviará desde el lado de la PC. La sincronización entre ROS y el dispositivo serie se maneja al enviar información de tiempo desde la PC al dispositivo en particular.

Las bibliotecas `rosterial_client`

Las bibliotecas `rosterial_client` (http://wiki.ros.org/rosterial_client) tienen una implementación del lado del cliente del protocolo `rosterial`. El cliente puede ser una plataforma de microcontrolador integrada, como con un Arduino, ARM u otro dispositivo serie, como Xbee. Puede ejecutarse en cualquier procesador que tenga un compilador ANSI C ++ y una interfaz serial a una computadora que ejecute ROS. Hay varios paquetes de bibliotecas `rosterial_client` por ahí para plataformas específicas. Aquí están algunas de esas bibliotecas `rosterial_client` :

- **`rosterial_arduino`:** este paquete te ayuda a construir una biblioteca Arduino llamada *ros_lib* que puede funcionar como una biblioteca `rosterial_client` para Arduino. Podemos incluir esta biblioteca para escribir nodos de clientes Arduino-ROS. Este paquete nos ayuda a crear un cliente `rosterial` en la mayoría de las plataformas Arduino, especialmente en Uno, Mega y Leonardo.
- **`rosterial_embeddedlinux`:** este paquete nos permite ejecutar `rosterial_client` en sistemas Linux integrados (como enrutadores, etc.).
- **`rosterial_windows`:** ROS no está completamente portado a Windows. Si tenemos que comunicarnos con cualquier aplicación de Windows, podemos usar este paquete para comunicarnos desde el cliente de Windows a ROS.
- **`rosterial_mbed`:** la mayoría de las placas basadas en controladores ARM admiten la programación *mbed* (capa de abstracción de hardware), que es similar a la programación de Arduino. Podemos programar tableros *mbed* utilizando un compilador en línea o fuera de línea usando un proyecto *gcc4embed* (<https://github.com/adamgreen/gcc4mbed>). El compilador fuera de línea solo puede funcionar con un número limitado de tableros, pero el compilador en línea puede usar tableros nuevos que admitan *mbed* .

- **rosserial_tivac:** El Tiva C Launchpad es otra placa controladora de Texas Instruments que se ejecuta en 80 a 120 MHz. Las últimas placas son 123GXL y 129XL, y admiten el IDE (entorno de desarrollo integrado) similar a Arduino llamado Energia (<http://energia.nu>). Podemos programar Tiva C utilizando el lenguaje de programación Arduino llamado Wiring (<http://wiring.org.co>).

Después de escribir el nodo rosserial en un dispositivo serie, tenemos que ejecutar otro nodo en el lado de la PC para codificar / decodificar los datos serie. Hay algunos paquetes para hacer este proceso. Esos son:

- **rosserial_python:** este paquete se usa comúnmente para la comunicación rosserial. Tiene un nodo Python que puede conectarse al dispositivo serie y recibir valores serie del dispositivo. Descodificará los paquetes seriales y generará temas y servicios de ROS que estarán disponibles en el lado de la PC. Este paquete es estable y recomendado para aplicaciones de PC.
- **rosserial_server:** esta es en realidad una implementación en C ++ de un receptor rosserial y se puede usar para aplicaciones de alto rendimiento. Las características de este nodo son menos en comparación con *rosserial_python* .
- **rosserial_java:** Si desea interconectar ROS con un Android, esta es la mejor opción.

Vamos a utilizar *rosserial_python* en el lado de la PC.

Componentes Requeridos

Estos son los dos componentes de hardware que necesitamos para comenzar con nuestra programación Arduino-ROS:

- Arduino Mega 2560 (recuerda que si sabes programar, cualquier placa puede funcionar)
- Cable USB tipo B (C, D, F, G, dependiendo de tu board en caso de cambiar)

Para comenzar a conectar nuestro Arduino a ROS, el primer procedimiento es instalar ROS en la PC.

Instalando ROS en Ubuntu 16.04 LTS

Esto ya lo hiciste, por lo que puedes bincar este paso, pero si eres de los que ya llevan 2 meses sin hacer nada, y solo trabajando en las practicas de la maestra Razo, pues, a empezar a trabajar, y realizar este procedimiento.

Para trabajar con un Arduino, podemos configurar una nueva distribución de ROS. También te daré las instrucciones para configurar ROS Indigo en Ubuntu 14.04 LTS.

Puede instalar Ubuntu en una PC real o VirtualBox (si desea trabajar desde Windows).

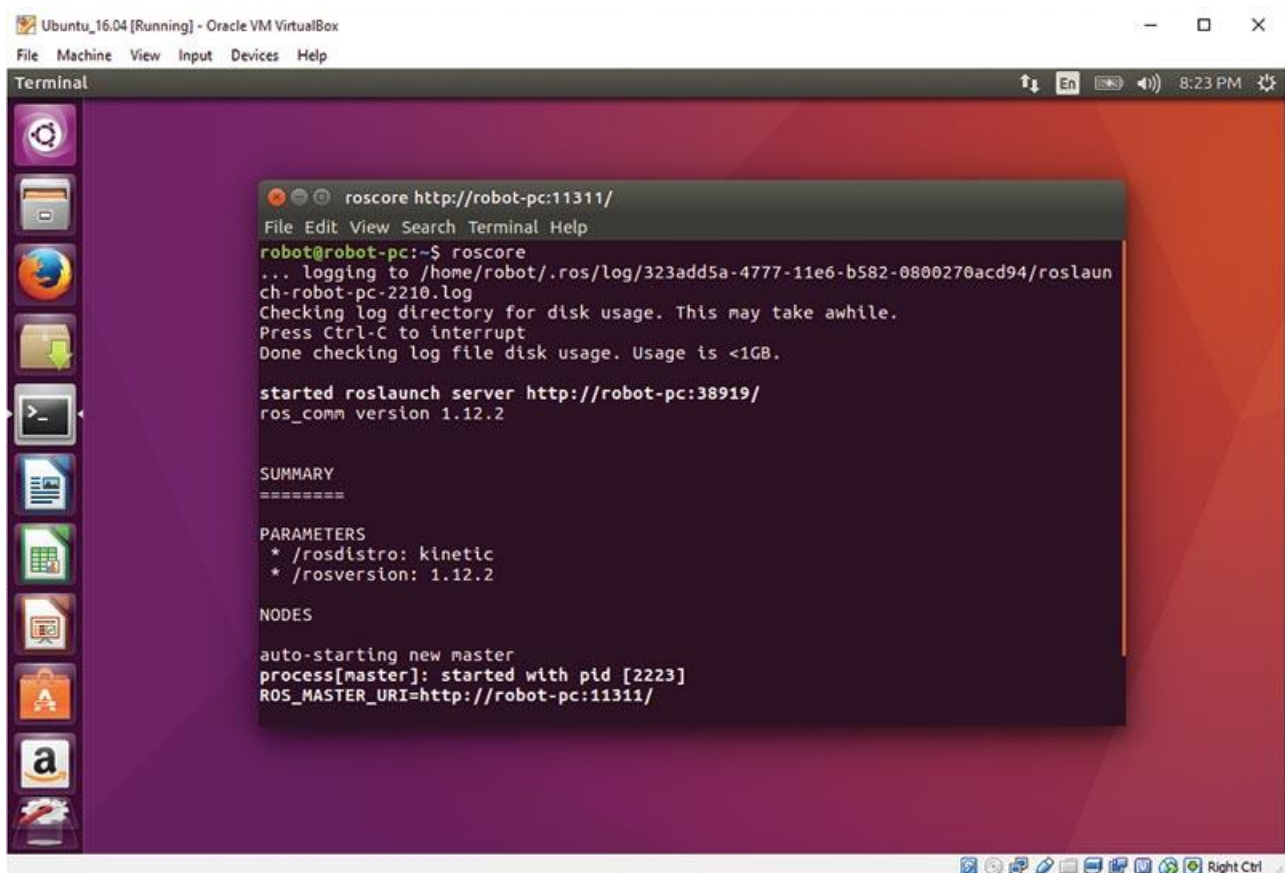
Estos son los procedimientos para configurar Ubuntu 16.04 LTS y ROS Kinect Kame:

- Descargue Ubuntu 16.04 LTS desde <http://releases.ubuntu.com/16.04> grabelo en un disco o escríbalo en una unidad USB usando *unetbootin* (

<https://unetbootin.github.io>) para instalarlo en una PC. También puede instalar Ubuntu en VirtualBox (<https://www.virtualbox.org/wiki/Downloads>). Los tutoriales completos de instalación de Ubuntu están disponibles en www.ubuntu.com/download/desktop/install-ubuntu-desktop. Tenga en cuenta que en la configuración de VirtualBox, la memoria de video debe ser alta, y debe instalar los Complementos de VirtualBox Guest después de Instalación de Ubuntu.

- A continuación, instale ROS en Ubuntu. Para obtener instrucciones detalladas sobre cómo instalar ROS Kinect en Ubuntu 16.04 desde el sitio web de ROS, vaya a <http://wiki.ros.org/kinetic/Installation/Ubuntu> . Si está interesado en ROS Indigo, utilice en su lugar <http://wiki.ros.org/indigo/Installation/Ubuntu> . También tenga en cuenta que, en lugar de Ubuntu 16.04, necesita 14.04 LTS para instalar ROS Indigo.
- Necesitará la instalación de escritorio ROS-Kinetic-Full para este proyecto.

Una vez que se complete la instalación de ROS, ejecute el comando `roscore` para verificar que todo funciona bien. Una captura de pantalla de este comando se muestra en la **Figura 2** .



```
Ubuntu_16.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
roscore http://robot-pc:11311/
File Edit View Search Terminal Help
robot@robot-pc:~$ roscore
... logging to /home/robot/.ros/log/323add5a-4777-11e6-b582-0800270acd94/roslaunch-robot-pc-2210.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://robot-pc:38919/
ros_comm version 1.12.2

SUMMARY
=====
PARAMETERS
* /rostdistro: kinetic
* /rosversion: 1.12.2

NODES

auto-starting new master
process[master]: started with pid [2223]
ROS_MASTER_URI=http://robot-pc:11311/
```

Figura 2. Ejecutando roscore.

Instalación del paquete de interfaz Rosserial en Ubuntu

Después de configurar ROS en Ubuntu, tenemos que instalar los paquetes rosseriales en ROS. La instalación de un paquete se puede hacer de dos maneras:

- 11....1. Instalación a través de *apt-get* que instalará binarios pre-construidos.
- 11....2. Instalación mediante compilación de código fuente.

La forma más fácil de instalar paquetes es usar *apt-get*, pero en las últimas versiones de ROS, es posible que la mayoría de los paquetes no estén disponibles como binarios. En ese caso, podemos crear un espacio de trabajo ROS y descargar los paquetes de origen e instalarlo. Aquí está el procedimiento para construir un paquete *rosserial* en ROS Kinect:

1. En su terminal, cree una carpeta vacía para el área de trabajo de ROS. Puedes darle cualquier nombre; Estoy usando **rosserial_ws**:

```
$ mkdir -p ~/rosserial_ws/src //Creating a folder called rosserial_ws, and src folder inside the workspace folder<br /> $ cd ~/rosserial_ws/src //Switch to src folder<br /> $ catkin_init_workspace //This will initialize a catkin workspace<br /> $ git clone [url=https://github.com/ros-drivers/rosserial ]https://github.com/ros-drivers/rosserial [/url]; //Cloning latest source code of serial package in src folder<br /> $ cd ~/rosserial_ws //Change into workspace folder<br /> $ catkin_make //Command to build the entire workspace
```

2. El comando *catkin_make* compilará todos los paquetes dentro del área de trabajo y generará carpetas adicionales como ' *build* ' y ' *devel* '. La carpeta de *compilación* contiene registros de compilación y la carpeta *devel* contiene scripts de shell y ejecutables generados. Para hacer que este paquete sea visible para el entorno de ROS, debe crear uno de los scripts de shell en el *devel* . El siguiente comando hará este trabajo:

```
$ echo "source ~/rosserial_ws/devel/setup.bash" >> ~/.bashrc<br /> $ source ~/.bashrc
```

¡¡Felicidades!! Ya ha terminado con la instalación de origen de los paquetes *rosserial*es !! Si está intentando instalar usando *apt-get* , puede usar este comando:

```
$ sudo apt-get install ros-kinectic-rosserial
```

Nota: Si falla, cambie a la instalación de origen. Si está trabajando con ROS-Indigo, puede instalarlo directamente usando el siguiente comando:

```
$ sudo apt-get install ros-indigo-rosserial
```

Configuración de la biblioteca cliente Arduino IDE y ROS

En esta sección, vamos a configurar el IDE de Arduino en Ubuntu 16.04. Arduinos se pueden programar fácilmente usando su simple IDE. Este IDE se puede descargar desde el sitio web de Arduino y está disponible para plataformas de SO populares. Vaya a <https://www.arduino.cc/en/Main/Software> . Podemos descargar el IDE desde este sitio web o instalarlo a través del administrador de paquetes de software de Ubuntu. La descarga directamente desde el sitio web le dará la versión más reciente.

Para instalar el IDE de Arduino desde el administrador de paquetes de software de Ubuntu, use este comando:

```
$ sudo apt-get install arduino
```


Después de la instalación, puede ejecutar el comando `arduino` en su terminal, o si está trabajando con binarios descargados del sitio web, extraiga el archivo y podrá ver un ejecutable llamado *arduino* . Simplemente ejecútalo usando este comando:

```
$ ./arduino
```

Si todo funciona bien, aparecerá el IDE de Arduino y verá la pantalla en la **Figura 3** .

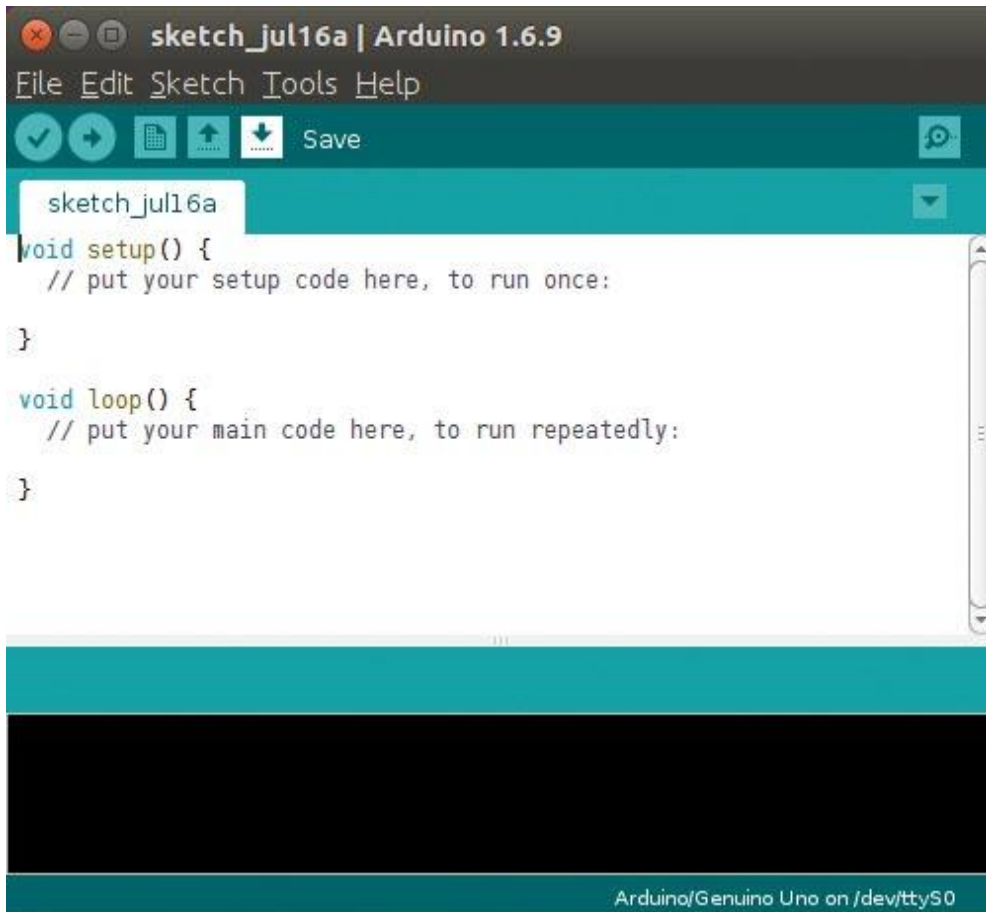


Figura 3. Arduino IDE.

¡Felicidades! Has configurado con éxito el IDE de Arduino en Linux. Nuestro siguiente procedimiento es configurar la biblioteca de Arduino rosserial.

Configurando `ros_lib` en el IDE de Arduino

Después de configurar el IDE de Arduino, tenemos que crear una biblioteca Arduino-ROS para escribir nodos Arduino-ROS. Aquí están los pasos para configurarlo:

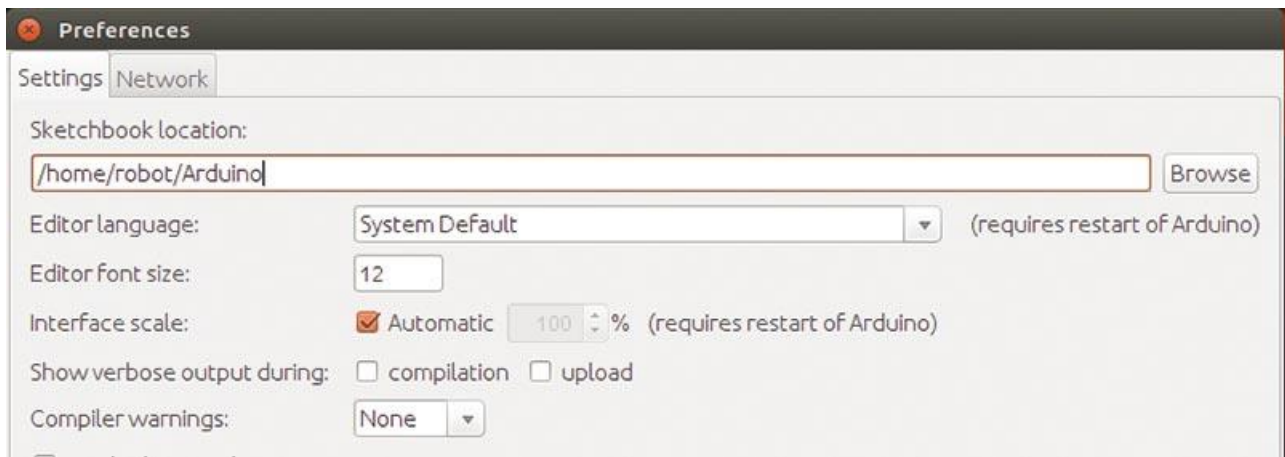


Figura 4. Preferencia de Arduino.

- 11....2.1. Vaya a *Archivo -> Preferencia* en el IDE de Arduino y busque la ubicación del cuaderno de bocetos (**Figura 4**). Vaya a la ubicación del cuaderno de bocetos y busque la carpeta llamada *bibliotecas* . Si no está allí, puede crear uno nuevo. Esta es la ubicación donde vamos a crear la biblioteca Arduino-ROS.
- 11....2.2. Para construir la biblioteca Arduino-ROS, abra una nueva terminal y ejecute:
\$ roscore
- 11....2.3. Abra un nuevo terminal en *arduino_sketchbook_folder* /*raries* e ingrese el comando que se muestra en la **Figura 5** . Este comando generará la biblioteca *ros_lib* , que consiste en mensajes equivalentes incrustados de mensajes ROS reales y API de cliente serie ROS. **Nota:** *Puede mostrar un error durante la generación. Simplemente ignora los mensajes de error. Para algunos mensajes ROS, las conversiones equivalentes integradas pueden no ser posibles.*

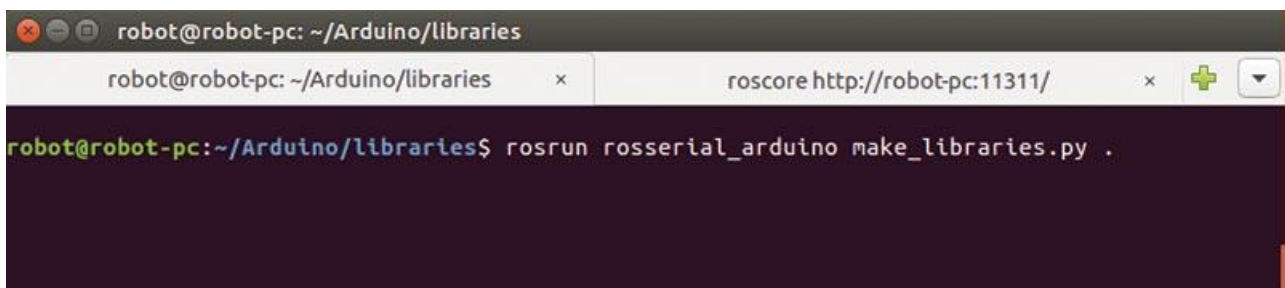


Figura 5. Generando Arduino *ros_lib*.

Si *ros_lib* ahora está correctamente en la carpeta, puede obtener los ejemplos de *ros_lib* de la siguiente opción (consulte la **Figura 6**):

Archivo -> Ejemplos -> ros_lib

Ahora, podemos trabajar en ejemplos simples usándolo. Podemos comenzar con un código *Blink LED* . Usando este ejemplo, podemos cambiar el estado del LED cada vez que publicamos un valor en un tema. Así es como lo hacemos.

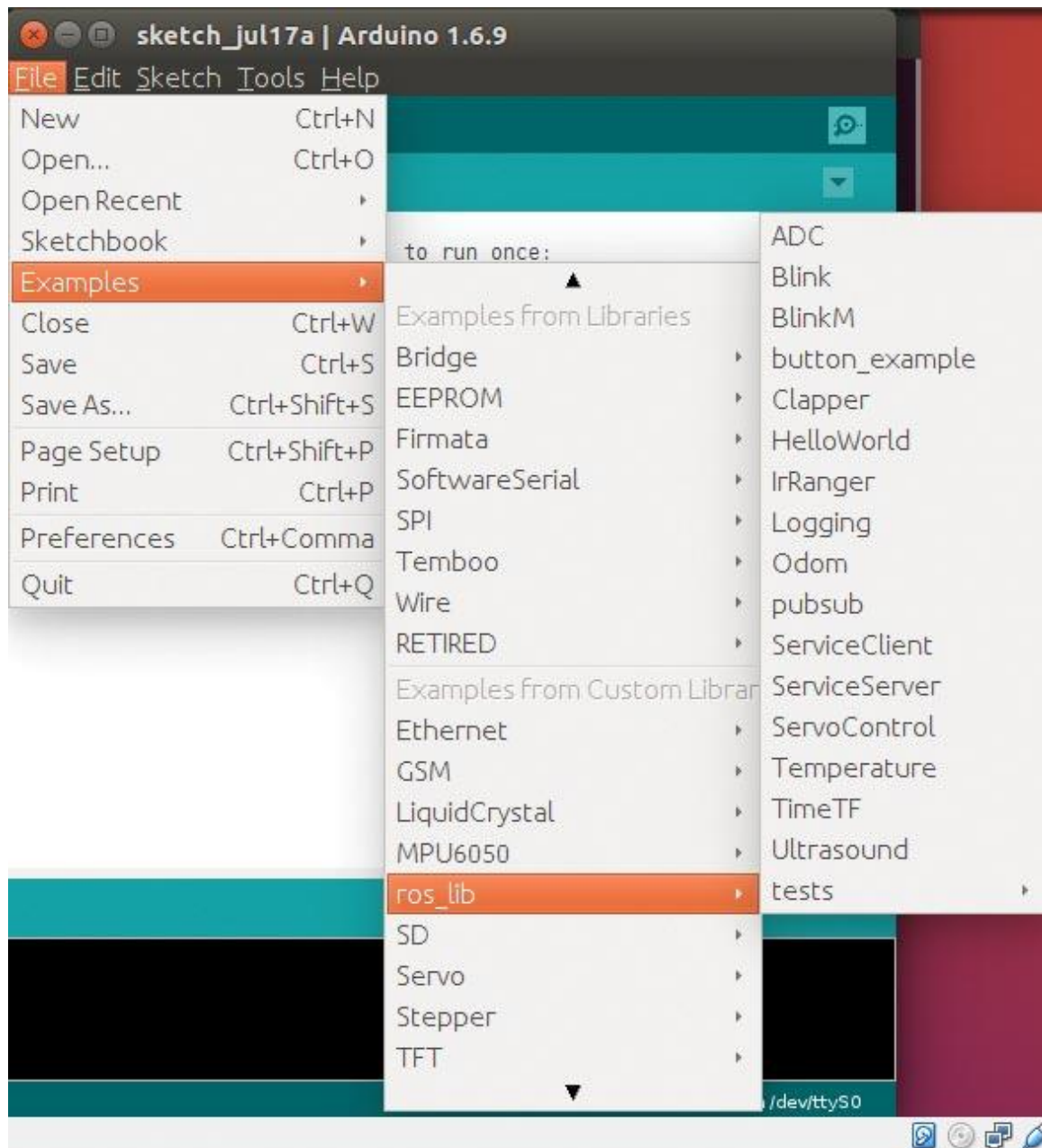


Figura 6. Arduino ros_lib ejemplos.

Configuración del nombre de la placa y el dispositivo serie en el IDE de Arduino

Puedes programar Arduino en el sistema Ubuntu real o VirtualBox. Si está trabajando con una PC, simplemente conecte el dispositivo Arduino y cuando conecte el Arduino en Ubuntu, actuará como un dispositivo serie y el kernel de Linux cargará el controlador USB a serie.

Use un terminal de Linux y ejecute `dmesg` o `dmesg | grep tty` . El `dmesg` mostrará los registros del kernel y podemos encontrar el dispositivo serial correspondiente al Arduino. La salida de `dmesg & dmesg | grep tty` se muestra en la **Figura 7** y la **Figura 8** .

```
[ 264.900785] usb 1-1.2: new full-speed USB device number 4 using ehci-pci
[ 264.997219] usb 1-1.2: New USB device found, idVendor=2341, idProduct=0042
[ 264.997230] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber
=220
[ 264.997235] usb 1-1.2: Manufacturer: Arduino (www.arduino.cc)
[ 264.997240] usb 1-1.2: SerialNumber: 753303039343516102F0
[ 265.049964] cdc_acm 1-1.2:1.0: ttyACM0: USB ACM device
[ 265.050611] usbcore: registered new interface driver cdc_acm
[ 265.050619] cdc_acm: USB Abstract Control Model driver for USB modems and ISD
N adapters
```

Figura 7. Mensajes del kernel de Linux.

```
lentin@lentin-Aspire-4755:~$ dmesg | grep tty
[ 0.000000] console [tty0] enabled
[ 265.049964] cdc_acm 1-1.2:1.0: ttyACM0: USB ACM device
lentin@lentin-Aspire-4755:~$
```

Figura 8. Mensajes Dmesg y grep.

El nombre del dispositivo para el Arduino es `ttyACM0` . Puede ser diferente para cada sistema.

ROS ejecutándose en VirtualBox desde Windows

Si está trabajando desde Windows y VirtualBox, primero debe instalar el controlador adecuado de Arduino para Windows. Los controladores están disponibles en la carpeta de instalación de Arduino. Es posible que deba descargar "Arduino para Windows" para obtener este controlador. Después de instalar el controlador, use el Administrador de *dispositivos* de Windows para verificar el número de puerto serie del Arduino. Si todo está configurado, debería verse como en la **Figura 9** .



Figura 9. Administrador de dispositivos de Windows.

Solo verás esto si los controladores Arduino están correctamente instalados. Después de configurar los controladores, puede compartir este dispositivo serie en VirtualBox.

Inicie VirtualBox y acceda a *Configuración -> Puerto serie* . Comparta el puerto real en el SO huésped como se muestra en la **Figura 10** .

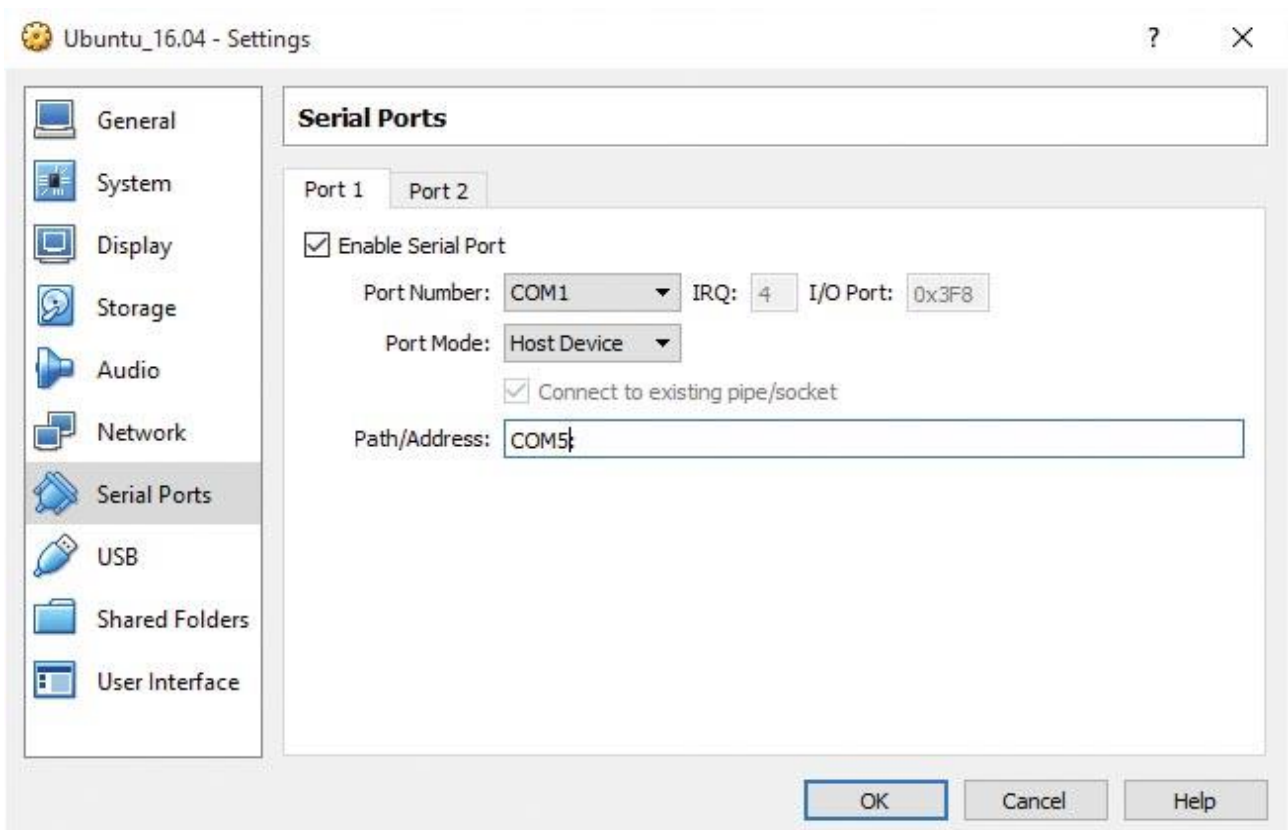


Figura 10. Compartiendo un puerto serie en VirtualBox.

Aquí, el dispositivo real es COM5 :. En el dispositivo invitado, será COM1, que corresponde a *ttyS0* en Linux.

Configuración del nombre de la placa y el puerto en el IDE de Arduino que se ejecuta en VirtualBox

Después de obtener el número de puerto serial del Arduino, deberíamos establecer estos parámetros dentro del IDE de Arduino. Así es como lo hacemos.

Para configurar el nombre del tablero, vaya a las *Herramientas del menú IDE -> Tablero -> Arduino Mega 2560* . También configure el nombre del puerto en *Herramientas -> Puerto -> ttyS0* .

Si está trabajando desde una PC, el puerto será *ttyACMO* . Si trabaja desde VirtualBox, recuerde que el puerto que asignamos fue *ttyS0* . Pruebe la conexión entre el Arduino y VirtualBox mediante la grabación de un boceto de muestra llamado *Blink* :

Archivo -> Ejemplos -> Conceptos básicos -> Parpadeo

Cargue el boceto, compílelo y cárguelo en el tablero.

Nota: Si obtiene un error de permiso denegado o algo similar, puede hacer la siguiente solución para resolverlo.

Estos comandos nos permitirán agregar un usuario actual a un grupo llamado *dialout* . Este usuario de grupo puede acceder a un dispositivo serie sin permiso:

```
$ sudo usermod -a -G dialout <username>
```

Para establecer el permiso de lectura / escritura en el dispositivo serie existente, use esto:

```
$ sudo chmod 666 /dev/ttyACMO or ttyS0
```

Si todo funciona bien, cargue un código de cliente Arduino-ROS para parpadear un LED en la placa Arduino. Lo obtendrá desde *Archivo -> Ejemplo -> ros_lib -> Parpadear* .

Primer proyecto: parpadea un LED

```
/*<br /> * roserial Subscriber Example<br /> * Blinks an LED on callback<br /> */  
  
// Arduino – ROS headers<br /> #include <ros.h><br /> #include <std_msgs/Empty.h>  
  
//Creating a Nodehandle object<br /> ros::NodeHandle nh;  
  
// Creating a callback for the topic toggle_led, whenever a value come through this topic, this  
callback will execute<br /> // The callback will toggle the state of LED which is on PIN 13  
  
void messageCb( const std_msgs::Empty& toggle_msg){<br /> /> digitalWrite(13,  
HIGH-digitalRead(13)); // blink the led<br /> }  
  
//Creating a subscriber with a name toggle_led, and its callback<br />  
ros::Subscriber<std_msgs::Empty> sub(“toggle_led”, &messageCb );
```



```
//Setting PIN 13 as output and initializing ROS node and subscriber object<br /> void setup()<br /> {<br /> pinMode(13, OUTPUT);<br /> nh.initNode();<br /> nh.subscribe(sub);<br /> }<br /> // Spining the node each times to listen from the topic<br /> void loop()<br /> {<br /> nh.spinOnce();<br /> delay(1);<br /> }
```

En este código, estamos creando un nodo ROS de suscriptor dentro de Arduino, que escuchará un tema llamado *toggle_led* . Cada vez que un valor publica este tema, el LED cambiará su estado.

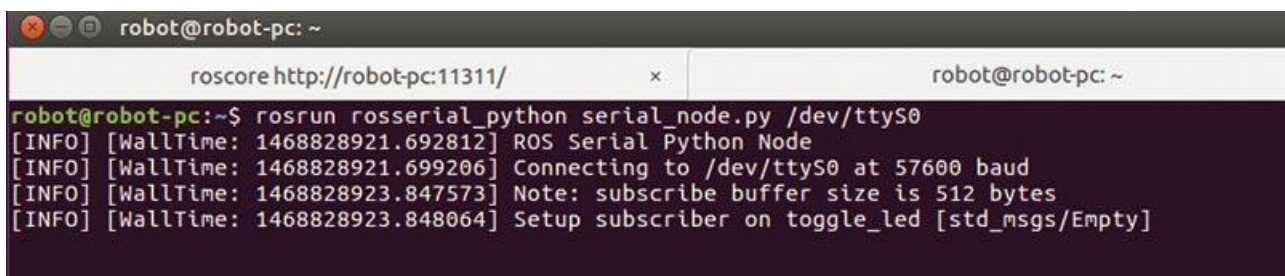
Suba este código fuente al Arduino y veamos cómo publicar valores en este tema. Primero, ve a una terminal y ejecuta:

```
$ roscore
```

Ejecute el nodo roserial en el lado de la PC para codificar / decodificar mensajes de Arduino:

```
$ rosrund roserial_python serial_node.py <serial_dev_name, eg: /dev/ttyACM0 )
```

Si este comando está bien, recibirá un mensaje como el que se muestra en la **Figura 11** .



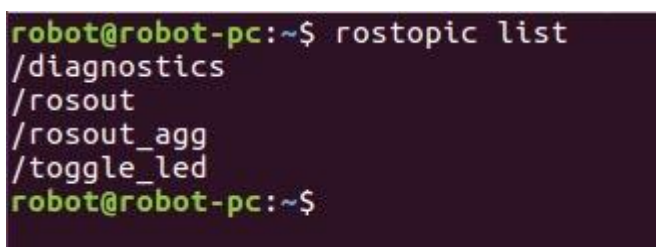
```
robot@robot-pc: ~
roscore http://robot-pc:11311/ x robot@robot-pc: ~
robot@robot-pc:~$ rosrund roserial_python serial_node.py /dev/ttyS0
[INFO] [WallTime: 1468828921.692812] ROS Serial Python Node
[INFO] [WallTime: 1468828921.699206] Connecting to /dev/ttyS0 at 57600 baud
[INFO] [WallTime: 1468828923.847573] Note: subscribe buffer size is 512 bytes
[INFO] [WallTime: 1468828923.848064] Setup subscriber on toggle_led [std_msgs/Empty]
```

Figura 11. Nodo serial ROS-Python.

Si el nodo está funcionando bien, ingrese el siguiente comando:

```
$ rostopic list
```

Se enumerarán los temas en el sistema ROS. Si todo es correcto, puede obtener una salida que se parece a la **Figura 12** .



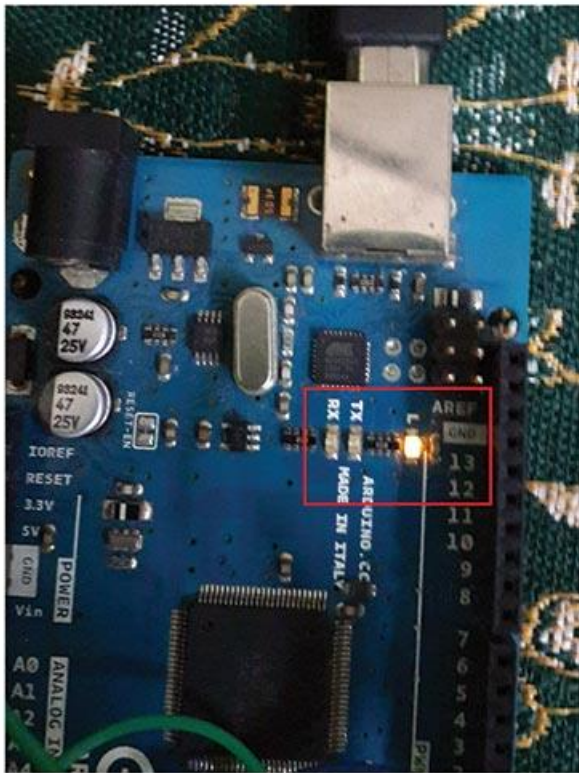
```
robot@robot-pc:~$ rostopic list
/diagnostics
/rosout
/rosout_agg
/toggle_led
robot@robot-pc:~$
```

Figura 12. Lista de temas de ROS.

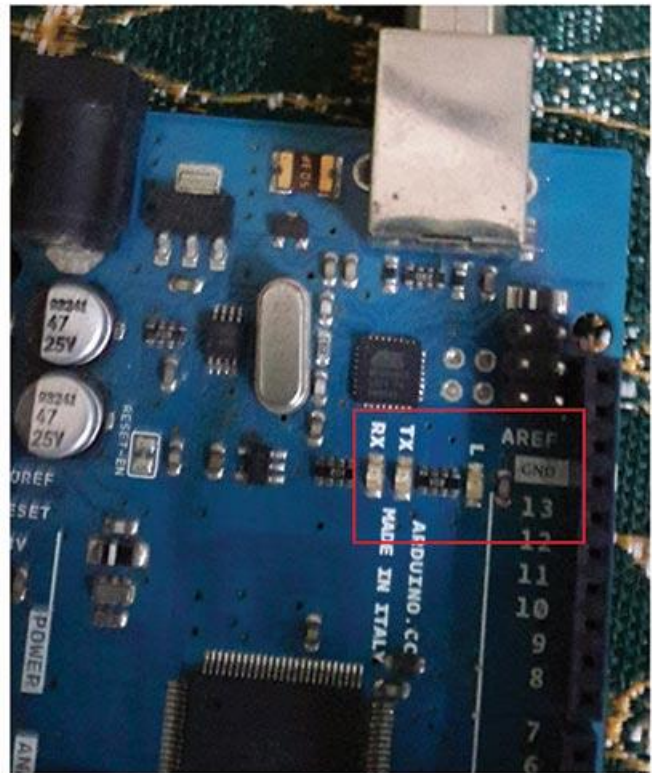
¡Sí! ¡Lo hicimos! Conseguimos el tema del cliente Arduino-ROS. Ahora, para publicar un valor para este tema, use este comando:

```
$ rostopic pub toggle_led std_msgs/Empty -once
```

Eche un vistazo a la **Figura 13** .



LED ON



LED OFF

Figura 13. Interruptor LED.

Puede encontrar más ejemplos de interfaces ROS-Arduino en http://wiki.ros.org/roserial_arduino/Tutorials.

Interfaz de un MPU-9150 en un Arduino usando ROS Kinetic

A continuación, discutiremos un ejemplo más complicado usando ROS y un Arduino. En este ejemplo, vamos a conectar una IMU (unidad de medición inercial) llamada MPU-9150 a una Arduino Mega 2560, y visualizar sus valores en Rviz.

Los componentes requeridos incluyen:

- Arduino Mega 2560
- Cable USB tipo B
- MPU-9150 junta de ruptura
- Cabezales de pin de ruptura
- Arduino cables de puente macho a hembra

Concepto de proyecto

El MPU-9150 es una IMU popular de una compañía llamada InvenSense (www.invensense.com). Tiene un acelerómetro incorporado, giroscopio y magnetómetro (pero puede usarse cualquier otro sensor para la practica). También tiene una unidad de procesamiento de movimiento llamada DMP (**Procesamiento de movimiento digital**) para fusionar los tres datos del sensor para obtener

información de orientación precisa. El MPU-9150 puede conectarse fácilmente a un Arduino utilizando la biblioteca Arduino.

Después de recuperar la información de orientación de los sensores, la enviará a ROS a través de la interfaz ROS-Arduino. El cliente Arduino-ROS publicará un mensaje IMU ROS y también publicará información de Transformación (TF). Los datos de TF se pueden visualizar directamente en Rviz.

Soldadura MPU-9150 Pin Headers

Cuando compre el MPU-9150, obtendrá sus encabezados de pines que deben soldarse al tablero de conexiones. **La figura 14** muestra cómo se sueldan los cabezales de los pines. Estos pines se pueden conectar a un Arduino utilizando los cables de puente macho a hembra.

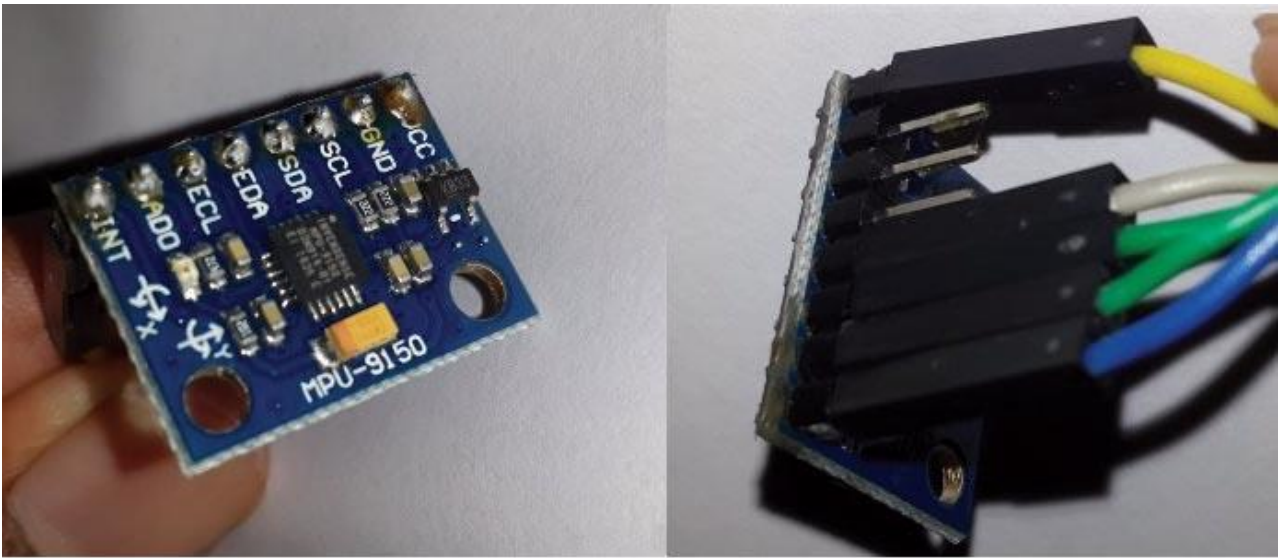


Figura 14. Tablero de arranque del MPU-9150.

Debe tener mucho cuidado al soldar porque el calor excesivo puede dañar el IC principal en la placa. Debe mantener la alineación del cabezal del perno perpendicular a la placa durante la soldadura. Si la alineación de los pines no es correcta, no podemos montar la IMU en una placa de pruebas.

Después de soldar, podemos conectar el módulo IMU utilizando los cables de puente M / F. El conector hembra se puede usar para conectar la IMU, y el conector macho se puede usar para el Arduino. **La Figura 15** muestra el circuito de la IMU al Arduino 2560.

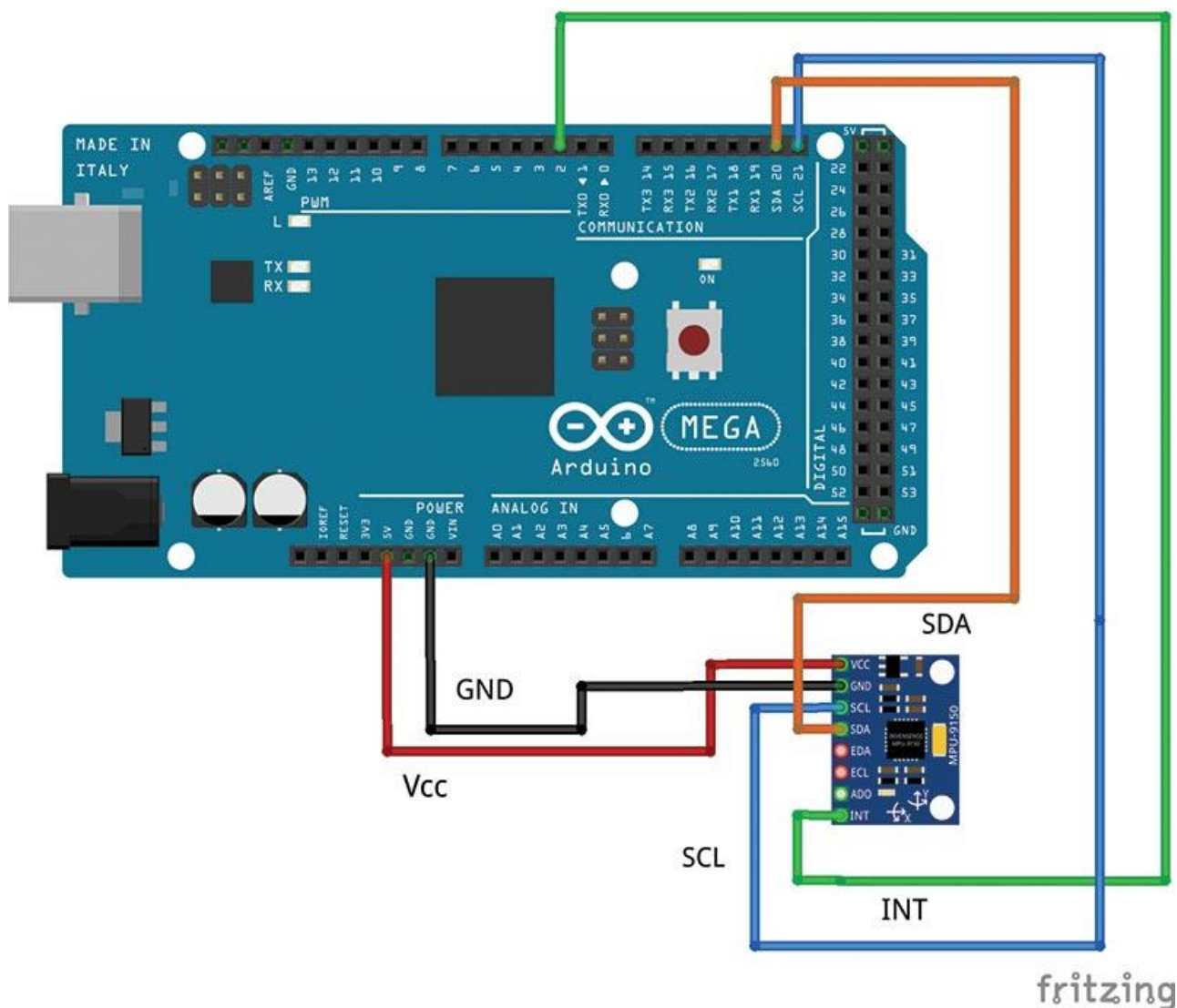


Figura 15. Circuito de interconexión MPU-9150 Arduino.

Programando el Arduino

Para comenzar a programar con el Arduino, primero necesitamos descargar la biblioteca Arduino para el MPU-9150. Podemos descargarlo desde https://github.com/sparkfun/MPU-9150_Breakout . Copie el contenido de la carpeta de firmware en la carpeta `<arduino_sketchbook_location> /libraries` . Abra el IDE de Arduino; si todo está configurado, debería obtener lo que ve en la **Figura 16** .

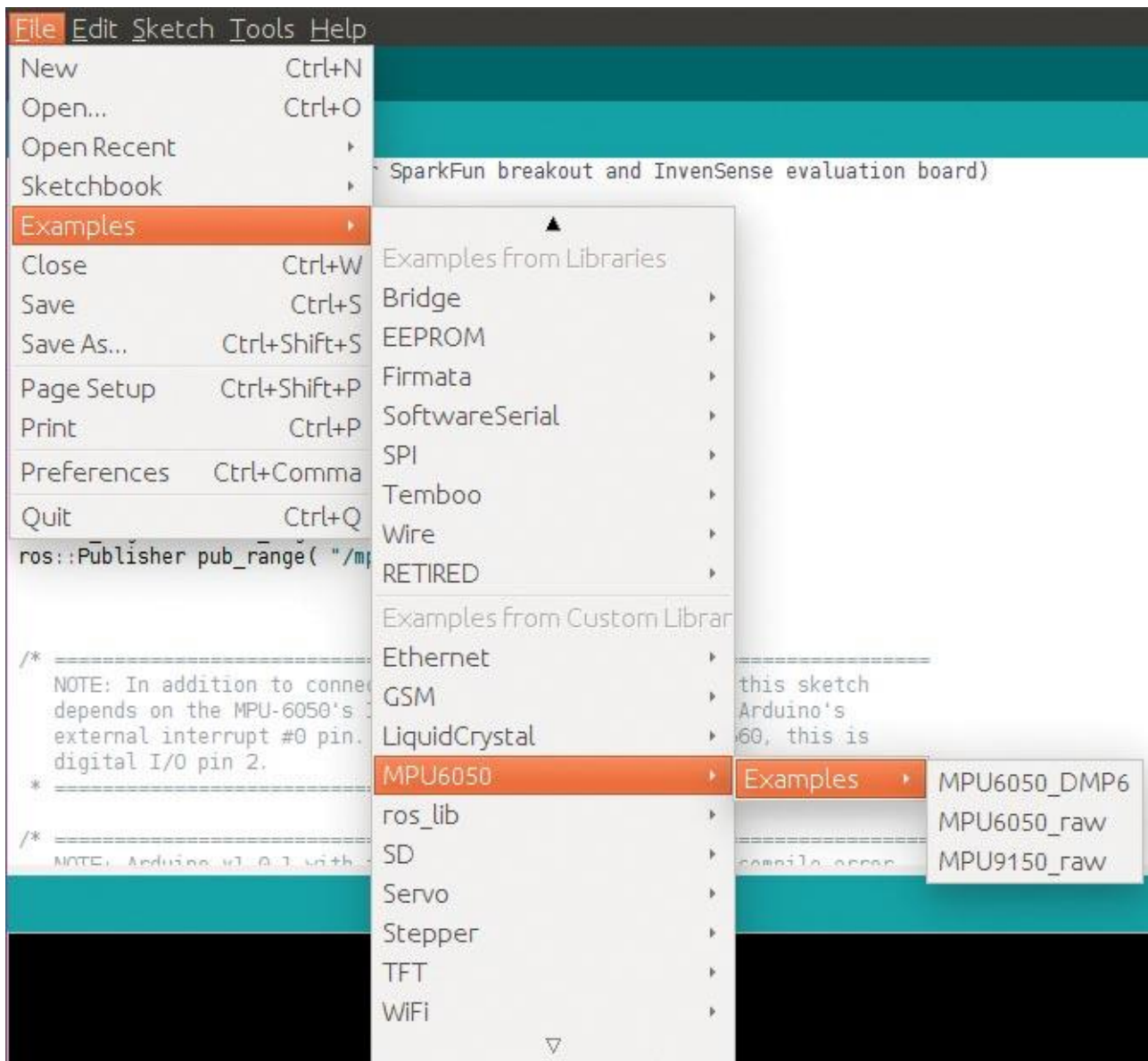


Figura 16. Biblioteca MPU-9150 en Arduino IDE.

Si obtiene esto, puede copiar el código de interfaz *ROS_IMU* al IDE de Arduino; Compila y quema al arduino. La explicación completa del código está comentada en el propio código.

Después de grabar el código *ROS_IMU* en el Arduino, inicie roscore en el lado de la PC. Además, el nodo serial de ROS Python puede darle su propio nombre de dispositivo usando este comando:

```
$ rossrun rosserial_python serial_node.py /dev/ttyACM0
```

Si todo funciona bien, recibirá un mensaje como el que se muestra en la **Figura 17** .

```
lentin@lentin-Aspire-4755:~$ rossrun rosserial_python serial_node.py /dev/ttyACM0
[INFO] [WallTime: 1469787351.360268] ROS Serial Python Node
[INFO] [WallTime: 1469787351.371303] Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [WallTime: 1469787354.107162] Note: publish buffer size is 512 bytes
[INFO] [WallTime: 1469787354.107599] Setup publisher on /tf [tf/tfMessage]
[INFO] [WallTime: 1469787354.111063] Setup publisher on imu_data [sensor_msgs/Imu]
```

Figura 17. Ejecución del nodo rosserial.

Puede enumerar los temas después de ejecutar el comando anterior. Puede obtener nuevos temas como `/imu_data`, `/tf` como en la **Figura 18**.

```
lentin@lentin-Aspire-4755:~$ rostopic list
/diagnostics
/imu_data
/rosout
/rosout_agg
/tf
```

Figura 18. Listado de temas.

También puede inspeccionar cada tema. El `imu_data` se puede repetir con el siguiente comando:

```
$ rostopic echo /imu_data
```

Cuando se hace eco del tema, puede obtener valores como los de la **Figura 19**.

```
header:
  seq: 33
  stamp:
    secs: 0
    nsecs: 0
  frame_id: /base_link
orientation:
  x: 0.197143554688
  y: 0.813171386719
  z: -0.153137207031
  w: 0.525634765625
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: 0.0
  y: 0.0
  z: 0.0
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
linear_acceleration:
  x: 0.0
  y: 0.0
  z: 0.0
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```

Figura 19. Haciendo eco de `imu_data`.

También puedes repetir el `/tf` usando este comando:

```
$ rostopic echo /tf
```

La **Figura 20** muestra una salida típica del tema `tf`.

```
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 1469787491
      nsecs: 693163974
    frame_id: /base_link
  child_frame_id: /map
  transform:
    translation:
      x: 1.0
      y: 0.0
      z: 0.0
    rotation:
      x: 0.19873046875
      y: 0.812744140625
      z: -0.154235839844
      w: 0.525451660156
  ---
```

Figura 20. Haciéndose eco de TF.

Después de verificar estos valores de tema, podemos mostrar los datos de TF dentro de Rviz como puede ver en la **Figura 21** .

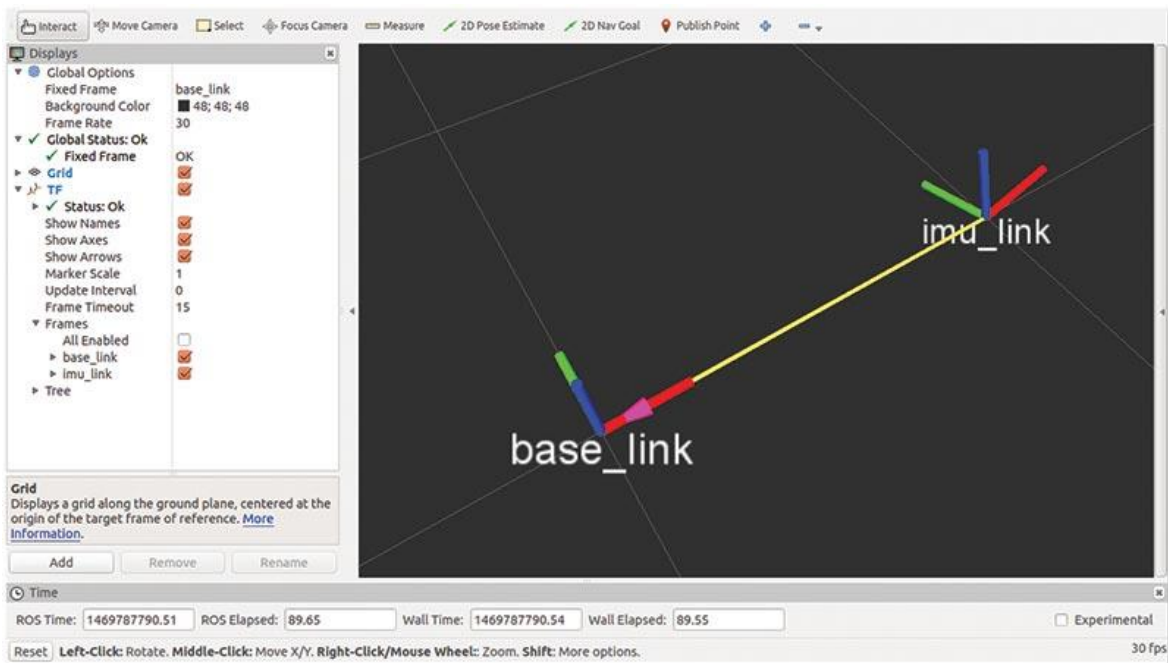


Figura 21. Visualización de TF.

La captura de pantalla muestra el *imu_link* con respecto al *enlace base* que es un enlace estático. El enlace IMU se coloca a una unidad de distancia de *base_link* . Puedes mover la IMU usando tu mano para cambiar su orientación. Puedes ver la misma orientación dentro de Rviz también.

¡Felicidades! ¡Ha terminado de interconectar una de las IMU disponibles en el mercado para ROS! Ahora puede usarlo para varias aplicaciones, como control de robots y teleoperación, quitarte esa bella falda y adaptar la configuración y programación a un verdadero micro.

La tarea para ser entregada el Miercoles 19, es hacer esta actividad usan un micro PSoC4 y el Arduino.