Practica No.5 Robot Operating System - ROS

Instituto Tecnológico Autónomo de México

Primavera 2019

1 Objetivo

- Repasar programación y compilación en C/C++.
- Aprender a utilizar ROS (Robot Operating System).
- Conectar ROS con Arduino.

2 Esbozo de la solución

2.1 ROS

En el repositorio de ésta práctica (https://github.com/garygra/PM_pract_5) se encuentran los archivos iniciales, compilar y probar su funcionamiento.

Implementar la siguiente funcionalidad al programa: el subscriptor le responda al publicador con un mensaje tipo *String* cuando un nuevo mensaje sea recibido. El publicador lee e imprime a la terminal el mensaje que reciba.

2.2 Turtlesim

Implementar un programa que mueva la tortuga utilizando el teclado de la siguiente manera:

- 2: da vuelta en sentido de las manecillas del reloj
- 4: se mueve hacia atrás.
- 6: se mueve hacia adelante.
- 8: da vuelta en sentido contrario a las manecillas del reloj

2.3 Arduino-ROS

Implementar en arduino un programa con la siguiente funcionalidad:

- Escuchar por nuevos mensajes
- Al recibir un mensaje, prender un LED por 3 segundos
- Al pasar los 3 segundos, enviar un mensaje de respuesta.

2.4 ROS en Múltiples Computadoras

Ejecutar Turtlesim en una computadora y controlarla desde otra utilizando el programa ya implementado.

3 Aspectos técnicos

3.1 ROS

Al utilizar ROS es necesario que siempre se encuentre en ejecución `roscore`. Cada programa que utilice ROS puede tener múltiples instancias de subscriptores y publicadores.

Cada publicador publica solamente un tipo de mensaje. En general, se publica un mensaje cuando existe un nuevo evento o a una frecuencia específica.

Cada subscriptor recibe solamente un tipo de mensaje y tiene asociada una función de *callback* que se ejecuta al recibir un mensaje nuevo, dicha función suele contener solamente el código fundamental para guardar los datos recibidos.

En los archivos iniciales se muestra cómo implementar subscriptores y publicadores.

3.2 Turtlesim

Turtlesim es un simulador básico de un robot tipo diferencial. Algunos aspectos importantes para utilizarlo son:

- El tópico */turtleN/cmd_vel* es utilizado para mover al robot. N corresponde al número de instancia del simulador empezando en 1.
- La velocidad a la que se mueve el robot es un valor entre 0 y 1.
- Al recibir el comando, el robot se mueve a la velocidad determinada durante 1 segundo.
- El tópico */turtleN/cmd_vel* es de tipo *geometry_msg/Twist*. Solamente se utilizan dos campos: linear.x para la velocidad en x y angular.z para la velocidad en angular en z.

3.3 Conexión ROS-Arduino

3.3.1 Instalación

Descargar la librería de *roserial* para intercomunicarse con el arduino.

```
cd <ws>/src
git clone https://github.com/ros-drivers/rosserial.git
cd <ws>
catkin_make
source devel/setup.bash
```

Instalar la librería de ROS para Arduino.

Para utilizar ROS desde Arduino, se necesita importar `ros.h` así como cualquier otra librería que utilice el programa, como los mensajes.

```
cd ~/Arduino/libraries //might be other path...
rm -rf ros_lib //only if ros_lib exists
rosrun rosserial_arduino make_libraries.py .
```

3.3.2 Publisher

Para utilizar ROS desde Arduino, se necesita importar `ros.h` así como cualquier otra librería que utilice el programa, como los mensajes.

```
#include <ros.h>
```

Se necesitan declarar variables globales para el *node handle*, los mensajes utilizados y el publicador.

```
ros::NodeHandle nh;
std_msgs::String str_msg;
ros::Publisher pub("arduino_msg", &str_msg);
```

En la función *setup* es necesario inicializar el nodo y el publicador:

```
  nh.initNode();
  nh.advertise(pub);
```

Para publicar un mensaje, dentro de la función *loop* se utilizan las siguientes funciones:

```
  str_msg.data = string_variable;
  chatter.publish( &str_msg );
  nh.spinOnce();
```

3.3.3 Subscriber

Para implementar un suscriptor, se tiene que declarar la función *callback*:

```
void message_ros( const std_msgs::String& ros_msg)
{
        string_variable = ros_msg.data;
}
```

Es necesario declarar el suscriptor:

```
ros::Subscriber<std_msgs::String> sub("other_msg", &message_ros );
```

En la función *setup* se inicializa:

```
  nh.subscribe(sub);
```

3.3.4 Ejecución

Despues de cargar el programa al Arduino, se debe realizar lo siguiente:

- Obtener el puerto donde esta conectado el Arduino, similar a */dev/ttyACM0*

- Ejecutar roscore
- Ejecutar:

```
rosrun rosserial_python serial_node.py PUERTO_ARDUINO
```

- Ejecutar el resto de los nodos, por ejemplo:

```
rostopic echo /arduino_topic_name
```

3.4 ROS en Múltiples Computadoras

Para poder correr en una computadora Turtlesim y controlar la tortuga desde otra, es necesario *decirle* a ROS en cuales computadoras se encuentran los procesos mediante las direcciones IPv4 de cada una.

- Con el comando *ifconfig* se puede obtener la dirección IP de cada computadora.
- En cada terminal que se utilice, utilizar el comando *export* para asignar valor a la variable de ambiente *ROS_HOSTNAME* de la siguiente manera:

  ```
  export ROS_HOSTNAME=ip_computadora
  ```

  donde *ip_computadora* es la ip de la propia computadora.
- Decidir cuál computadora se va a utilizar como *master*, donde se corre *roscore*.
- Realizar en cada terminal utilizada:

  ```
  export ROS_MASTER_URI=http://ip_master:11311/
  ```

  donde *ip_master* es la ip de la computadora *master* y *11311* es el puerto que ROS utiliza para comunicarse.

Referencias Recomendadas

Linux man pages https://linux.die.net/man/

C++ Resources Network http://www.cplusplus.com/

Robot Operating System http://www.ros.org/

ROS Cheat Sheet https://github.com/garygra/ROS_cheat_sheet/blob/master/main.pdf