Instituto Tecnológico y de Estudios Superiores de Monterrey

TE3001B.101

Fundamentación de robótica (Gpo 101)

Semestre: Febrero - Junio 2022

Reto semanal 1 Manchester Robotics

Alumnos:

Frida Lizett Zavala Pérez	A01275226
José Jezarel Sánchez Mijares	A01735226
Antonio Silva Martínez	A01173663

Fecha de entrega: 20 de Febrero del 2023

Reto semanal 1. Manchester Robotics

Resumen

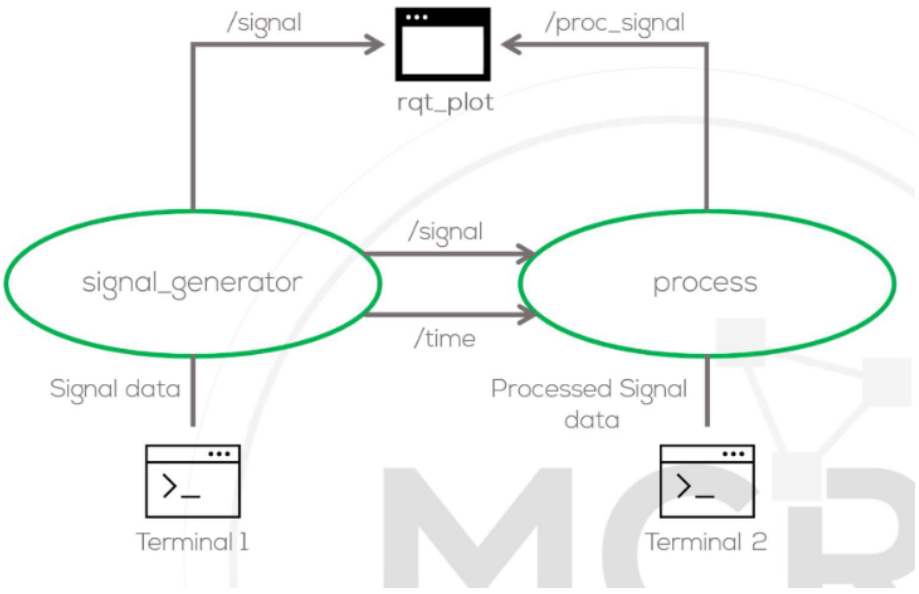
Se desarrolla el primer reto del curso de ROS,curso que consiste en introducir los conceptos básicos y conocimiento general del ambiente de ROS.

Descripción general del entorno de ROS, tales como los nodos, topics y mensajes, los cuales son la base de la forma en la que se lleva a cabo el proceso de comunicación en ROS.

Objetivos

El objetivo de este reto además de funcionar como una introducción a ROS y conocer su manera de comunicación, es generar un nodo que mande una señal a otro nodo para procesarla.

El primer nodo actuará como un generador de señales. Generará una señal sinusoidal, mientras que el segundo actuará como un procesador, que tomará la señal generada por el nodo anterior y la modificará para generar una señal procesada. Ambas señales se mostrarán usando rqt_plot. Dos terminales diferentes mostrarán la información de cada una de las señales. Por último con un archivo launch se deben ejecutar los nodos, las terminales y rqt_plot al mismo tiempo.

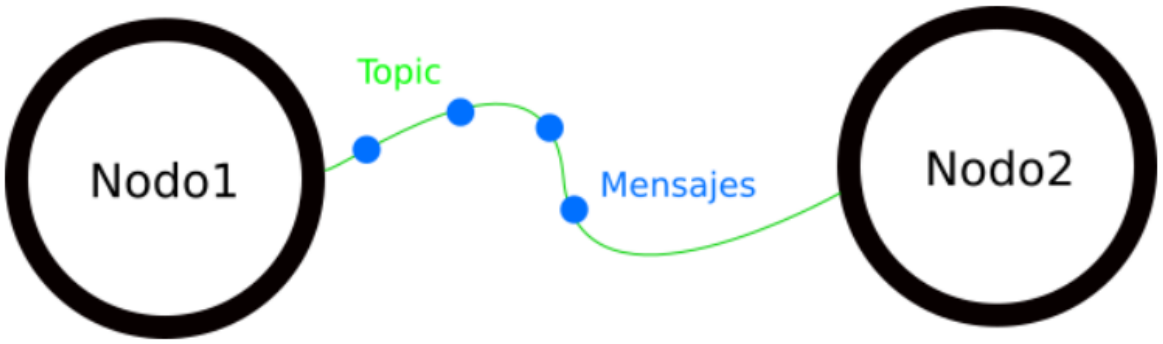


Introducción

ROS (Robot Operating System) es un marco de software gratuito y de código abierto diseñado para el desarrollo de robots. ROS proporciona muchas herramientas, bibliotecas y convenciones para ayudar a los desarrolladores a crear aplicaciones automatizadas. Está diseñado para ser modular y distribuido, lo que significa que puede ejecutarse en diferentes sistemas operativos y diferentes arquitecturas de hardware. Las aplicaciones ROS se dividen en nodos, que son unidades de software que se comunican entre sí a través de mensajes. Esto permite que los nodos se especialicen en tareas específicas y se comuniquen con otros nodos para realizar tareas más complejas. ROS también proporciona herramientas de simulación y visualización, lo que permite a los desarrolladores probar sus aplicaciones en un entorno simulado antes de implementarlas en un robot real. En resumen, ROS es un conjunto de bibliotecas y herramientas de software que ayudan a los desarrolladores a construir aplicaciones robóticas de forma modular y distribuida.

ROS funciona como un sistema de mensajería el cual nos permite la comunicación entre nodos de software. Cada uno de estos nodos realiza una tarea en específico y se comunican con otros nodos enviando y recibiendo mensajes. Para que estos puedan comunicarse estos deben de estar conectados por medio de un grafo el cual define la estructura de los nodos y las conexiones entre ellos.

Los principales agentes de comunicación de ROS, los cuales fueron usados en este reto son, como ya se ha mencionado, los nodos, los cuales se comunican entre sí a partir de los topics, que es por donde pasan los mensajes.



Solución del problema

Para la resolución de este mini reto primeramente se realizó la instalación de Ubuntu 18.04 ya que esta versión es la que es compatible con ROS Melodic, que es el que será usado para este curso. Por lo que lo siguiente fue hacer su instalación. Una vez instalado lo requerido y configurado el ambiente de trabajo comenzamos.

Como primer paso se creó un espacio de trabajo donde creamos un paquete nuevo con el nombre “courseworks”, y creamos un nodo llamado “signal_generator”, el cual será el que generará una onda senoidal con respecto al tiempo $y = f(t) = sin(t)$. El script debe realizar lo siguiente:

Manda el resultado usando un mensaje Float32 de ROS, a través de un topic con el nombre “/signal”, del mismo modo manda el tiempo a otro topic llamado “/time”. Se especifica que se usará una frecuencia de 10 Hz. Por último imprime el resultado en la terminal usando rospy.loginfo. Todo esto se logró en el primer código que aunque presentaron problemas al imprimir el tiempo se lograron solucionar colocando un contador más sencillo

Enseguida se crea el nodo que procesa la información, éste con el nombre de “process”, el cual estará conectado a los topics “/signal” y “/time”. Este nodo procesa la información obtenida por el primer programa, es decir, la señal, así con fórmulas trigonométricas pudimos desfasar la señal nueva de la original.

Por último para finalizar el challenge tuvimos que realizar el código para el launch de los nodos, ahí se especificó que se abrieran dos terminales nuevas al momento de ejecutarse el programa y también que automáticamente se abrieran tanto la gráfica como el esquema resultante de los resultados dados.

```python
#!/usr/bin/env python
import rospy
import numpy as np
from std_msgs.msg import Float32

o_signal = 0
o_time = 0

def callbacksignal(msgsignal):
   global o_signal
   o_signal = msgsignal.data

def callbacktime(msgtime):
   global o_time
   o_time = msgtime.data

if __name__=='__main__':
   rospy.init_node('process')
   rospy.Subscriber("signal", Float32, callbacksignal)
   rospy.Subscriber("time", Float32, callbacktime)

   pub_signal2 = rospy.Publisher("process_2", Float32, queue_size =
10)
   pub_time = rospy.Publisher("time_2", Float32, queue_size = 10)
   rate = rospy.Rate(10) #10 Hz

   while not rospy.is_shutdown():
       t=o_time

       des_sign=((o_signal * np.cos(5)) + np.cos(t) *
np.sin(5))*-np.cos(t)
       resultado = "%3f | %3f" %(des_sign,t)
       rospy.loginfo(resultado)
       pub_signal2.publish(des_sign)
       pub_time.publish(t)

       rate.sleep()
```

Signal_generator.py

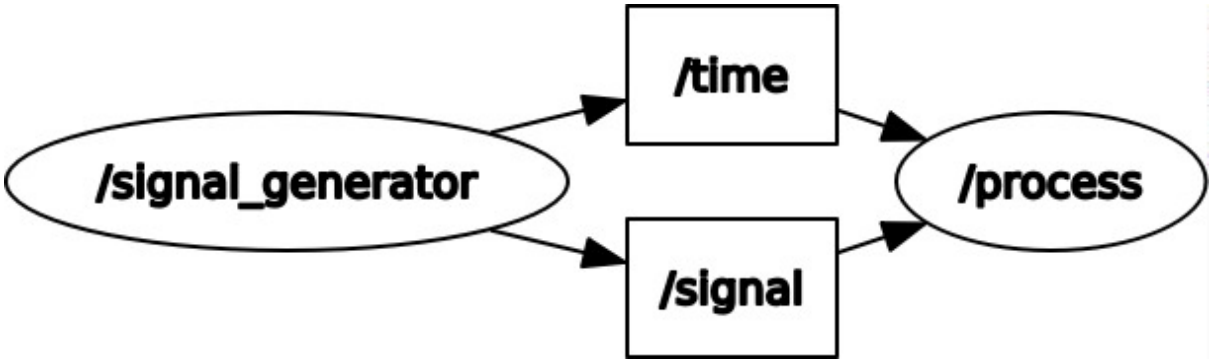
```python
#!/usr/bin/env python
import numpy as np
import rospy
from std_msgs.msg import Float32
if __name__=='__main__':
   rospy.init_node("signal_generator")
   pub_signal=rospy.Publisher("signal", Float32, queue_size=10)
   pub_time=rospy.Publisher("time", Float32, queue_size=10)
   rate=rospy.Rate(10)
   time= 0
   while not rospy.is_shutdown():
       time+=0.1
       signal= np.sin(time)
       result = "%3f | %3f" %(signal,time)
       rospy.loginfo(result)
       pub_signal.publish(signal)
       pub_time.publish(time)
       rate.sleep()
```

```
#launch file

<?xml version="1.0" ?>

<launch>



                    <node   name="signal_generator"   pkg="courseworks"
type="signal_generator.py"                                  output="screen"
launch-prefix="gnome-terminal --command" />

             <node  name="process"  pkg="courseworks"  type="process.py"
output="screen" launch-prefix="gnome-terminal --command" />

                <node  name="rqt_plot"  pkg="rqt_plot"  type="rqt_plot"
output="screen" args="/Signal /proc_signal"/>



             <node  name="rqt_graph"  pkg="rqt_graph"  type="rqt_graph"
output="screen"/>
```

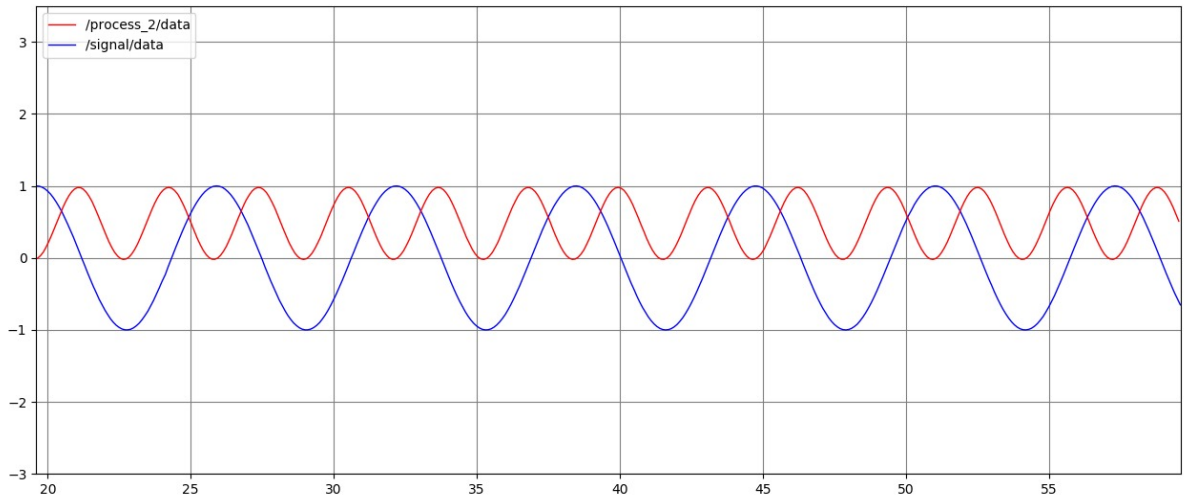
```
</launch>
```

Resultados

Diagrama de los nodos y topics



Señales de entrada y salida



Los resultados obtenidos fueron gracias a los programas en python ya que estos fueron los que nos proyectaron el comportamiento de las ondas senoidales con respecto al tiempo, a su vez la organización de las carpetas dentro de ROS jugó un

papel muy importante para la comunicación ya que la forma en cómo se crearon y distribuyeron las carpetas nos dejaron ver el funcionamiento de dichas ondas.

Conclusiones

Para esta primera semana nos enfrentamos a un reto de introducción de ROS en donde tuvimos que realizar el procesamiento de una señal senoidal, este proyecto nos sirvió de introducción ya que vimos el funcionamiento general de este entorno de programación dándonos los conceptos básicos y el cómo tenemos que inicializar la comunicación entre nodos. Por otra parte, tuvimos que pensar en cómo podríamos proyectar lo solicitado desde los scripts de python por lo que tuvimos que desempolvar los conocimientos previos sobre programación en python y ponerlos en práctica para poder proyectar dichas señales.

Este mini reto a pesar de ser bastante básico, no fue tan sencillo de primera instancia ya que el proceso de instalación previo, ocasionó pequeños inconvenientes, además como con cualquier primer acercamiento a una herramienta, el reconocimiento de la manera en la que se trabaja con ROS y su proceso de comunicación fue algo muy nuevo para nosotros, sin embargo una vez que nos familiarizamos con este entorno de trabajo y entendimos los principios básicos, pudimos proseguir con más seguridad en la realización del reto.

Referencias

4 Nodos, Topics y Mensajes. Turtlesim – ROS TUTORIAL. Tutorial ROS en español. (s. f.). http://rostutorial.com/4-nodos-topics-y-mensajes-turtlesim/

M. (s. f.). *GitHub - ManchesterRoboticsLtd/TE3001B_Robotics_Foundation: TE3001B Repository*. GitHub. https://github.com/ManchesterRoboticsLtd/TE3001B_Robotics_Foundation

Qué es ROS (Robot Operating System). (2020, 25 marzo). OpenWebinars.net. https://openwebinars.net/blog/que-es-ros/