Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Fundamentación de Robótica

TE3001B gpo 101

Semestre ago-dic 2022

Challenge 02

Integrantes:

Jonathan Josafat Vázquez Suárez	A01734225
Jhonatan Yael Martinez Vargas	A01734193
Diego García Rueda	A01735217

Profesor

Rigoberto Cerino Jiménez

Fecha de Entrega: 28 de Febrero del 2023

Objetivos
Aprende a implementar mensajes customizados y parámetros en ROS para generar un sistema de control para un sistema de primer orden.

Introducción
En esta actividad, el objetivo es diseñar un controlador para un sistema de primer orden en ROS, el cual simula el comportamiento dinámico de un motor de corriente continua. Para llevar a cabo la creación del controlador, es necesario crear tres nodos principales: **"set_point_generator", "controller" y "motor_system".**

El nodo **"set_point_generator"** se encarga de generar una señal triangular que va desde -100 a 100, en intervalos de más menos uno y que se publica a través del tópico ***"set_point"*.**

El nodo **"controller"** es el encargado de recibir los datos de la señal generada por el nodo **"set_point_generator"** y compararla con la velocidad sensada que da el motor el cual se simula en el nodo **"system"**, para así generar la señal de control adecuada que se enviará de vuelta a este nodo.

Por último, el nodo **"system"** es el encargado de recibir la señal de control generada por el nodo **"controller"** y aplicarla al motor para que éste se mueva a la velocidad deseada.

Todo esto se puede crear en ROS mediante un archivo launch, en el cual se pueden utilizar parámetros mediante rosparam para configurar la velocidad deseada, las ganancias del controlador y otros parámetros relevantes.

Algunos puntos importantes tratados durante el desarrollo de esta actividad son namespaces, parameters y custom messages, estos son funcionalidades clave en ROS, las cuales permiten la configuración y personalización de nodos y tópicos.

Los Namespace son maneras lógicas de organización para nodos y tópicos, esto permite tener múltiples instancias del mismo nodo o tópico dentro de un controlador, permitiendo agruparlos en diferentes namespaces.

Los Custom Message permiten crear diccionarios de información en un mensaje, de esta forma podemos personalizar la manera de comunicación entre nodos, este método permite almacenar cualquier tipo de dato, así como matrices y estructuras. La declaración de estos objetos es mediante un archivo .msg que es declarado dentro de CMakeLists.txt.

Los parámetros son variables que se pueden utilizar para configurar un nodo o un tópico. Los parámetros se pueden definir y ajustar en tiempo de ejecución, lo que permite una mayor flexibilidad en la configuración del sistema. Los parámetros pueden ser de varios tipos, como enteros, flotantes o cadenas de caracteres.

Solución al problema

El abordaje en esta situación es mediante los objetivos y diagramas presentados, como todo proyecto en ROS, debe ser montada toda la estructura y definirse los principales nodos así como el archivo launch y sus carpetas correspondientes. En este caso recibimos el diagrama completo de resultados esperados junto a el script completo de “/system”.

Código del nodo controller:

```
#!/usr/bin/env python
import rospy
import numpy as np
from control.msg import motor_input, motor_output, set_point
from std_msgs.msg import Float64

controller_current_speed = 0
target_speed = 0
i = 0.0

def pid_controller(_system_error, _prev_error,_Kp, _Kd, _Ki):

   global i
   dt = 0.01

   # Sistema proporcional
   p = _Kp * _system_error

   # Sistema integral
   i = _Ki * (i + _system_error * dt)

   # Sistema derivativo
   d = _Kd * (_system_error - _prev_error) / dt

   # Salidad del sistema PID
   pid_response = p + i + d

   return pid_response

# Funcion que simula el estado nulo del motor
def stop():

   global controller_pub

   # Setup the stop message (can be the same as the control message)
   print("Stopping")

   # Creacion y seteo de valores para una variable de tipo "motor_input"
   end_point = motor_input()

   end_point.input = 0.0
```

```
    end_point.time = rospy.get_time() - start_time

    # Se publica el estado nulo del motor
    controller_pub.publish(end_point)

    rospy.loginfo("Total Simulation Time = %f" % end_point.time)

def system_response_callback(msg):

    global controller_current_speed

    system_response = msg

    controller_current_speed = system_response.output
    system_response_time = system_response.time
    system_response_status = system_response.status

    rospy.loginfo("Motor Time: %f", system_response_time)
    rospy.loginfo("Motor Speed: %f", controller_current_speed)
    rospy.loginfo("Motor Status %s", system_response_status)
    print("\n")

def system_target_callback(msg):

    global target_speed

    system_target = msg
    target_speed = system_target.new_target
    target_time = system_target.time

    rospy.loginfo("Time: %f", target_time)
    rospy.loginfo("New   Recived: %f", target_speed)
    print("/n")

if __name__ == '__main__':

    try:

        rospy.init_node("controller", anonymous = True)

        print("The Controller is Running")

       controller_pub = rospy.Publisher("/motor_input", motor_input, queue_size = 1)
        error_pub = rospy.Publisher("/sys_error", Float64, queue_size = 1)

        rate = rospy.Rate(100)

        rospy.on_shutdown(stop)

        controller_Kp = rospy.get_param("/controller_kp")
        controller_Ki = rospy.get_param("/controller_ki")
```

```
      controller_Kd = rospy.get_param("/controller_kd")

      start_time = rospy.get_time()

      rospy.Subscriber("/set_point", set_point, system_target_callback)
      rospy.Subscriber("/motor_output", motor_output, system_response_callback)

      prev_error = 0.0

      while not rospy.is_shutdown():

          # Error del sistema
          system_error = (target_speed - controller_current_speed) * 0.100

          print("System Error: ", system_error)
          print("Target Speed: ", target_speed)
          print("Current_Speed: ", controller_current_speed)
          print("\n")

          # Respuesta del controlador PID
             pid_output = pid_controller(system_error, prev_error, controller_Kp,
controller_Ki, controller_Kd)

          # Timestamp de cuando entra el
          pid_time = rospy.get_time() - start_time

          # Creacion y seteo de valores a una variable de tipo "motor_input"
          new_target = motor_input()

          new_target.time = pid_time
          new_target.input = pid_output

          print("Nueva señal de salida: ", new_target.input)

          # Se almacena el error previo
          prev_error = prev_error + system_error

          # Se publica el nuevo target del controlador
          error_pub.publish(system_error)
          controller_pub.publish(new_target)

          rate.sleep()

  except rospy.ROSInitException:
      pass
```

Código del nodo set_point_generator:

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

# Funcion que genera puntos entre -100 y 100 con intervalos de 10
def new_data(_max_power, _min_power, _current_power):

   global flag

   if flag:

       if _current_power < _max_power:

           _current_power = _current_power + 1

       else:

           flag = False

   else:

       if _current_power > _min_power:

           _current_power = _current_power - 1

       else:

           flag = True

   return _current_power

if __name__ == '__main__':

   try:

       rospy.init_node("set_point_generator")

       setpoint_pub = rospy.Publisher("/set_point", set_point, queue_size = 1)


      max_power = rospy.get_param("/setpoint_singal_amplitud", 100)
       min_power = rospy.get_param("/setpoint_signal_frecuency", -100)
       current_power = rospy.get_param("/setpoint_current_power", 0)

       start_time = rospy.get_time()
```

```
    rate = rospy.Rate(100)


    new_point = set_point()

    flag = True

    while not rospy.is_shutdown():

        current_power = new_data(max_power, min_power, current_power)

        new_point.time = rospy.get_time() - start_time
        new_point.new_target = current_power

        rospy.loginfo("Time: %f" % new_point.time)
        rospy.loginfo("New Target: %f" % new_point.new_target)
        print("\n")

        setpoint_pub.publish(new_point)

        rate.sleep()

except rospy.ROSInitException:

    pass
```

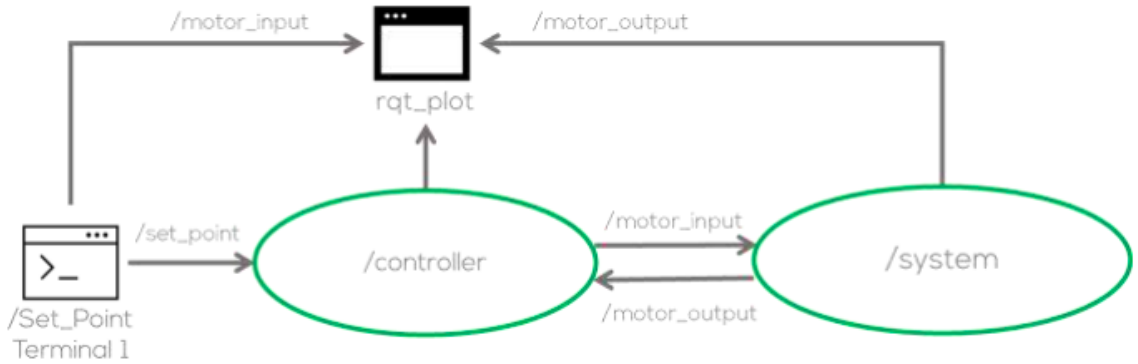

Figura 01. Resultados esperados

Para el caso de **"/set_point"**, podemos definir completamente su comportamiento, este nodo tiene el objetivo de enviar mediante su mismo nombre un mensaje personalizado con al menos dos parámetros a el nodo **"controller"**. Nodo encargado de procesar mediante un sistema PID la salida del motor **"/motor_output"** y **"/set_point"** para

A continuación se presenta el código de los nodos utilizados en este reto:

Resultados

De acuerdo a lo realizado, se obtuvo la siguiente gráfica utilizando la herramienta *rqt plot.*

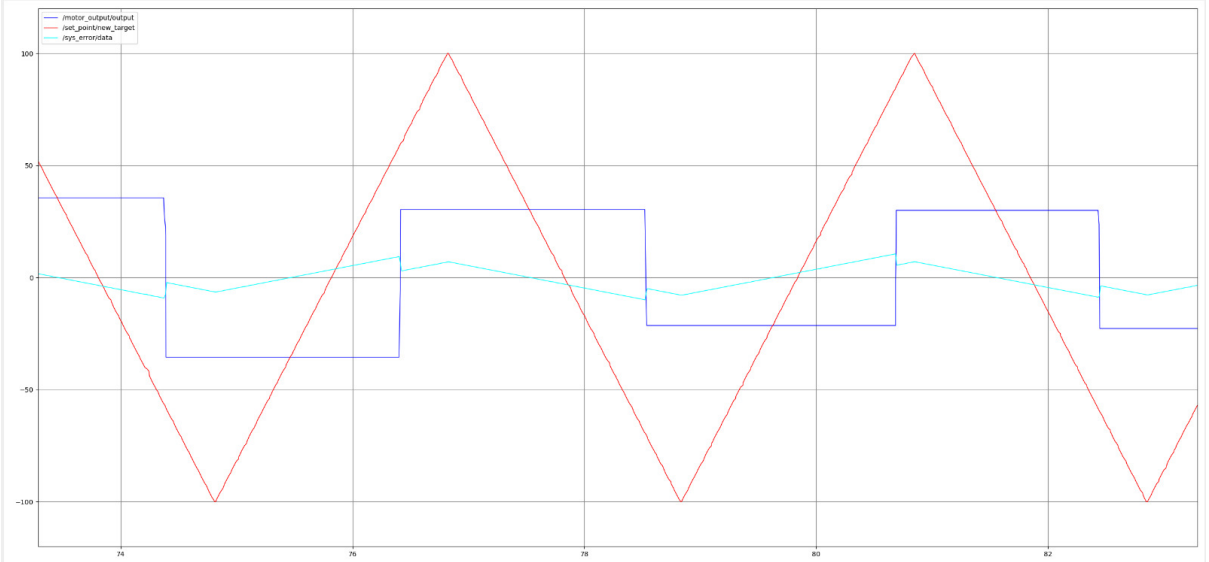

Figura 02. Análisis de señales de entrada, salida y error

En esta graficación se puede observar la señal de entrada (línea roja), la azul fuerte es la señal de salida del motor (línea azul marino) y el error (linea celeste), cabe destacar que el comportamiento que presenta la salida del motor es similar a una señal cuadrada y esto se debe a que el motor está entrando en su fase de saturación tanto para cuando este gira en sentido horario como en anti horario.

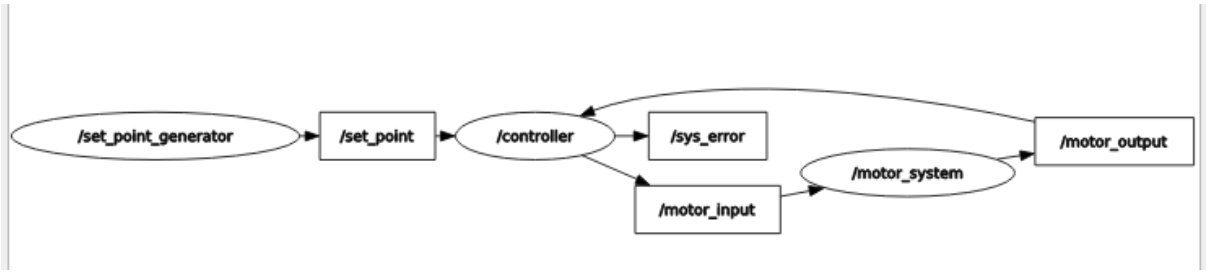

Figura 03. Gráfico de nodos y tópicos resultantes

Gracias a la figura 03, se puede comprobar que los resultados obtenidos son los esperados pues el flujo de conexiones que existe entre los distintos nodos es el esperado, cabe mencionar que para poder graficar la señal del error proveniente de nodo **“controller”** se tuvo que crear un tópico extra **“sys_error”**

Conclusiones

Los resultados de este reto se cumplieron satisfactoriamente, existieron algunos inconvenientes al momento de la creación y ajuste de las constantes del controlador PID. Consideramos que este challenge fue retador a comparación del challenge pasado ya que la diferencia de dificultad se hizo notar, respecto al manejo de ROS hubo mejoras respecto

al challenge anterior ya que la creación de los nodos, tópicos y sus respectivas publicaciones y suscripciones fue más rápida y sencilla de hacer consiguiendo así una manipulación de estos aspectos mucho más fácil