Tutorial Aplicación ROS

Control de turtlesim con Joystick Arduino

Autor: René Torres
Fecha: 30 de octubre de 2023

Índice

1. Introducción

En este documento se detalla el proceso de instalación y configuración de un proyecto de ROS, el cual consiste en el control de la tortuga de turtlesim mediante un joystick conectado a un arduino. El proyecto se encuentra disponible en el repositorio de github: https://github.com/ReneTorresA/jstk_turtlesim.

La aplicación esta configurada para que al presionar el botón del joystick se mueva la tortuga de forma aleatoria en el espacio de trabajo de turtlesim, como se muestra en la figura 1, y además se cambie el color y espesor del trayecto de la tortuga de forma aleatoria.

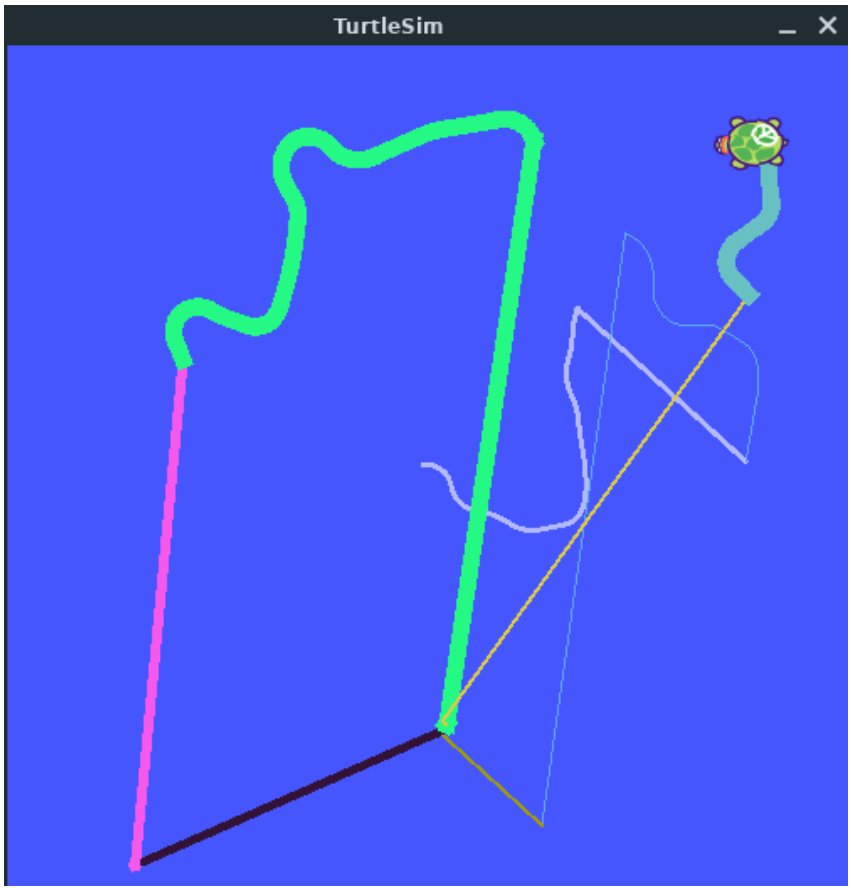

Figura 1: Espacio de trabajo de turtlesim.

2. Instalación de Dependencias

Para que este proyecto funcione adecuadamente es necesario instalar las siguientes dependencias y software:

1. ROS Noetic: http://wiki.ros.org/noetic
2. Libreria Rosserial Arduino: http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup
3. Arduino IDE: https://www.arduino.cc/en/software
4. Python 3: https://www.python.org/

El sistema operativo usado en este proyecto es una distribución de Linux basada en **Ubuntu 20.04**.

2.1. Diagrama de conexión y configuración de arduino

Para este proyecto es necesario contar con una placa de arduino y un modulo de joystick, a continuación en la figura 2 se muestra el diagrama de conexión de los componentes.

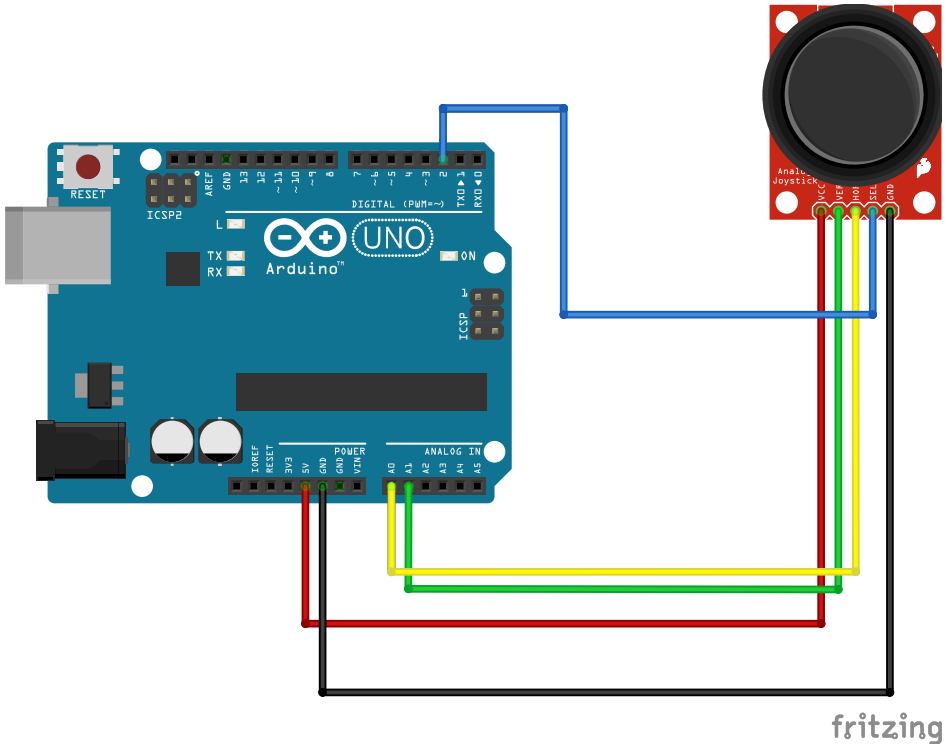

Figura 2: Diagrama de conexión.

Una vez realizada las conexiones se procede a modificar (si es necesario) y cargar el código de arduino en la placa, el cual se encuentra en el repositorio de github y adjunto en el apéndice, sección 4.2. Se recomienda que verifique que su placa de arduino este conectada correctamente y que guarde el nombre del puerto serial, ya que lo necesitara para modificar el archivo .
Para saber que puerto está usando su placa de arduino puede visualizarlo en la aplicación de Arduino IDE, como se muestra en la figura 3.

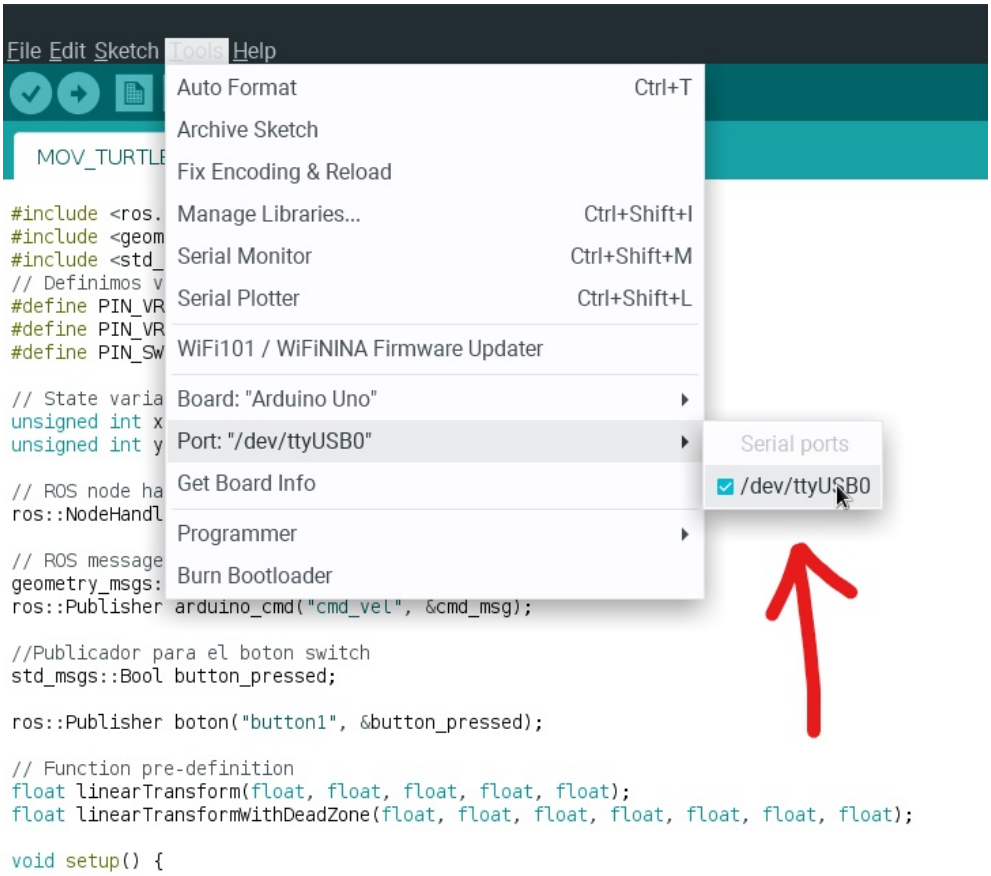

Figura 3: Puerto serial de arduino.

En este caso el puerto serial es **/dev/ttyUSB0**, por lo que hay que asegurar que en el archivo **jstk__turtlesim.launch** que se encuentra en la carpeta **launch** de la aplicación, en la etiqueta con nombre "port.[en] el apartado default tiene que estar el nombre correcto del puerto, como se muestra a continuación:

jstk_turtlesim.launch

```
<launch>
  <!-- Start turtlesim_node node -->
  <node name="turtlesim" pkg="turtlesim" type="turtlesim_node"/>

  <!-- Start rqt_graph for node visualization -->
  <node name="rqt_graph" pkg="rqt_graph" type="rqt_graph"/>

  <!-- Start button_suscriber node for call service -->
  <node name="listener_button" pkg="jstk_turtlesim" type="button_suscriber.py"/>

  <!-- Rosserial arduino node -->
  <arg name="port" default="/dev/ttyUSB0"/>

  <node name="arduino_node" pkg="rosserial_python" type="serial_node.py">
    <param name="port" type="string" value="$(arg port)"/>
    <remap from="cmd_vel" to="turtle1/cmd_vel"/>
```

```
  </node>
</launch>
```

3. Uso de la aplicación

Para usar la aplicación se recomienda seguir los siguientes pasos:

Paso 1: Clone el repositorio de github en la carpeta **src** de su espacio de trabajo de ROS. Para esto abra una terminal y ejecute el siguiente comando:

Paso 1

```
1 cd ~/catkin_ws/src
2 git clone https://github.com/ReneTorresA/jstk_turtlesim.git
```

3.1. Estructura del proyecto

La estructura, luego de ser clonado el respositorio, deberia quedar de la siguiente forma:

```
catkin_ws
|-- CMakeLists.txt
|-- package.xml
|-- build
|-- devel
|-- src
    |-- jstk_turtlesim
        |-- src
            |-- scripts
                |-- button_subscriber.py
            |-- launch
                |-- jstk_turtlesim.launch
            |-- arduino
                |-- jstk_turtlesim.ino
            |-- docs
                |-- presentacion
                    |-- presentacion.pdf
                    |-- presentacion_latex
                |-- tutorial
                    |-- tutorial.pdf
                    |-- tutorial_latex
            |-- CMakeLists.txt
            |-- package.xml
```

Una vez clonado el repositorio se recomienda agregar el *path* a la terminal. Para esto ejecute el siguiente comando en la carpta del espacio de trabajo:

Paso 2

```
1 cd ~/catkin_ws
2 source devel/setup.bash
```

Paso 3: Para ejecutar la aplicación ejecute el siguiente comando en la terminal:

Paso 3

```
1 roslaunch jstk_turtlesim jstk_turtlesim.launch
```

Si todo ha salido bien debería ver la aplicación de turtlesim y el nodo de rqt_graph, como se muestra en las figuras 5 y 4, respectivamente. Si presiona el botón del joystick debería ver que la tortuga se mueve de forma aleatoria en el espacio de trabajo de turtlesim y que el color y espesor del trayecto de la tortuga cambia de forma aleatoria, con el joystick puede controlar el movimiento de la tortuga lineal y angularmente.

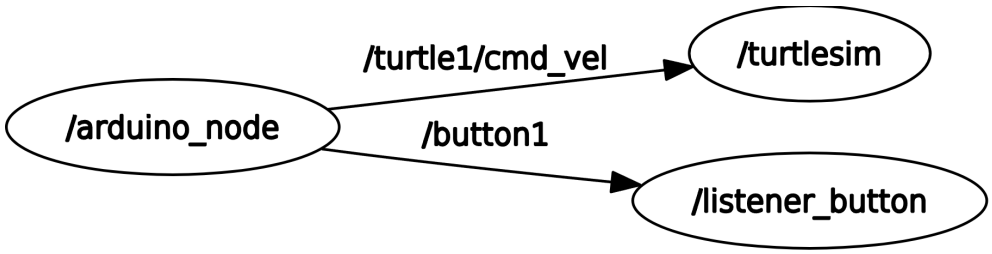

Figura 4: Comunicación entre nodos.

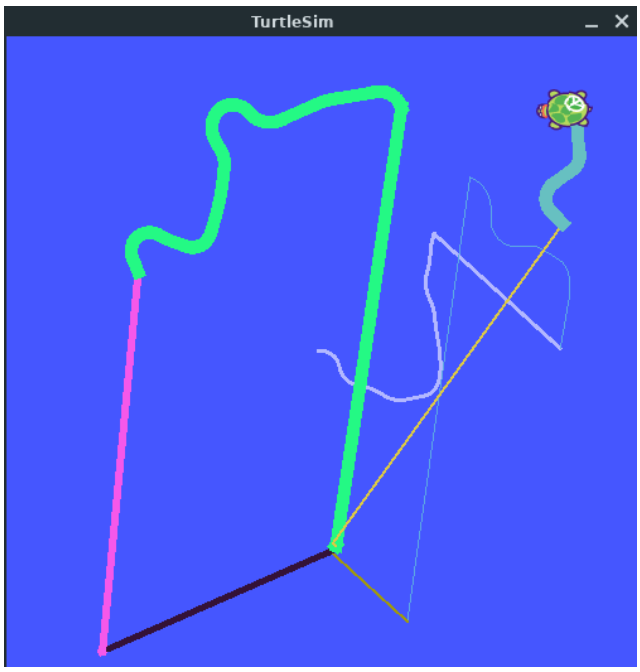

Figura 5: Espacio de trabajo de turtlesim.

Referencias

1. PASCAL., José. Apuntes de clases: Robotic Operating System. *Universidad de Santiago de Chile, Departamento de Mecánica*. [s.f.].
2. AL., Stanford Artificial Intelligence Laboratory et. *Robotic Operating System*. 2023. ROS Noetic. Disponible también desde: https://www.ros.org.

4. Apéndice

4.1. Nodo button_suscriber.py

button_suscriber.py

```
1  #!/usr/bin/env python3
2
3  import rospy
4  from std_msgs.msg import Bool
5  import random
6  from turtlesim.srv import TeleportAbsolute, TeleportAbsoluteRequest, SetPen,SetPenRequest
7  # Function when a message is received
8  def callback(data):
9    rospy.wait_for_service('/turtle1/teleport_absolute')
10   spawn_client = rospy.ServiceProxy('/turtle1/teleport_absolute',TeleportAbsolute)
11
12   request = TeleportAbsoluteRequest()
13
14   request.x = random.uniform(0,11)
15   request.y = random.uniform(0,11)
16   request.theta = random.uniform(0,3.1416)
17
18   rospy.wait_for_service('/turtle1/set_pen')
19   set_pen_client = rospy.ServiceProxy('/turtle1/set_pen',SetPen)
20   requestdos=SetPenRequest()
21   requestdos.r = random.randint(0,255)
22   requestdos.g = random.randint(0,255)
23   requestdos.b = random.randint(0,255)
24   requestdos.width = random.randint(0,10)
25
26 if data.data == True:
27   spawn_client(request)
28   set_pen_client(requestdos)
29   rospy.loginfo("La tortuga ha sido movida aleatoriamente")
30 else:
31   rospy.loginfo(f"Esperando por el servicio")
32
33 rospy.init_node('listener_button') # Create node
34 rospy.Subscriber('/button1', Bool, callback) # Create subscriber
35 rospy.spin() # Wait for a message
```

4.2. Código Arduino (jstk_turtlesim.ino)

jstk_turtlesim.ino

```
// Incluimos las librerias necesarias de ROS
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/Bool.h>
// Definimos variables para los puertos del Arduino
#define PIN_VRx A0
#define PIN_VRy A1
#define PIN_SW 2

// State variables
unsigned int xJoystick = 0;
unsigned int yJoystick = 0;
// ROS node handle
ros::NodeHandle nh;
// ROS message and publisher
geometry_msgs::Twist cmd_msg;
ros::Publisher arduino_cmd("cmd_vel", &cmd_msg);
//Publicador para el boton switch
std_msgs::Bool button_pressed;
ros::Publisher boton("button1", &button_pressed);
// Function pre-definition
float linearTransform(float, float, float, float, float);
float linearTransformWithDeadZone(float, float, float, float, float, float, float);
void setup() {
pinMode(PIN_SW,INPUT_PULLUP);
nh.initNode();
nh.advertise(arduino_cmd);
nh.advertise(boton);
while(!nh.connected()) {
  nh.spinOnce();
}
nh.loginfo("Startup␣complete!");
}
```

jstk_turtlesim.ino

```
44 void loop() {
45
46 // Read data from joystick
47 xJoystick = analogRead(PIN_VRx);
48 yJoystick = analogRead(PIN_VRy);
49
50 // Scale x value
51 float cmd_vel_x = linearTransformWithDeadZone(yJoystick, 0, -1, 500, 520, 1023, 1);
52
53
54 // Scale y value
55 float cmd_vel_rot = linearTransformWithDeadZone(xJoystick, 0, 3, 500, 520, 1023, -3);
56
57 // Send cmd_vel msg
58 cmd_msg.linear.x = cmd_vel_x;
59 // Send cmd_rot msg
60 cmd_msg.angular.z = cmd_vel_rot;
61
62 arduino_cmd.publish(&cmd_msg);
63 nh.spinOnce();
64 // Publicamos el estado del boton
65 if (digitalRead(PIN_SW) == 0){
66   button_pressed.data = true;
67   boton.publish(&button_pressed);
68 }else{
69   button_pressed.data = false;
70   boton.publish(&button_pressed);
71   }
72   delay(100);
73 }
74
75 float linearTransform(float x, float x1, float y1, float x2, float y2) {
76   float m = (y2 - y1) / (x2 - x1);
77   return m * (x - x1) + y1;
78 }
79
80 float linearTransformWithDeadZone(float x, float x1, float y1, float deadx1, float deadx2,
       float x2, float y2) {
81
82   if ( x < deadx1 ) {
83     return linearTransform(x, x1, y1, deadx1, 0);
84   }
85   else if ( x > deadx2 ) {
86     return linearTransform(x, deadx2, 0, x2, y2);
87   }
88   else { //(x > deadx1 && x < deadx2)
89     return 0;
90   }
91 }
```