Laboratorio de Principios de Mecatrónica

8 de abril de 2022

Práctica #5

ROS

Grupo: 1

L002

Estudiante:

- García Cortez Alejandro
- Hermida Flores Manuel Joaquín
- Chicatti Avendaño Josué Doménico

Profesor:

Benito Granados–Rojas

Índice

1. Introducción

ROS (Robot Operating System) es un conjunto de herramientas que nos permiten la creación de aplicaciones con robots. Mediante el uso de un simulador llamado *Turtlesim* aprenderemos las bases de este framework.

Para instalar ROS se descargó inicialmente una máquina virtual, en la cual se instaló Ubuntu. Esto se realizó de esta manera, ya que la versión para Windows todavía no es estable. Posteriormente, se realizó la instalación de ROS dentro de la máquina virtual. Ambas instalaciones se lograron al seguir tutoriales.

El objetivo principal es conocer esta plataforma de comunicación y simulación. Primero, se trabajará con una simulación de un robot diferencial en *Turtlesim*. En este caso, la comunicación se realizará entre un *Publisher* (manda información) y un *Subscriber* (recibe información) [1]. Por el momento esta transmisión de información se hace dentro de ROS, pero se tiene miras hacia poder comunicarse con el exterior, específicamente un robot, ya en físico.

2. Experimentos y Simulaciones

2.1. *Turtlesim*

Una vez que se instaló tanto Ubuntu como ROS en la máquina virtual, se procedió a configurar los paquetes necesarios. Dado que la instalación no creo un espacio de trabajo para *catkin*, se realizó su creación manualmente siguiendo los siguientes comandos:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
```

Con el ambiente listo, se configuró el paquete usando los comandos dados en la práctica. Una vez terminada esta configuración, se inició el simulador. Para correr *Turtlesim* se llevaron a cabo los siguientes pasos en la terminal:

```
roscore
rosrun turtlesim turtlesim_node
```

Con el simulador abierto, teníamos disponibles dos comandos muy útiles que servían para borrar y hacer aparecer nuestra tortuga en cierta posición y mirando hacia cierto lado:

```
rosservice call /kill 'turtle1'
rosservice call /spawn x y angle 'turtle1'
```

Lo único que sobraba era hacer correr nuestro programa de Python. Para hacer correr estos programas correctamente, se hicieron dos modificaciones: añadir un 3 a Python y corregir los errores de indentación. Posteriormente, se usaron los comandos:

```
. ~/catkin_ws/devel/setup.bash
rosrun paquete node.py
```

2.2. *Orientate*

Orientate es la función que hace girar al robot tal que esté apuntando hacia una coordenada *(xgoal, ygoal)*.

Para esta práctica, llamamos a esta función cuando queremos que el robot se desplace en línea recta: cuando orientamos a la tortuga hacia un punto antes de moverse hacia este, la tortuga no tiene que hacer una corrección de ángulo, no hay una velocidad angular y la curva de desplazamiento se vuelve una recta.

Esta función primero debe determinar el ángulo al que pondrá a la tortuga. El ángulo se determina simplemente con la función atan2 a la que le ingresamos las dos coordenadas objetivo menos las coordenadas de la posición actual.

```
desired_angle_goal = math.atan2(ygoal-y, xgoal-x)
```

A continuación se determina una diferencia de theta *dtheta* que simplemente es la diferencia del ángulo actual y el ángulo deseado. Para que todos los ángulos nos queden en forma canónica siempre les agregamos 2π si son negativos.

```
if theta < 0:
theta = theta + 2*math.pi

if desired_angle_goal < 0:
desired_angle_goal = desired_angle_goal + 2*math.pi
```

Cuando inicie su recorrido, la tortuga girará con una velocidad angular proporcional a esta diferencia de theta.

```
ka = 1
...
angular_speed = ka * dtheta
```

2.3. *Go to Goal*

Go to Goal es la función que indica al robot a qué posición ir. Esta función comienza igual que la función anterior. Aunque muchas veces se corre una rutina primero orientando y después avanzando, tener esta corrección de ángulo da más robustez al programa y permite que, aunque el robot y el destino no estén en la misma orientación, sea posible llegar.

La parte nueva que introduce esta función se encuentra en el desplazamiento del robot. Para lograr el movimiento comenzamos obteniendo la distancia entre la posición del robot y la posición deseada. Esto se calcula a con la fórmula de distancia euclidiana $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. A partir de esta distancia calculamos la

velocidad lineal multiplicando la distancia por un parámetro.

El *Go to Goal* se ejecuta hasta que el cálculo de la distancia sea menor a 0.1. Una vez llegado a este objetivo se detiene la rutina y se sale de ella. Como en la función de *orientate*, el parámetro de velocidad es transmitido al robot a través del comando *velocity* = *message_linear.x* = *linear_speed*. A continuación se encuentra el código que calcula y transmite la velocidad lineal.

```
kv = 1.0
distance = abs(math.sqrt(((xgoal-x)**2) + ((ygoal-y)**2)))
linear_speed = kv * distance

if (distance < 0.1):
    time.sleep(1)
    break

velocity_message.linear.x = linear_speed
```

2.4. Rectángulo

La primer simulación consistió en programar el robot para que dibujara una trayectoria rectangular. El problema en esta actividad se encontró principalmente en que los métodos de *Go to Goal* y *Orientate* entraban en una singularidad al girar más de 360 grados. Para solucionar esto programamos *if's* que, en caso de detectar ángulo negativos, se sumara 2π para traer el ángulo a positivo.

Una vez solucionado esto la trayectoria se redujo a programar 4 *Goto's* y 4 *Orientate*. Los puntos que visitó el robot fueron, en orden: (8,1), (8,5), (1,5), (1,1). De esta forma el robot regresaba a la posición en que había comenzado.

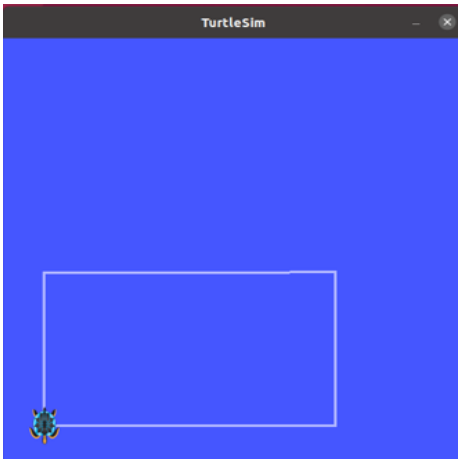

Figura 1: Trayectoria rectangular trazada por el robot.

El código para esta trayectoria es el siguiente.

```
orientate(8.0, 1.0)
go_to_goal(8.0,1.0)

orientate(8.0, 5.0)
go_to_goal(8.0,5.0)

orientate(1.0,5.0)
go_to_goal(1.0,5.0)

orientate(1.0,1.0)
go_to_goal(1.0,1.0)
```

2.5. Circle

Para el segundo ejercicio la tarea fue dibujar un círculo. La manera en la que abordamos esta tarea fue tomar lo que ya habíamos aprendido haciendo el rectángulo y aplicarlo al nuevo caso.

El acercamiento para la resolución de la trayectoria fue hacer un polinomio de N lados tal que fuera aproximando un círculo. Para lograrlo colocamos un par de funciones *orientate*y *go-to-goal* dentro de un ciclo *for* que da N vueltas.

Ambas funciones reciben las mismas coordenadas objetivo. Esto es, para X el destino es la abscisa del centro del círculo más el coseno de i, donde i es el i-ésimo término de un vector lineal que va de 0 a 2π dividido en N pasos.

Mientras que para Y el destino es la ordenada del centro del círculo más el seno de i.

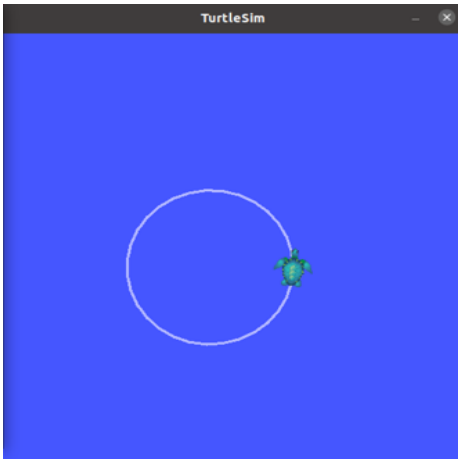

Figura 2: Trayectoria circular trazada por el robot.

```
n = 32
for i in np.linspace(0, 2*(math.pi), n):
```

```
xg = 5 + 2*math.cos(i)
yg = 5 + 2*math.sin(i)
orientate(xg, yg)
go_to_goal(xg, yg)
```

Encontramos que la trayectoria se aproxima más a un círculo cuando N vale más de 30.

3. Conclusiones

Con la práctica se pudo conocer de manera efectiva los básicos de ROS. Se le comunicó a una tortuga (robot) posiciones a las cuales dirigirse y ángulos a los cuales orientarse. De igual manera, se recibia la posición actual de la tortuga de manera constante.

Para realizar las distintas actividades de la práctica fue necesario utilizar conocimientos de geometría analítica. Añadimos un valor despreciable a la función de arcotangente para evitar que se fuera a infinito y corregimos ángulos negativos sumando 2π. También representamos una circunferencia usando seno y coseno. Finalmente, aprendimos que era necesario mantener controlada la velocidad y dar retrasos para que la tortuga pudiera moverse y orientarse de manera correcta.

Uno de los grandes retos de esta práctica fue la instalación de las distintas herramientas. Las descargas se tuvieron que adaptar a las características de cada máquina. Además, fue necesario realizar pasos adicionales para configurar de manera correcta ROS y sus paquetes. Aún así, es muy útil irse familiarizándose con estas herramientas que serán usadas posteriormente.

4. Enlaces externos

`https://github.com/ManoHF/lab-mecatronica`

5. Referencias

[1] “Exchange Data with ROS Publishers and Subscribers,” Mathworks.com, 2022. https://www.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html (accessed Apr. 07, 2022).