

LABORATORIO DE PRINCIPIOS DE MECATRÓNICA

8 de abril de 2022

Práctica #5

Robot Operating
System

Grupo:

L001

Estudiantes:

- Bermúdez Guillen
Alejandro
- Caballero Enciso
Carla
- Montes de Oca Villa
Luciano

Profesor:

Benito Granados-Rojas

Índice

1. Introducción	2
2. Experimentos y Simulaciones	2
2.1. Experimento 2.1: Cuadrado	2
2.2. Experimento 2.2: Círculo	5
3. Conclusiones	6
4. Enlaces externos	6
Referencias	6

The logo of the Instituto Tecnológico y de Estudios Superiores de Occidente (ITAM) is displayed. It consists of the letters "ITAM" in a bold, white, sans-serif font, centered within a solid dark green square.

1. Introducción

Robot Operating System (ROS) es un middleware robótico, es decir, una colección de frameworks para el desarrollo de software de robots. A pesar de no ser un sistema operativo, ROS provee los servicios estándar de uno de estos tales como la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes (Ortego Delgado, 2017).

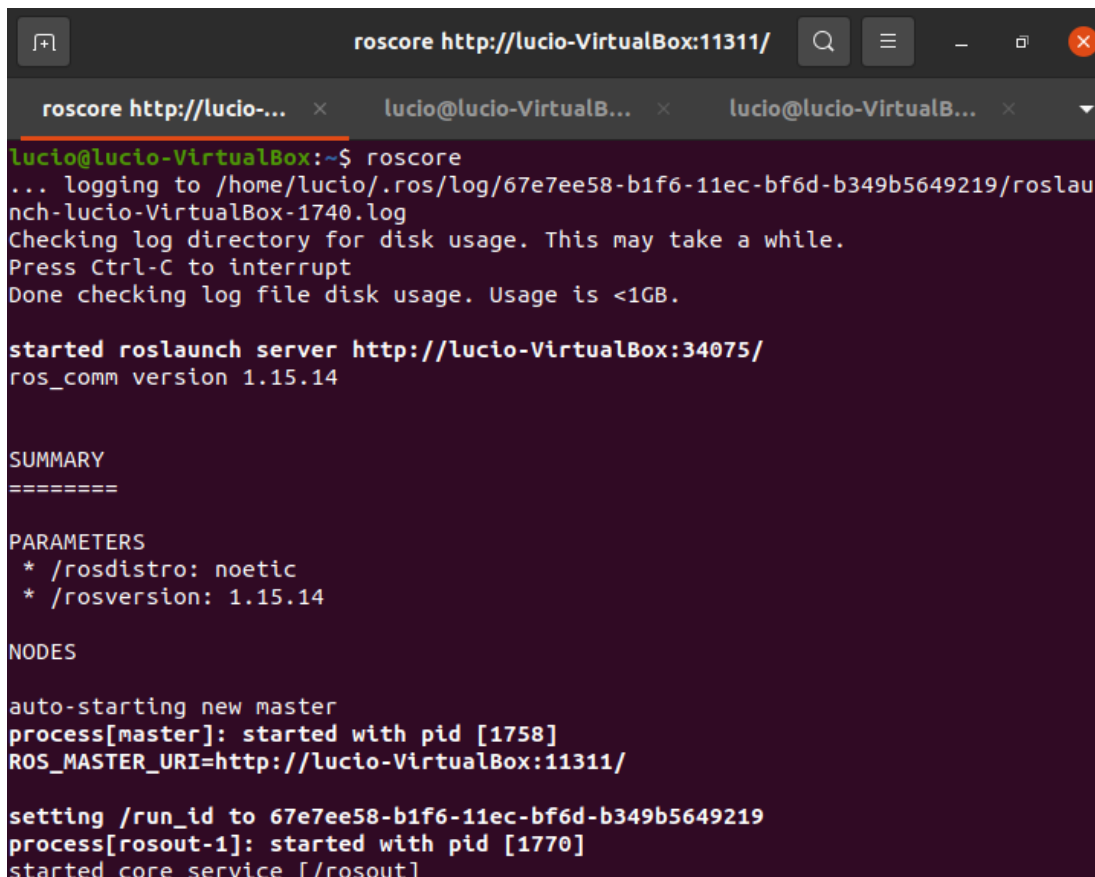
En esta práctica, utilizaremos ROS para tratar de acercarnos un poco al mundo de simulación de agentes móviles en un entorno específico. Además, con la realización de los ejercicios de esta práctica, seremos capaces de realizar el modelado y control de un robot móvil de tipo diferencial, así como distinguir las características de ROS como una plataforma de comunicación, control y simulación para aplicaciones de Robótica.

Recordemos que la Robótica es una rama de la Mecatrónica, ambas son multidisciplinarias por lo que se complementan una a la otra y esta práctica es fiel evidencia de esta afirmación (GES Comunicación, 2019).

2. Experimentos y Simulaciones

2.1. Experimento 2.1: Cuadrado

Para este primer ejercicio, comenzamos abriendo una terminal en la máquina virtual de Linux “Ubuntu 20.04” y ejecutamos el comando “*roscore*” para inicializar ROS, previamente instalado en la computadora.



```
roscore http://lucio-VirtualBox:11311/
roscore http://lucio-... x lucio@lucio-VirtualB... x lucio@lucio-VirtualB... x
lucio@lucio-VirtualBox:~$ roscore
... logging to /home/lucio/.ros/log/67e7ee58-b1f6-11ec-bf6d-b349b5649219/roslau
nch-lucio-VirtualBox-1740.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://lucio-VirtualBox:34075/
ros_comm version 1.15.14

SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [1758]
ROS_MASTER_URI=http://lucio-VirtualBox:11311/

setting /run_id to 67e7ee58-b1f6-11ec-bf6d-b349b5649219
process[rosout-1]: started with pid [1770]
started core service [/rosout]
```

Imagen 1. Ejecución de ROS.

Posteriormente, abrimos una segunda terminal para ejecutar el comando “***roslaunch turtlesim turtlesim_node***” que abrirá una interfaz para el simulador Turtlesim.

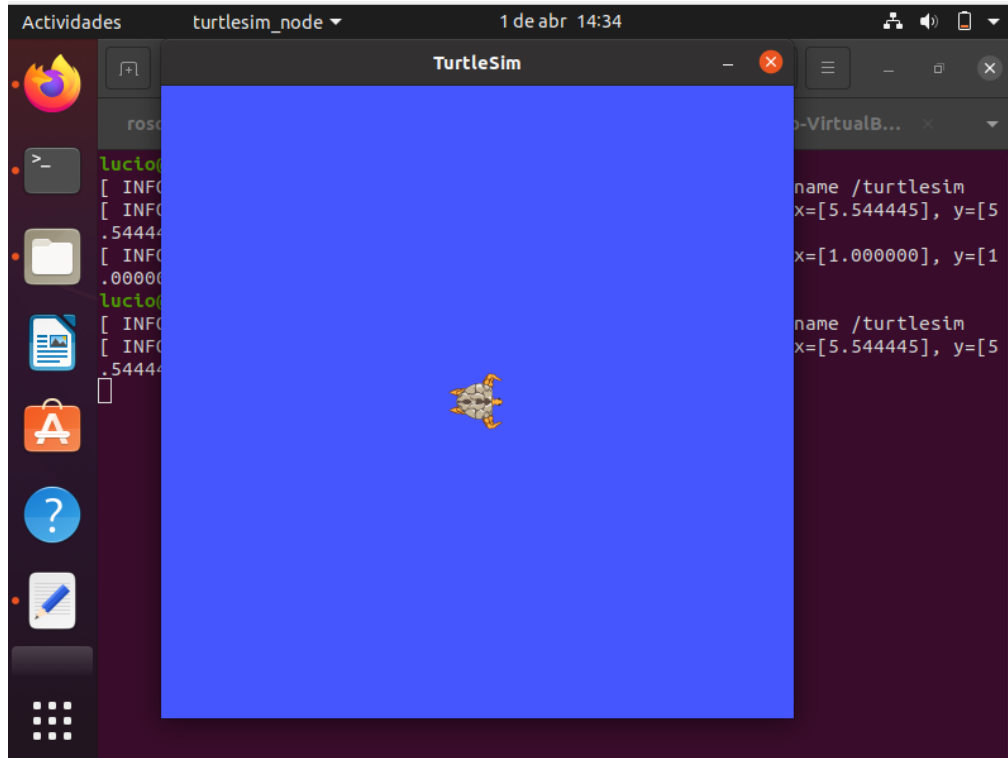


Imagen 2. Simulador Turtlesim.

Después, en una tercera terminal ejecutamos los servicios de Ros para manipular la posición o interactuar con la tortuga. Primero, el comando “***rosservice call /kill 'turtle1'***” borró la tortuga que aparece por defecto cuando abres el simulador Turtlesim. Segundo, con el comando “***rosservice call /spawn 1 1 0 'turtle1'***” instanciamos una nueva tortuga y la posicionamos en la coordenada (1,1) mirando al ángulo 0 en radianes (establecimiento de la posición inicial).

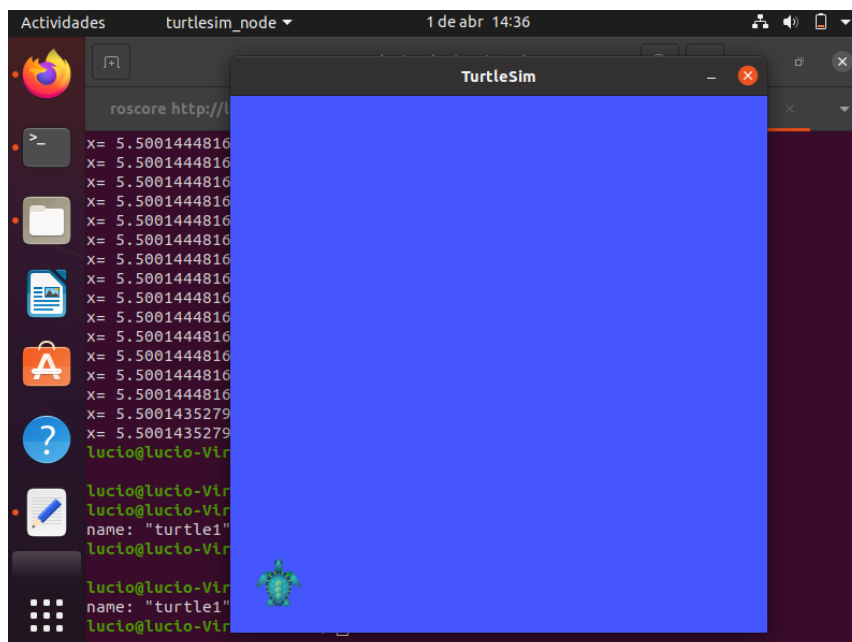


Imagen 3. Estableciendo posición inicial.

Pasando con los ejercicios, en primer lugar, dibujamos simplemente una línea recta y esto se logró con la ejecución del comando “**roslaunch paqueteprueba líneaRectaRos.py**” donde únicamente modificamos el código inicial “**rosmove0.py**” y orientamos a la tortuga en el (1,1) utilizando el método **orientate(1,1)** y luego la desplazamos a la coordenada (10,1) con el método **go_to_goal(10,1)**. El código de este ejercicio que lleva por nombre “**líneaRectaRos**” se encuentra en GitHub (véase *Enlaces Externos*).

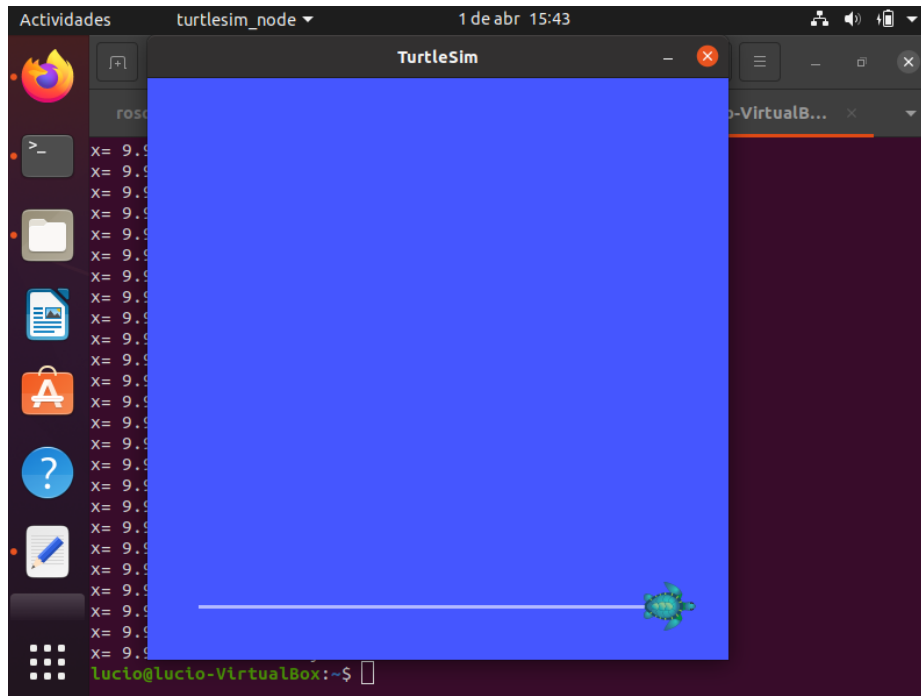


Imagen 4. Trazado de una línea recta.

Posteriormente, para dibujar un cuadrado tuvimos que hacer algunas modificaciones a los algoritmos **orientate** y **go_to_goal** ya que estábamos teniendo problemas con atan2 y con los ángulos del cuadrado.

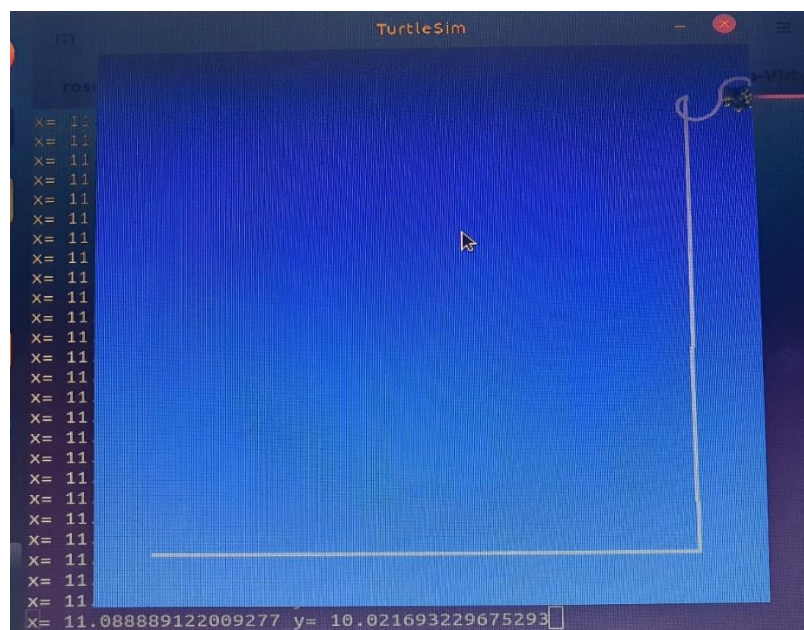


Imagen 5. Problemas con tangentes y ángulos negativos

Para solucionar estos problemas, buscamos cómo evitar que atan se fuera a infinito y empleamos un caso especial para poder trabajar adecuadamente con ángulos negativos. Finalmente, logramos completar la trayectoria cuadrada usando cuatro veces las funciones **orientate** y **go_to_goal** en el main, donde los parámetros se trataron de las 4 esquinas del cuadrado que dibujamos. El código de este ejercicio que lleva por nombre “**cuadradoRos**” se encuentra en GitHub (véase *Enlaces Externos*).

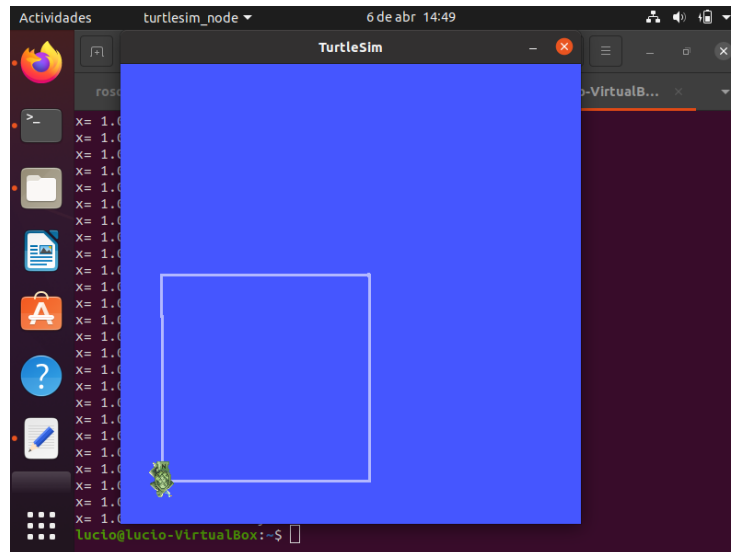


Imagen 6. Trazado de una trayectoria cuadrada

2.2. Experimento 2.2: Círculo

En este segundo ejercicio, lo que hicimos fue modificar el main para crear un linspace dentro de un ciclo for e ir trasladando a la tortuga dependiendo del seno y coseno de los radianes que obtuviéramos. Estos serían los parámetros de **orientate** y **go_to_goal**. En cada **x_deseada** y **y_deseada**, además de calcular el seno y coseno del ángulo, también le sumamos el radio del círculo. Estas funciones no sufrieron nuevas modificaciones respecto al cuadrado. El código de este ejercicio que lleva por nombre “**circuloRos**” se encuentra en GitHub (véase *Enlaces Externos*).

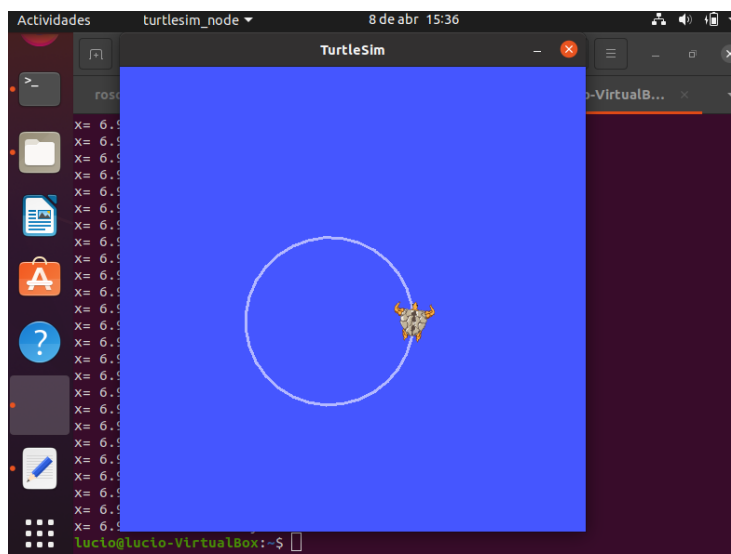


Imagen 7. Trazado de una trayectoria circular.

3. Conclusiones

Esta práctica fue un poco complicada pero quizás no tanto como las anteriores. Lo que nos generó mayores dificultades fue la instalación de la máquina virtual de Ubuntu y del programa ROS ya que este último no admite todas las versiones de Ubuntu y se ejecutan muchos comandos para que corra y quede bien instalado. Lo que esta vez realizamos fue muy distinto a lo de las cuatro prácticas previas debido a que involucró herramientas que nunca habíamos utilizado, es decir, ROS pero afortunadamente logramos los ejercicios correspondientes sin ningún problema. La programación, al ser en Python, no hubo mucha complicación porque ya teníamos algo de conocimiento del lenguaje. Aunque fue una gran ayuda tener un código base para los problemas a solucionar, lo verdaderamente difícil fue ver que es lo que teníamos que hacer y poder comprender a detalle qué hacía cada parte del código que se nos proporcionó. Al final, la práctica se realizó con éxito.

4. Enlaces externos

<https://github.com/Lucio27MV/Lab-Principios-Meca>

Referencias

- GES Comunicación. (30 de Septiembre de 2019). *Mecatrónica y robótica, ¿son lo mismo?* Obtenido de Galileo Universidad: <https://www.galileo.edu/trends-innovation/mecatronica-y-robotica-son-lo-mismo/#:~:text=La%20rob%C3%B3tica%20es%20la%20rama,colaboraci%C3%B3n%20entre%20robots%20y%20personas>.
- Ortego Delgado, D. (21 de Septiembre de 2017). *Qué es ROS (Robot Operating System)*. Obtenido de OpenWebinars: <https://openwebinars.net/blog/que-es-ros/>