Reporte Práctica 5

Mildred Limón Santamaria
177229

Renata Monsalve Rubí
176371

***Abstract*—ROS es una red de conexiones entre nodos y temas (a los que se conectan los nodos como suscriptores o editores) orientada a robots, que permite la captura de información para computarla y realizar alguna tarea, la cual puede ser a su vez publicada para realizar un proceso continuo automatizado. En este caso hemos usado el simulador de la tortugas de ROS y script en python como nodos, conectados mediante tópicos de posición y velocidad.**

I. Introduction

La automatización y programación de robots requiere de varios recursos de software y hardware utilizados de forma simultánea. Una de las herramientas de mayor uso en la actualidad, para dar estructura a estos sistemas, es ROS, el cual es un marco de trabajo que permite la transmisión de mensajes (que pueden ser señales obtenidas por sensores), control de dispositivos de bajo nivel, control de hardware por medio de scripts y, por tanto, la automatización de procesos.

Entender el funcionamiento de ROS permite la conección de varios elementos, que ni siquiera requieren estar físicamente presentes con un radio máximo de distancia o que pueden, incluso, ser enteramente digitales. Mediante el uso de Nodos y temas o tópicos, se puede regular y filtrar la información que se accede y usa (de entre toda la disponible), incluso a un mismo tema generado por un editor, pueden estar suscritos diferentes lectores y, a pesar de ello, cada sistema puede procesar con diferentes propósitos y modos la información leída.

Un ejemplo de estos sistemas y nodos puede ser fácilmente entendible mediante turtlesim. Este es instalable en ros y al correrse genera una tortuga que se desplaza sobre un área de trabajo, lo que funciona de forma perfecta como un simulador de un robot con ruedas o móvil a discreción propia. Este se puede conectar a un script de Python o C++ para programar sus acciones o automatizar un proceso. Mediante el uso de esta herramienta, el siguiente trabajo busca entender el funcionamiento de ROS, los nodos y los temas. Así como probar el uso de scripts para la programación de autómatas.

II. Marco Teórico

El sistema operativo robótico o ROS (del inglés Robot Operating System) es un marco de trabajo para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. ROS provee servicios como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Además cuenta con funcionalidades de alto nivel como llamadas síncronas y asíncronas, base de datos centralizada y un sistema de configuración para el robot. La arquitectura de ROS está basado en grafos; donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estado, planificaciones, actuadores, etc.

Para entender más del funcionamiento de ROS debemos entender el sistema basado en nodos. En este contexto, un nodo es cualquier hardware o software con el cual se puede interactuar. ROS define un estándar de comunicación entre los diferentes nodos que comprenden el sistema final. Los mecanismos de comunicación que incorpora ROS: suscripción/publicación y servicios. La suscripción/publicación es el mecanismo de comunicación más habitual y se basa en el envío y recepción de mensajes; ya sea la publicación o la suscripción de mensajes se hace a través de tópicos, que son nombres asociados a un tipo de mensaje concreto. El mecanismo de servicios se realiza mediante la conexión entre dos elementos de forma exclusiva; durante esta conexión solo ambos extremos recibirán los mensajes enviados. De esta manera, podemos decir que un nodo es una pieza que dispone de un mecanismo de comunicación con ROS mediante mensajes y que siempre puede tener un mecanismo de interconexión con elementos externos. Si un nodo sólo se dedica a recibir información, realizara alguna operación y devolverá un resultado, sin necesidad de una comunicación externa. También es importante destacar que ROS define una lista amplia de formato de mensajes que podrían ser utilizados por los nodos. El uso de tipo de mensaje estándar nos permite la comunicación con nodos.

Los cinco principios de ROS son: punto-a-punto, arquitectura que permite que los nodos puedan dialogar directamente con cualquier otro de forma síncrona o asíncrona; multilenguaje, ROS puede programar en varios lenguajes de programación; basado en herramientas, cuenta con un diseño de micro kernel que utiliza una gran cantidad de herramientas para construir y ejecutar sus diversos nodos; ligero, los algoritmos se empaquetan en ejecutables independientes reutilizables y de tamaño reducido; gratuito y de código abierto. Otros aspectos importantes de ROS es su sistema de archivos, que está organizando de forma jerárquica, donde destacan los terminos paquete y pila. El paquete es un directorio que contiene nodos, bibliotecas externas, datos y archivos de configuración. La

pila es un directorio que contiene paquetes.Cómo se mencionó anteriormente, ROS permite ejecutar gran cantidad de nodos en paralelo que requieren intercambiar datos. El responsable de alcanzar ese objetivo es el grafo de cómputo de ROS, que involucra los conceptos maestro, nodo, tópico, servicios y mensaje. El maestro es un servicio de declaración y registro de nodos, que hace posible que los nodos se encuentren e intercambien datos.

III. Desarrollo

A. Instalación del espacio de trabajo

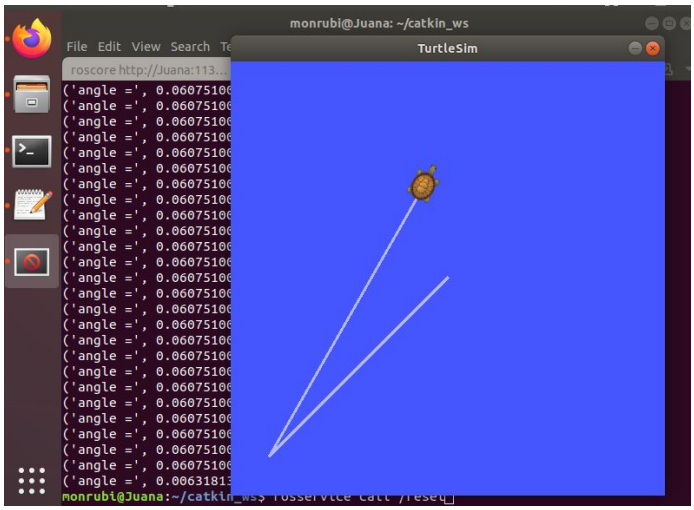
Debido a que ROS trabaja exclusivamente con Linux se requería tener acceso a una máquina con Ubuntu. Para ello se hizo uso de Oracle VM Virtualbox. Esta herramienta nos permite generar máquinas virtuales incluyendo Ubuntu 18 (última versión). Ya instalado se requería instalar ROS; se intentaron diferentes versiones de ROS, finalmente la que funcionó fue Melodic. Después se requería instalar python, sin embargo, la actualización o instalación de Python 3 des habilitaba la terminal, sin terminal tampoco se podía regresar a una versión previa de python, por ello se tuvo que reiniciar el proceso desde el inicio. Finalmente, tras intentos y error se consiguió hacer correr la quinta máquina virtual que se intentó con Ubuntu 18, ROS melodic y Python 2.7.

Ya con ROS instalado, se siguió la guía de ROS Wiki para preparar el workspace (guardado como catkin_ws) del profesor y se configuró el primer paquete, para trabajar con scripts de C++ y Python, usando la guía del profesor. Se instaló el turtlesim y tras arrancarlo como nodo principal se probó con el script de rosmove3.py y se hicieron adaptaciones para que pudiese hacer: líneas rectas, un cuadrado (o rectángulos) y finalmente un círculo.

Para ello debe poder comunicarse con el turtlesim, esto se consigue mediante los topics, se conecta como suscriptor a “position_topic” (del cual recibe la posición actual de la tortuga conforme se mueve) y como editor a “cmd_vel_topic” (mediante el cual comunicará a la tortuga el ángulo y velocidad al que debe moverse. Usando estos temas se puede ubicar la tortuga y definir los valores que se deben publicar para moverse hacia un objetivo dado. Los scripts dados por el profesor incluyen los métodos de “orientate” que giran a la tortuga y “go_to_goal” lo que la traslada ajustando en cada movimiento su ángulo de orientación.

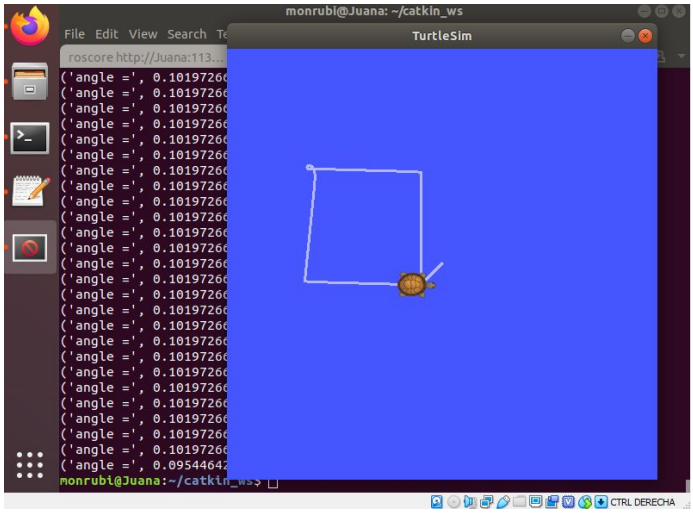
B. Líneas rectas

Las líneas se podían correr con el código dado. Servía para verificar la conección al turtle sim y la correcta instalación del paquete y lectura del script. Se hizo que la tortuga hiciera un trayecto a las coordenadas (1,1,0) y luego a (5,8,0). El primer valor es X, el segundo es Y y el tercero es Z. Aquí notamos que se desplaza en X y Y pero rota sobre Z. Aquí notamos que al acercarse al punto objetivo la tortuga desacelera. Para volver más rápido el proceso se cambió por una velocidad constante. También se retiraron los time.sleep() que generan pausas, ya que el código continúa el incremento en la posición de la tortuga pero sin publicarlo para actualizarlo, lo que repercute en que la tortuga cree estar en un lugar distinto a aquel en el que realmente está posicionada.



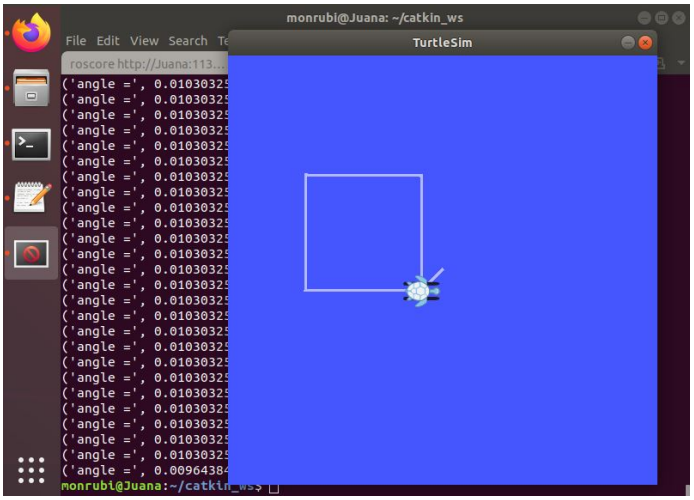
C. Cuadrado

Aquí notamos que no podía simplemente establecerse las 4 coordenadas de un cuadrado y hacer que la tortuga se moviese. El cuadrado que se obtenía estaba torcido y en la segunda esquina no giraba hacia la izquierda, sino que caminaba a su derecha y enderezaba la trayectoria sobre la marcha de su avance. Debido a eso se implementaron dos cambios: antes de iniciar el script se transporta la tortuga a las coordenadas (5,5,0) como coordenadas de inicio. Para arreglar el giro, se evaluó el código.



El ángulo que se gira corresponde a la resta entre el ángulo deseado y el ángulo actual, sin embargo, si dicho valor es inferior a 0.1 se detiene. Ello implica que los valores negativos no son procesados y por tanto, si el ángulo deseado es menor que el actual, la tortuga no girará. Se agregó un if que invierte el orden de la resta si el resultado es negativo ya que esto conserva el propósito de conseguir la diferencia entre los

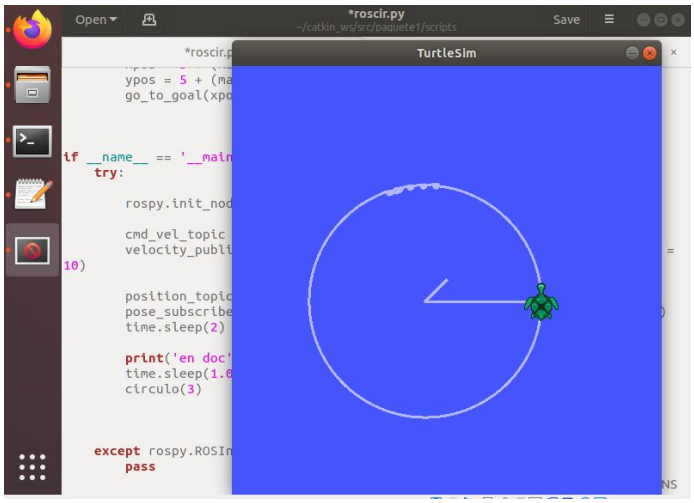
ángulos pero que no afecta el punto de detención marcado en valores menores a 0.1.



D. Círculo

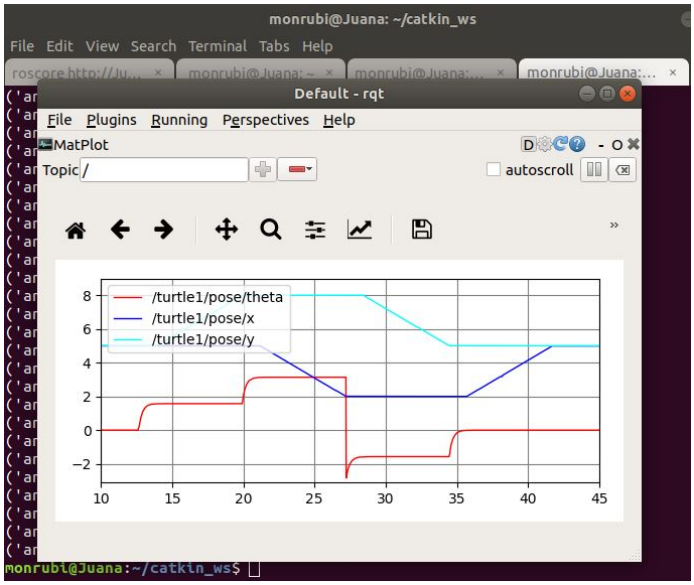
El círculo se construyó para poder resolver radios aleatorios (sin filtrar que quepan en el tablero de la tortuga, se asume un usuario con conocimiento del espacio de trabajo). Dado que el "go_to_goal" ya incluye una redirección de la orientación de la tortuga se pudo evitar usar el "orientate. Simplemente se estableció un avance en grados y se dividió la totalidad del círculo en esos grados, con lo que obtenemos el número de veces que la tortuga deberá completar un recorrido, es decir, el muestreo de puntos sobre la circunferencia. A menor tamaño de avance mayor curvatura pero mayor peso de procesamiento. Se determinó trabajar con un valor de 5 grados. Se inicia con un ángulo 0 y se mueve la tortuga al punto de inicio y se orienta contra reloj. Luego, con cada iteración se aumenta el ángulo acumulado, el valor de X y Y se obtienen mediante coseno y seno respectivamente y se multiplican por el radio del círculo para hacer el escalamiento del vector. Finalmente, dado que no se gira en torno a cero sino a (5,5,0), se suma el centro a los valores obtenidos para hacer la traslación del punto sobre el plano.

Aquí nos topamos con un inconveniente, la función del arcotangente que nos da la theta que se girará la tortuga pasa de negativo infinito a negativo positivo al calcular cerca de PI, es decir, pasa de +Pi a -Pi de forma contínua, como si la tortuga vibrara. Esto ocasiona que la tortuga avance dando vueltas sobre sí misma cuando su ángulo es PI. Dado que la trayectoria general se conserva, y que aún no conseguimos establecer una solución al problema, se ha dejado así. Los intentos por usar un valor absoluto y casos no resultaron efectivos, pero este error será corregido para la entrega final del proyecto.

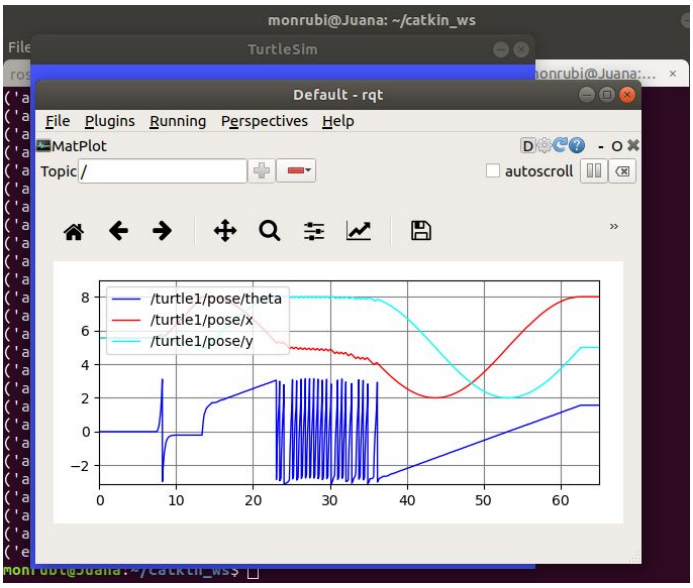


RESULTADOS

Una vez obtenidas las trayectorias se instaló rqt en conjunto con varios de sus plug ins. Estas son librerías open-source. Tras inicializar el nodo del turtlesim se inicia, en una nueva pestaña de la terminal, rqt. Se prepara la posición de la tortuga transportándola (en vez de hacer un /kill y /spawn, simplemente porque no afecta el resultado final y es una línea menos de código) y se abre en el GUI el plugin de visualización¿plot y el de NodeGraph. Se corre el script a graficar y se ajustan los valores del plot para que ocupen todo el espacio disponible, eso facilita su lectura. Es notorio en el movimiento del círculo el problema encontrado en el inciso anterior, aquí la línea correspondiente a Theta nos muestra esta variación entre Pi y -Pi, pero la conservación de la trayectoria general de X y Y.

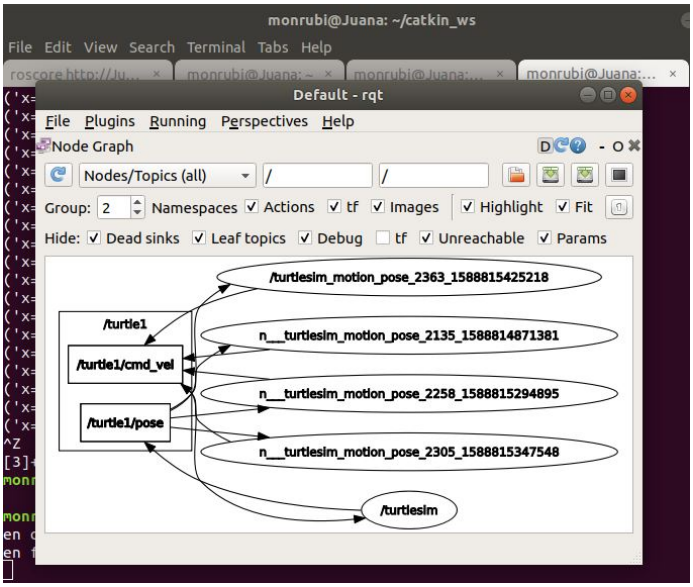


rqt_plot del cuadrado



rqt_plot del círculo

En el caso del Nodo se conecta a los mismos tópicos para funcionar en ambos códigos fuente, por ello las conecciones se pueden evaluar en una única gráfica.



rqt_graph

Conclusiones

- ROS es un sistema versátil. La creación sin restricción de número de "topics" y nodos permite que un sistema pueda conectarse a un amplio número de sensores, captar en tiempo real la información proporcionada por ellos y actuar de forma automatizada. El resultado final es un sistema que puede tener diferentes escalas de complejidad.
- Si bien ser open-source otorga a las librerías una mayor amplitud de temas y funcionalidades, el hecho de que no se ha centralizado ni normado ha generado que haya varias versiones de ROS y que no todas las librerías funcionen con todas las versiones de ROS, además de que alguna están tan desactualizadas que no funcionan con librerías nuevas o que las librerías no funcionan con versiones más recientes de ROS. Todo ello genera un detrimento en el sistema y aumenta la dificultad de su uso.
- El sistema final no depende únicamente de ROS y de la correcta lectura y transmisión de datos. Fue notorio en el desarrollo del código del círculo, que la mitad del problema a resolver está conformado por la parte de computación más que mecatrónica. Los algoritmos deben ser óptimos para manejar y filtrar la información de tal manera que el comportamiento del sistema sea realmente el esperado.

References

[1] Wiki.ros.org. 2020. Turtlesim - ROS Wiki. [online] Available at: http://wiki.ros.org/turtlesim [Accessed 1 May 2020].

[2] Wiki.ros.org. 2020. rqt - ROS Wiki. [online] Available at: http://wiki.ros.org/rqt [Accessed 1 May 2020].

[3] Wiki.ros.org. 2020. Ubuntu install of ROS Melodic - ROS Wiki. [online] Available at: http://wiki.ros.org/melodic/Installation/Ubuntu [Accessed 6 April 2020].

[4] Wiki.ros.org. 2020. Creating a ROS Package - ROS Wiki. [online] Available at: http://wiki.ros.org/ROS/Tutorials/CreatingPackage [Accessed 6 April 2020].