Taller 2 Robótica

Santiago Martínez Castaño
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá
s.martinezc@uniandes.edu.co

Martin David Galvan Castro
Departamento de Ingeniería
de Sistemas y Computación
Universidad de los Andes, Bogotá
md.galvan@uniandes.edu.co

Santiago Montaño Díaz
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá
vs.montano@uniandes.edu.co

I. Introducción

La solución del Taller 2 permite familiarizarse con el manejo de datos obtenidos de un láser para visualizar los obstáculos que un robot puede observar desde su marco de referencia y cómo estos se trasladan al marco de referencia global. También se estudia una primera aproximación de control de posición por una ley de control proporcional, obteniendo como resultado la posibilidad de trasladar el robot de un punto a otro del espacio. El desarrollo está basado al rededor de una simulación en V-REP para el robot Turtlebot2, para esta se han creado nodos que permiten llevar el robot de un punto a otro en su ambiente y graficar su posición en el espacio.

El presente informe se divide en 2 partes principales: En la sección II se presenta el desarrollo de los nodos mencionados anteriormente, así como su conexión a la simulación y una breve explicación a manera de macroalgoritmo. Esta sección permitirá entender cómo se ha desarrollado la solución a los problemas planteados sin entrar en mayor detalle del código. Luego, en la sección III se explicará al usuario cómo correr la simulación y comprobar el funcionamiento de los nodos desarrollados, así como las herramientas adicionales requeridas para su correcta ejecución.

II. Desarrollo

*II-A. **Punto 1 - Datos de un Laser***

Para el desarrollo de este punto se hizo un script llamado Punto1.py. En este script se definieron las posiciones del robot con respecto al marco inercial y del laser con respecto a el marco del robot que son $\xi_R = [1m, 0{,}5m, \frac{\pi}{4}]_I^T$ y $\xi_L = [0{,}2m, 0m, 0]_R^T$ respectivamente. Seguido a esto se hace la carga del archivo 'laserscan.dat' y se crea un arreglo del mismo tamaño de datos del archivo que contiene los ángulos α_i en el intervalo $[\frac{-\pi}{2}, \frac{\pi}{2}]$. Además, se definieron funciones para calcular las rotaciones entre marcos $R(\theta)$ y $R^{-1}(\theta)$. Al ejecutar el script se hacen dos gráficas, la de medición del láser contra ángulo y la ubicación del robot con los las mediciones del láser. Para esto se hicieron las siguientes operaciones:

$$\xi_{Láser_I} = [1m, 0{,}5m, \frac{\pi}{4}]_I^T + R^{-1}(\frac{\pi}{4})[0{,}2m, 0m, 0]_R^T \quad (1)$$

$$\xi_{Obstáculo_{i_I}} = \xi_{Láser_I} + medición_i \times [cos(\alpha_i), sin(\alpha_i), 0]^T \quad (2)$$

Con la ecuación 1 se encontró las coordenadas del láser en el marco global, y con la ecuación 2 se encontró la posición de la medición i del láser en el marco local. La Figura 1 las mediciones del láser para diferentes ángulos del intervalo anteriormene mencionado, se puede apreciar que cuando el ángulo se acerca a 0, la distancia aumenta, lo que refleja que el robot debe estar ubicado en algún pasillo y que la distancia del robot a cualquiera de las paredes es de mas o menos 1.5m. La discontinuidad que se aprecia debe ser por que el robot se esta aproximando a una esquina y debe hacer un giro.

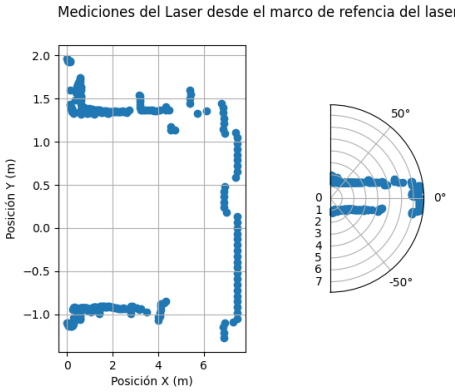

Figura 1. Literal A. Mediciones del Láser vs Ángulo

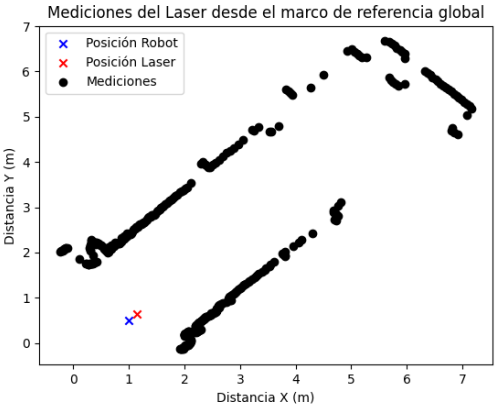

Figura 2. Literal C. Gráfica del Mundo

En la Figura 2 se evidencia lo afirmado anteriormente. En esta el robot es representado por una X azul, el láser por una X roja y los puntos negros son las mediciones del láser. Se puede ver que efectivamente el robot esta en un pasillo y que este tiene un giro a la izquierda.

II-B. Punto 2 - Odometría

El algoritmo siguió la lógica de la ecuación que se observa a continuación:

$$\begin{pmatrix} X' \\ Y' \\ \theta' \end{pmatrix} = \begin{pmatrix} cos(\omega\Delta_t) & -sin(\omega\Delta_t) & 0 \\ sin(\omega\Delta_t) & cos(\omega\Delta_t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_I - ICC_x \\ y_I - ICC_y \\ \theta_I \end{pmatrix} + \begin{pmatrix} ICC_x \\ ICC_y \\ \omega\Delta_t \end{pmatrix} \tag{3}$$

donde el vector $\begin{pmatrix} X' \\ Y' \\ \theta' \end{pmatrix}$ corresponde a la posición del robot en el marco de referencia global, ω y el ICC_x, ICC_y se calcularon como se observa a continuación:

$$\begin{aligned} w &= \frac{(V_r - V_l)}{l} \\ ICC_x &= x - Rsin(\theta) \\ ICC_y &= y + Rcos(\theta) \end{aligned} \tag{4}$$

donde x y y corresponde a las coordenadas iniciales y R a:

$$R = \frac{l}{2}(\frac{V_r + V_l}{V_r - V_l}) \tag{5}$$

V_r y V_l corresponden a las velocidades de la rueda derecha es izquierda del robot respectivamente.

Para el desarrollo de este punto, se desarrolló el nodo el cual tenia como función leer el archivo .txt para obtener la información sobre los perfiles de velocidades, después los iba a organizar en un arreglo como se puede visualizar en la Figura 3.

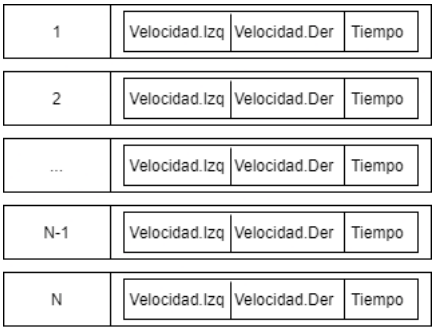

Figura 3. Estructura de datos para guardar archivo .txt

Para publicar los mensajes se siguió el algoritmo descrito en la Figura 4, la publicación de los mensajes se hacia en el tópico **/turtlebot_wheelsVel**. Para graficar la posición del robot en tiempo real respecto a los datos obtenidos por el tópico **/turtlebot_position** y las posición calculada por odometria se tuvo que ajustar el algoritmo de 4, para que dentro del ciclo de publicación se actualizara la gráfica y la

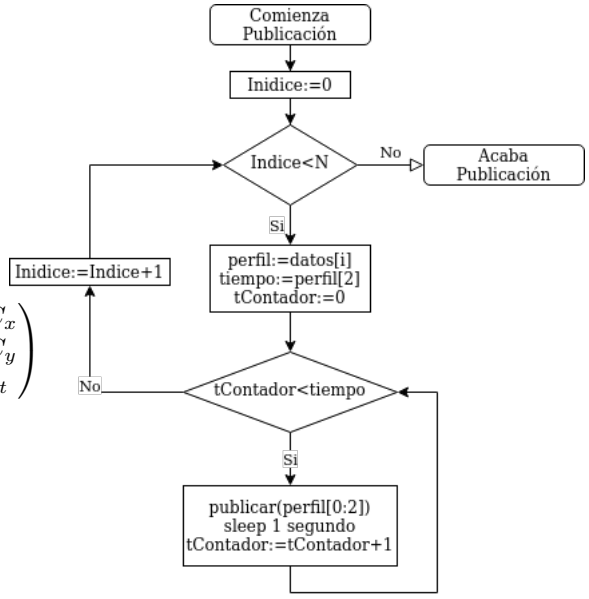

Figura 4. Estructura de datos para guardar archivo .txt

función para actualizar la posición del robot calculada por odometria. El algoritmo final queda representado en la Figura 5

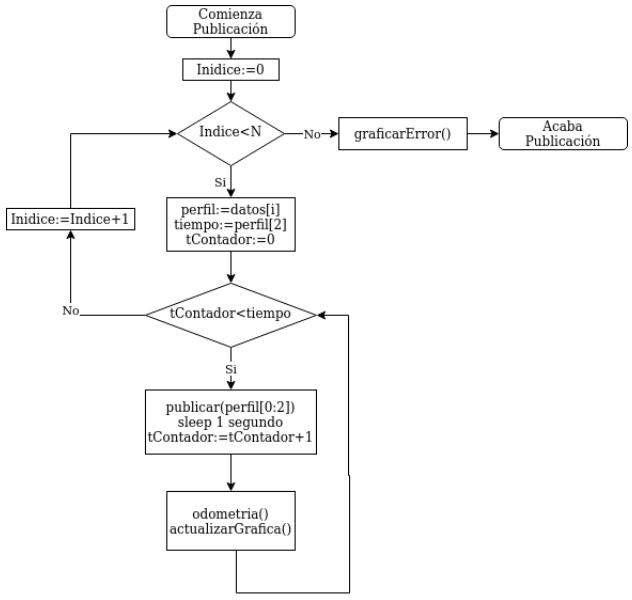

Figura 5. Estructura de datos para guardar archivo .txt

Para la implementación se declararon 3 Listeners, uno al topico **/turtlebot_position**, al topico **/turtlebot_orientation** y **/simulationTime**. Para el primero la función de callback almacenaría la posición actual del robot en un arreglo, para el segundo, su función de callback guardaría la orientación del robot y para el ultimo, su función guardaría el tiempo. Una vez obtenidos estos datos, el algoritmo de publicación actualizaría la gráfica cada ciclo. Para generar la gráfica se hizo uso de la

librería *Matplotlib*. En la Figura 6 se puede ver el grafo del nodo

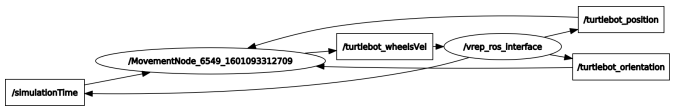

Figura 6. Grafo de comunicación Punto 2

Los resultados obtenidos del archivo *velPrueba.txt* Se pueden ver en la Figura 7, Figura 8 y Figura 9

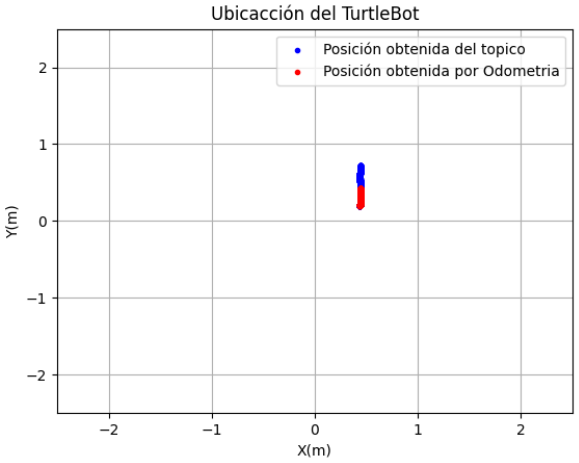

Figura 7. Posiciones TurtleBot linea recta

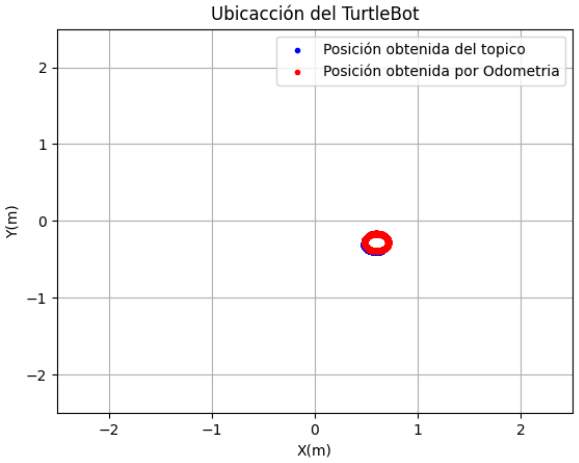

Figura 8. Posiciones TurtleBot curva

Se puede observar que el calculo realizado por odometría sigue la misma trayectoria de las posiciones mandadas por el tópico **turtlebot_orientation**, esto se puede observar en la gráfica del error y la gráficas de posición, al correr la simulación se observo que las gráficas calculadas por odometría se anticipaban a las del tópico, esto se puede deber a a las

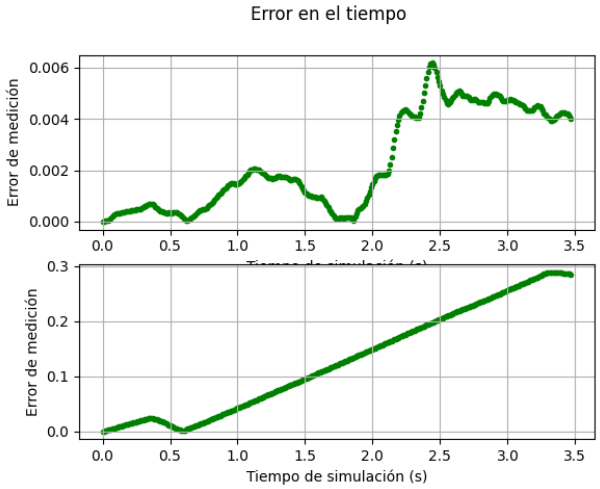

Figura 9. ErrorTurtleBot linea recta

diferencia del tiempo y frecuencia de muestreo entre V-rep y el nodo de odometria.

II-C. Punto 3- Control

Para controlar el robot se creó el nodo **turtle_bot_control**, este nodo se encarga de calcular la velocidad lineal y angular que debe tener el robot diferencial trabajado para que logre una posición final (P_F) especificada por parámetro. Para lograr esto se aplicó un control proporcional, el cual se realizó en 3 etapas. Considere un robot diferencial ubicado en la posición (x, y) y con orientación θ.

Etapa de control 1: Orientación de movimiento

Ya que un robot diferencial solo puede moverse en la dirección X de su marco local de referencia, se ha optado por realizar una primera etapa en la que el robot rota hasta quedar en la dirección en la cual si avanza hacia adelante llegará al punto P_F moviéndose en línea recta. La ecuación que modela el error de orientación α para esta etapa es la siguiente:

$$\alpha = arcTan\left(\frac{dy}{dx}\right) - \theta \tag{6}$$

Dónde dx y dy son las distancias que se deben recorrer desde la posición actual hasta P_F en el eje horizontal y vertical, respectivamente.

Etapa de control 2: Distancia hasta P_F

En esta etapa se busca reducir la distancia entre la posición actual del robot y el punto deseado P_F. La ecuación que modela el error de distancia ρ es la siguiente:

$$\rho = \sqrt{dx^2 + dy^2} \tag{7}$$

Así, cuando el robot se mueva en dirección de P_F se reducirá dx y dy llevando este error a 0.

Etapa de control 3: Orientación final deseada

En la última etapa el robot ya se encuentra en la posición x_F, y_F deseada y ahora se busca orientar el robot en la dirección deseada. Para esto se modela el error angular de orientación de la siguiente forma:

$$\beta = \theta_F - \theta \tag{8}$$

Dónde θ_F representa la orientación final deseada y θ la orientación actual del robot.

Con los errores α, ρ y β calculados se procede a calcular las velocidades lineales y angulares que gobiernan el control del robot. Estas ecuaciones de velocidad son las siguientes:

$$\dot{\alpha} = -K_\rho \sin(\alpha) + K_\alpha \alpha \tag{9}$$

$$\dot{\rho} = K_\rho \rho \cos(\alpha) \tag{10}$$

$$\dot{\beta} = K_\rho \sin(\beta) - K_\beta \beta \tag{11}$$

Note que es una ley de control proporcional ya que se requiere el valor del error en cada punto del cálculo y se multiplica este por constantes para determinar el efecto de control sobre estos errores. Entonces la elección de K_α, K_ρ y K_β se realizó por sintonización manual. Se evaluó el tiempo que le toma al robot llegar a la posición $[-2,\ -2,\ -3\pi/4]_F$ partiendo de la posición $[0,\ 0,\ -\pi/4]$. Para seleccionar estas constantes se deben cumplir las siguientes condiciones:

$$(K_\alpha - K_\rho) > 0 \ \mid \ K_\rho > 0 \ \mid \ K_\beta < 0 \tag{12}$$

Cumpliendo las condiciones de la ecuación 12 se probaron diferentes combinaciones de estas constantes y se seleccionó la que permitiera al robot llegar a la posición final deseada en el menor tiempo posible, si la simulación tomaba más de $300s$ se consideraba que estaba fuera del límite esperado y se le asignaba un NTR (No Terminó el Recorrido).

Cuadro I
ELECCIÓN DE CONSTANTES PROPORCIONALES PARA EL CONTROL
NTR: No Terminó el Recorrido

-	1	2	3	4	5	6	7
K_α	0,70	0,60	0,55	0,55	0,50	0,50	0,50
K_ρ	0,65	0,55	0,52	0,47	0,47	0,45	0,40
K_β	-0,70	-0,60	-0,55	-0,55	-0,50	-0,50	-0,50
Tiempo[s]	NTR	NTR	192,2	NTR	128,8	176,3	149,8

Con los resultados del Cuadro I se determinó que las constantes a usar en el modelo serían:

$$K_\alpha = 0,50$$
$$K_\rho = 0,47$$
$$K_\beta = -0,50$$

Comunicación del nodo

El nodo de control recibe la posición actual del robot a través del tópico **/turtlebot_position** y utiliza esto datos para estimar las velocidades que se deben aplicar, estas son enviadas a través del tópico **/turtlebot_cmdVel** para que se actualice el modelo de simulación. Estas interacciones se pueden ver en la Figura 10.

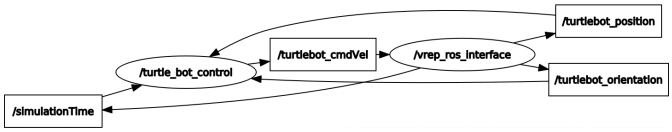

Figura 10. Grafo de comunicación de /turtle_bot_control

Estimación de posición

Además de esto se estima la posición del robot a partir del comando de velocidad especificado por el control realizando Integración Numérica. Este resultado puede diferir de la posición real ya que la estimación de la posición por este método no tiene en cuenta efectos como: la aceleración de los motores, fricción con el suelo, deslizamiento de las ruedas, inestabilidades en el cuerpo del robot, entre otros. Por esto se puede ver la diferencia entre la trayectoria real del robot y la trayectoria estimada por el controlador en la Figura 11.

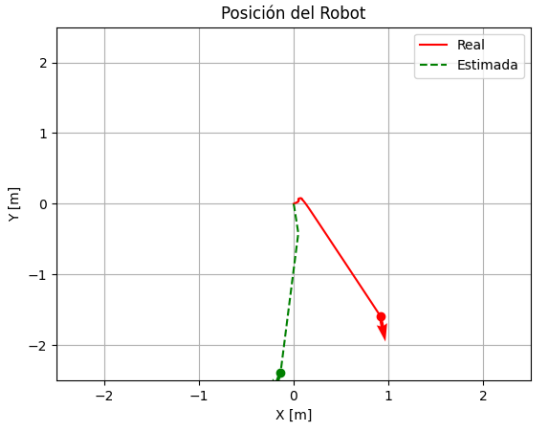

Figura 11. Trayectoria real y estimada del robot para llegar a $[-1, -1{,}5, -\pi/2]_F$

Errores de posición y orientación durante el recorrido

Al finalizar el recorrido del robot se muestra una gráfica con el error de posición y el error de orientación a lo largo del tiempo durante el recorrido, esta gráfica se muestra en la Figura 12. En esta gráfica se puede ver como primero se reduce el error de orientación (Etapa 1), luego al rededor de $t = 2s$ se empieza a corregir la distancia hasta P_F (Etapa 2) y finalmente en $t = 7,6s$ se corrigió la orientación final (Etapa 3).

III. INSTRUCCIONES DE EJECUCIÓN

Para comprobar el funcionamiento de los nodos creados es necesario que el dispositivo que ejecuta la simulación disponga de herramientas de terceros que han sido empleadas en el código de esta solución. Por esto, en la sección III-A se explicará cómo instalar todas las librerías adicionales requeridas para el funcionamiento de los nodos en ROS. Posterior a esto, en

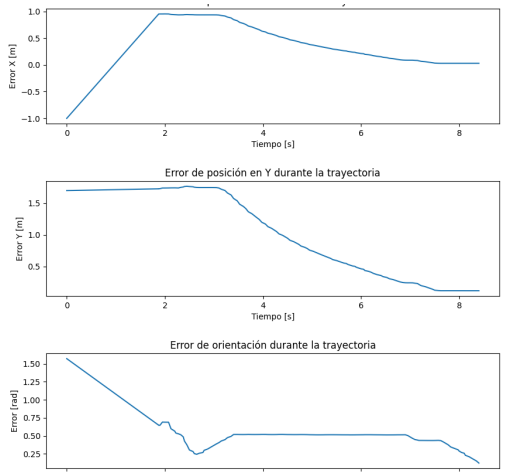

Figura 12. Error de posición y orientación para pasar de $[-1,-1{,}5,-\pi/2]$ a $[-1,-1{,}7,-3\pi/4]_F$

la sección III-B, se explicará cómo ejecutar la simulación y comprobar el funcionamiento de los nodos desarrollados.

Es importante que maneja la versión de Python3 en el equipo dónde se va a correr la simulación, y que tenga instalado el manejador de paquetes de Python3 'pip3' para la instalación de las siguientes librerías. Si está utilizando la máquina virtual del curso de robótica ya contará con esta versión de Python. Si no lo tiene, pude obtenerlo con el siguiente comando:

```
$ sudo apt install python3-pip
```

III-A. Herramientas adicionales

Librería *matplotlib* [1]

Esta librería se utiliza para graficar datos en Python, y ha sido utilizada en el nodo **/turtle_bot_position** para representar en tiempo real la posición del robot. Adicionalmente se uso la librería PyQT5 que permite a *matplotlib* ser usada de manera concurrente. Para instalar esta librería utilice los siguientes comandos en una terminal:

```
$ pip3 install matplotlib
$ pip3 install pyqt5
```

Librería *numpy* [2]

Esta librería se utiliza para calcular operaciones trigonométricas y realizar operaciones entre matrices de manera eficiente. Ha sido utilizada en el nodo **/turtle_bot_control** para estimar la posición actual del robot. Para instalar esta librería utilice los siguientes comandos en una terminal:

```
$ pip3 install numpy
```

III-B. Prueba de funcionamiento de los nodos

Para comprobar el funcionamiento de los nodos es necesario correr primero la simulación de V-REP para el Turtlebot2, para esto abra una terminal y corra el *roscore* de ROS:

```
$ roscore
```

En otra terminal diríjase a la carpeta en donde se encuentra el simulador V-REP y ejecute el script *vrep.sh*. Si está utilizando la máquina virtual del curso de robótica será:

```
$ cd Documents/Programas/
$ cd V-REP_PRO_EDU_V3_6_2_Ubuntu18_04/
$ ./vrep.sh
```

Cuando V-REP se haya ejecutado abra la escena de la simulación en el menú File→Open. Ejecute la simulación para comprobar el funcionamiento de los nodos.

Ahora descomprima el archivo *taller2_grupo4.tar.gz* y compile el paquete ROS utilizando los siguientes comandos en su entorno de trabajo para ROS, suponiendo que es *catkin_ws*:

```
$ tar -xzvf taller2_grupo4.tar.gz
$ cd catkin_ws/
$ source devel/setup.bash
```

Nodo /MovementNode

Para ejecutar este nodo abra una terminal y corra el nodo con dicho nombre perteneciente al paquete de ROS **taller2_4**:

```
$ rosrun taller2_4 Punto2.py <nombreArchivo>.txt
```

Nodo /turtle_bot_control

Para ejecutar este nodo abra una terminal y corra el nodo con dicho nombre perteneciente al paquete de ROS **taller2_4**:

```
$ rosrun taller2_4 turtle_bot_control.py
```

Para que la gráfica se guarde de manera correcta, es necesario detener la simulación en V-REP. Después se puede hacer una interrupción por teclado.

REFERENCIAS

[1] J. D. Hunter and M. Droettboom, *Librería matplotlib para Python*, 2010. Disponible en https://pypi.org/project/matplotlib/ - Versión: 3.3.1.

[2] R. Mers, *Librería numpy para Python*, 2016. Disponible en https://pypi.org/project/numpy/ - Versión: 1.19.2.