

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO
Departamento Académico de Sistemas Digitales

Temas Selectos de Robótica: SDI-31911
Proyecto 1: Generación de trayectorias en ROS

Materia:
Robótica: SDI-11911

Profesor:
Dr. Marco Morales

Presenta:
Job Magdiel González González

Otoño 2016

Índice

1. Introducción	1
2. Objetivo	1
3. Marco Teórico	1
3.1. ¿Qué es ROS?	1
3.2. Metas	1
3.3. Sistemas operativos	1
3.4. Conceptos	2
3.4.1. Nivel de Sistema de archivos	2
3.4.2. Nivel de Cómputo Gráfico	2
3.5. Trayectoria	4
4. Desarrollo	5
4.1. Instalación	5
4.2. Observaciones para la realización	5
4.3. Implementación y pruebas	7
4.4. Definición del experimento	8
4.5. Simulaciones	8
4.6. Resultados	10
5. Conclusiones	13

1. Introducción

En los últimos años, la comunidad robótica ha alcanzado grandes progresos en la creación de robots de bajo costo. Al mismo tiempo, se ha desarrollado algoritmos que ayudan a estos robots que funcionan con cierto nivel de autonomía. A pesar de este rápido progreso, los robots presentan retos muy interesantes e importantes.

En el presente proyecto se presenta un sistema operativo del robot que está dirigido a presentar posibles soluciones a algunas dificultades que se presentan durante el desarrollo de software. Entendiendo el funcionamiento de Turtlesim (paquete de ROS) y la estructura de los tópicos de publicación y suscripción se presenta una posible solución a un problema propuesto.

2. Objetivo

Aprender los fundamentos de ROS y aplicarlos para construir nodos con capacidad de publicar y suscribirse a tópicos.

3. Marco Teórico

3.1. ¿Qué es ROS?

ROS (Robot Operating System) provee librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes y más. ROS está bajo la licencia open source, BSD.¹

3.2. Metas

El objetivo principal de ROS es apoyar la reutilización de código en la investigación y desarrollo en la robótica. ROS es un marco distribuido de procesos (nodos) que permite diseñar ejecutables de forma individual y en parejas sueltas en tiempo de ejecución. Estos procesos pueden agruparse en paquetes y pilas que pueden ser fácilmente compartidos y distribuidos. También es compatible con los repositorios que permiten la colaboración a nivel de la comunidad.

3.3. Sistemas operativos

ROS actualmente sólo se ejecuta en plataformas basadas en UNIX. Se prueba principalmente en sistemas Ubuntu y Mac OS X. Mientras que un puerto para Microsoft Windows es posible todavía no se explorado completamente.

¹<http://wiki.ros.org/es>

3.4. Conceptos

3.4.1. Nivel de Sistema de archivos

Los conceptos de nivel de sistema de archivos se refieren principalmente a los recursos de ROS que se encuentra en el disco.²

- ✓ Paquetes: Los paquetes son la unidad principal para la organización de software en ROS. Un paquete puede contener procesos (nodos), una biblioteca, bases de datos, archivos de configuración, o cualquier otra cosa que se organiza de manera útil en conjunto. Los paquetes son el elemento de construcción y de lanzamiento más atómico en ROS. Esto significa que lo más granular que se puede construir y liberar es un paquete.
- ✓ Metapaquetes: Son paquetes especializados que sólo sirven para representar un grupo de otros paquetes relacionados. Comúnmente se utilizan como un marcador de posición compatible con versiones anteriores.
- ✓ Manifiestos: (package.xml) Proporcionan metadatos acerca de un paquete, incluyendo su nombre, versión, descripción, información de la licencia, dependencias y otra información de metadatos como paquetes exportados.
- ✓ Repositorios: Una colección de paquetes que comparten un sistema VCS en común. Los repositorios también pueden contener un solo paquete.
- ✓ Tipos (NVI): Definen las estructuras de datos para los mensajes enviados en ROS.
- ✓ Tipos de servicio: Definen las estructuras de datos de petición y respuesta de los servicios de ROS.

3.4.2. Nivel de Cómputo Gráfico

La Computación Gráfica es la red de procesos de ROS que se están procesando datos en conjunto peer-to-peer. (ver Figura 1).

- ✓ Nodos: Los nodos son procesos que realizan el cálculo. ROS está diseñado para ser modular en una escala de grano fino. Un sistema de control de robot comprende normalmente muchos nodos.
- ✓ Maestro: El ROS Maestro proporciona el registro de nombres y de consulta para el resto de la Computación Gráfica. Sin el Maestro, los nodos no serían capaces de encontrar cada uno de los mensajes de otro o invocar servicios.
- ✓ Servidor de parámetros: El servidor de parámetros permite que los datos sean almacenados por la clave en una ubicación central. En la actualidad es parte del maestro.
- ✓ Mensajes: Los nodos se comunican entre sí mediante el paso de mensajes. Un mensaje es simplemente una estructura de datos que comprende tipos primitivos estándar (enteros, punto flotante, booleanos, etc.) así como arreglos de tipos primitivos. Los mensajes pueden incluir estructuras y arreglos (tanto como estructuras) C arbitrariamente anidados.
- ✓ Temas: Los mensajes se enrutan a través de un sistema de transporte con publicación / suscripción semántica. Un nodo envía un mensaje por la publicación a un determinado tema. El tema es un nombre que se utiliza para identificar el contenido del mensaje. Un

²<http://wiki.ros.org/ROS/Concepts>

nodo que está interesado en un determinado tipo de datos va a suscribirse al artículo apropiado. Puede haber varios editores y suscriptores concurrentes para un solo tema, y un único nodo puede publicar y / o suscribirse a varios temas. En general, los editores y suscriptores no son conscientes de la existencia de los demás. (ver Figura 2).

- ✓ Servicios: El modelo de publicación / suscripción es un modelo de comunicación muy flexible, pero su transporte de muchos a muchos unidireccional no es apropiado para interacciones de petición / respuesta que a menudo se requieren en un sistema distribuido. La solicitud / respuesta se realiza a través de los servicios que se definen por un par de estructuras de mensaje: uno para la solicitud y otro para la respuesta. Un nodo que ofrece un servicio bajo un nombre y un cliente utiliza el servicio mediante el envío del mensaje de solicitud y espera la respuesta.
- ✓ Bolsas: Las bolsas son un formato para guardar y reproducir datos de mensaje ROS. Las bolsas son un mecanismo importante para el almacenamiento de datos, tales como datos de los sensores, que pueden ser difíciles de recoger pero son necesarios para desarrollar y probar algoritmos.

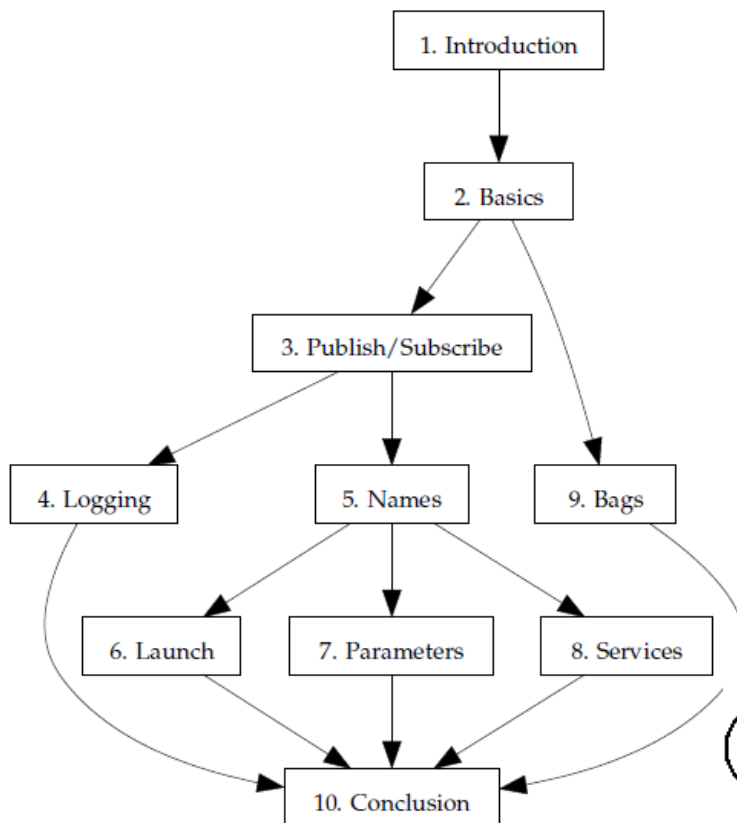


Figura 1: Ejemplo de dependencias

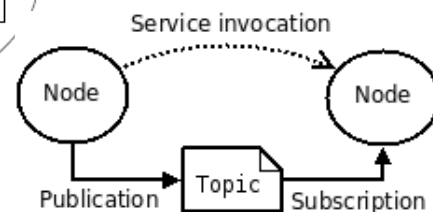


Figura 2: Concepto básico

3.5. Trayectoria

Una trayectoria es una secuencia de posiciones en el tiempo.

La velocidad es una magnitud física de carácter vectorial que expresa la distancia recorrida de un objeto por unidad de tiempo. Su unidad en el Sistema Internacional de Unidades es el metro por segundo (símbolo $\frac{m}{s}$).

La velocidad angular es una medida de la velocidad de rotación. Se define como el ángulo girado por una unidad de tiempo y se designa mediante la letra griega ω . Su unidad en el Sistema Internacional es el radián por segundo $\frac{rad}{s}$.

Aunque se la define para el movimiento de rotación del sólido rígido, también se la emplea en la cinemática de la partícula o punto material, especialmente cuando esta se mueve sobre una trayectoria cerrada (circular, elíptica, etc).

Para calcular la velocidad lineal y el ángulo de giro hacia un punto en 2D utilizaremos las siguientes fórmulas:

✓ Velocidad Lineal

$$\text{Velocidad} = \frac{\text{distancia}}{\text{tiempo}} = \frac{\sqrt{(x^*-x)^2 + (y^*-y)^2}}{\text{tiempo}}$$

✓ Ángulo de giro

$$\theta = \tan^{-1}\left(\frac{y^*-y}{x^*-x}\right)$$

Sea:

- x^* Posición X objetivo
- y^* Posición Y objetivo
- x Posición X actual
- y Posición Y actual

4. Desarrollo

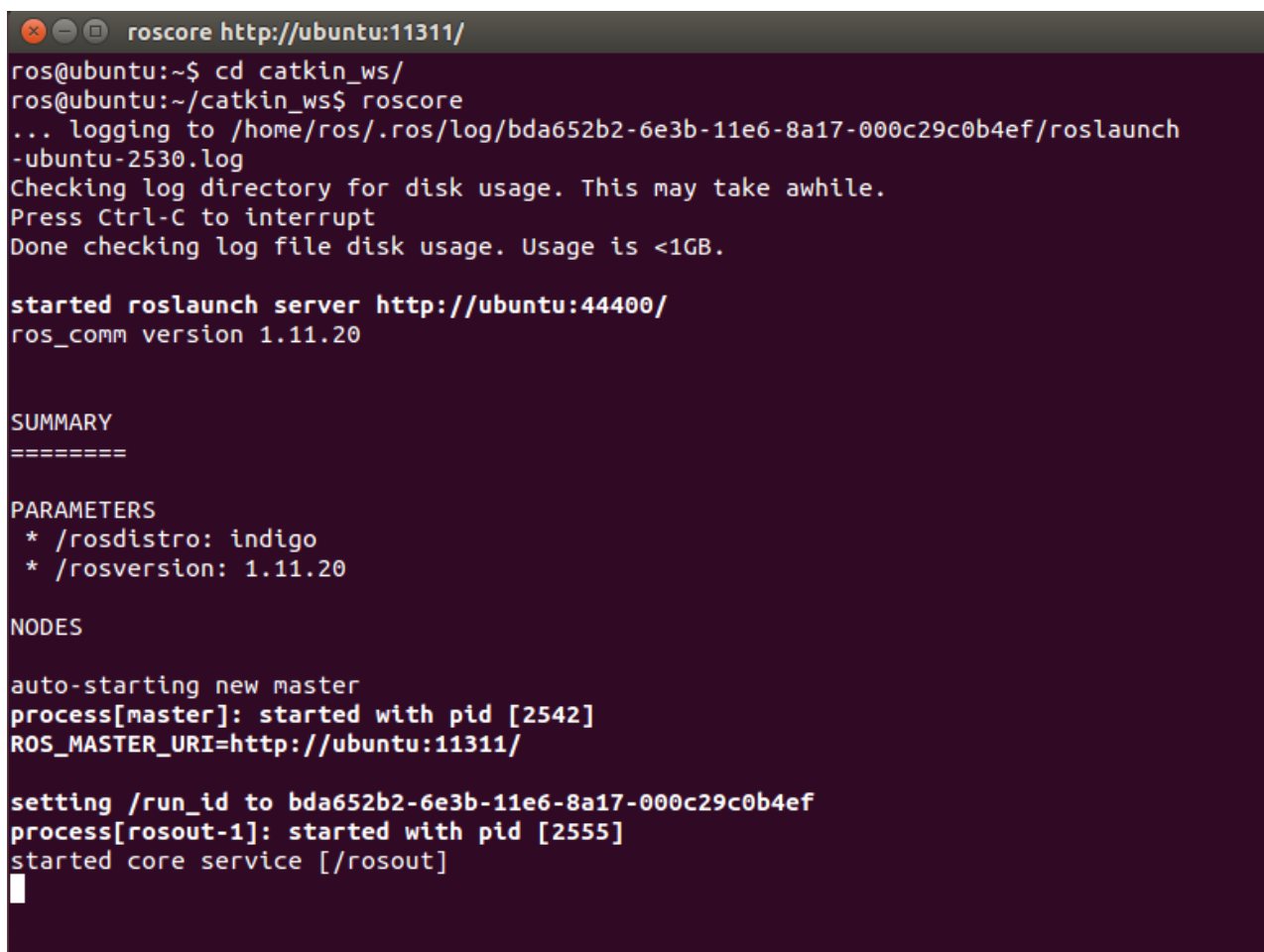
4.1. Instalación

Para la instalación de ROS, ver la documentación de ROS.³

4.2. Observaciones para la realización

Para la comprobación de la instalación, los siguientes comandos fueron ejecutados individualmente en una terminal por separado. Las terminales independientes nos permite ejecutar los tres comandos de forma simultánea.

1. **roscore**. El comando “roscore” inicia el ROS Maestro, el servidor de parámetros y un nodo de registro “rosout” (ver Figura 3).



```
roscore http://ubuntu:11311/
ros@ubuntu:~$ cd catkin_ws/
ros@ubuntu:~/catkin_ws$ roscore
... logging to /home/ros/.ros/log/bda652b2-6e3b-11e6-8a17-000c29c0b4ef/roslaunch
-ubuntu-2530.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:44400/
ros_comm version 1.11.20

SUMMARY
=====

PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.20

NODES

auto-starting new master
process[master]: started with pid [2542]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to bda652b2-6e3b-11e6-8a17-000c29c0b4ef
process[rosout-1]: started with pid [2555]
started core service [/rosout]
```

Figura 3: comando “roscore”

³<http://wiki.ros.org/indigo/Installation/Ubuntu>

2. **roslaunch turtlesim turtlesim.launch**. El comando “roslaunch turtlesim turtlesim.launch” abre una ventana gráfica similar al de la Figura 4. Esta ventana muestra un robot simulado en forma de tortuga que vive en un mundo cuadrado. (La apariencia de la tortuga puede ser diferente).

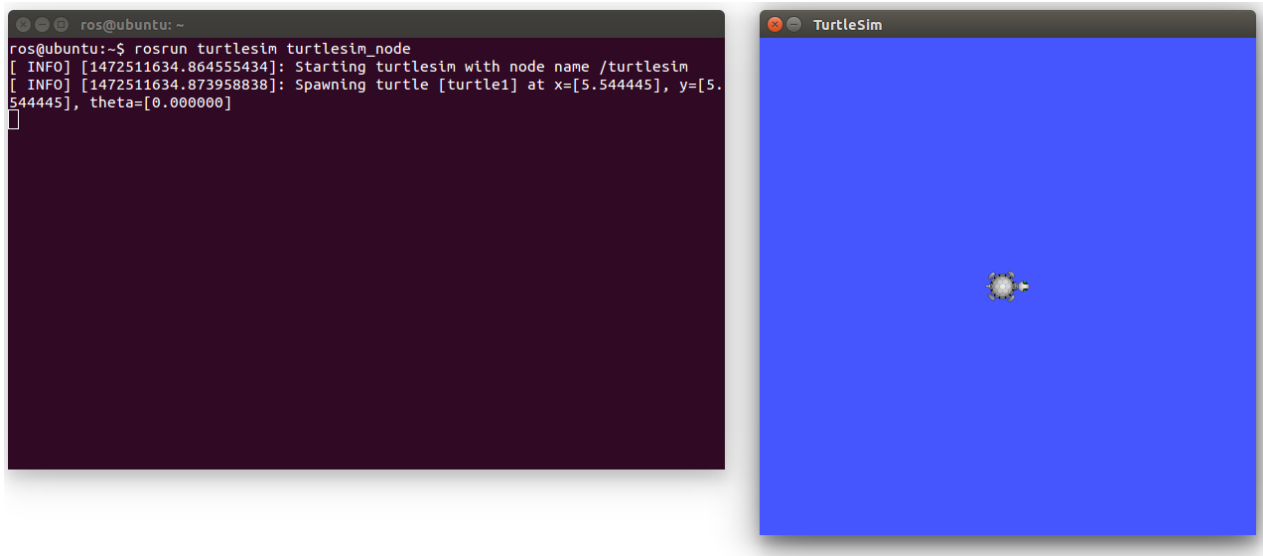


Figura 4: comando “roslaunch turtlesim turtlesim.launch”

3. **roslaunch turtlesim turtle_teleop_key**. El comando “roslaunch turtlesim turtle_teleop_key” nos permite mover la tortuga al pulsar en el teclado las flechas de dirección (arriba, abajo, izquierda, derecha). La tortuga deja un rastro detrás de él en respuesta a las teclas pulsadas (ver Figura 5).

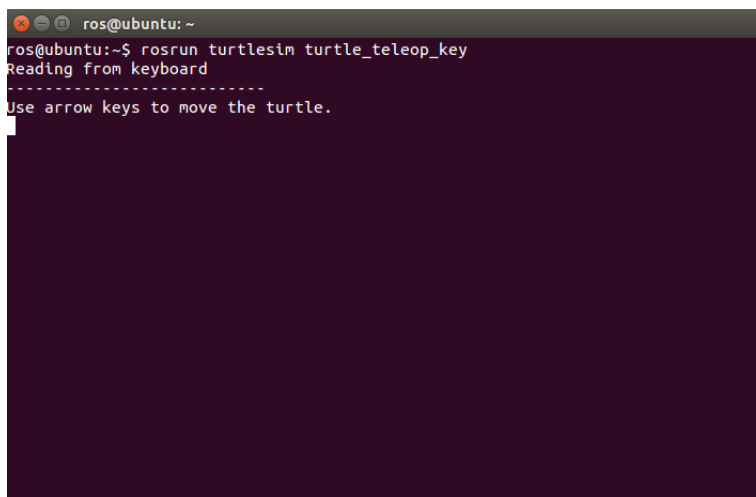


Figura 5: comando “roslaunch turtlesim turtle_teleop_key”

Debe mantenerse abiertas estas tres terminales para poder interactuar con este sistema (ver Figura 6).

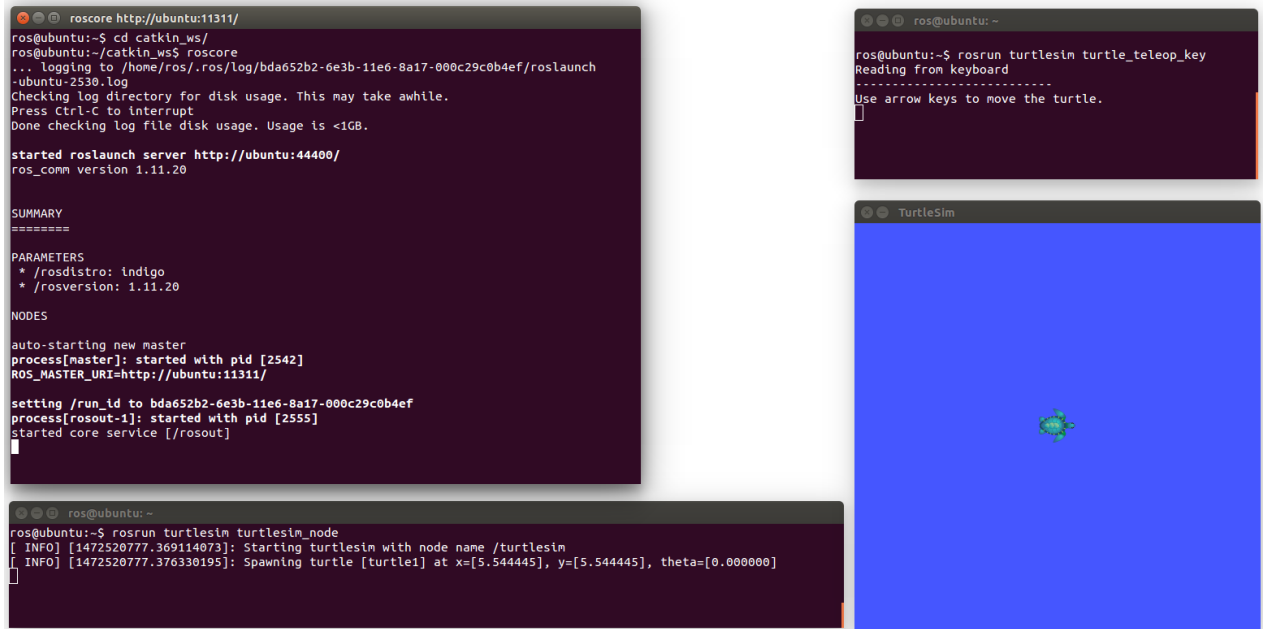


Figura 6: Vista de las tres terminales ejecutándose simultáneamente

4.3. Implementación y pruebas

Se realizaron las configuraciones necesarias en el “manifiesto” (package.xml) donde fueron declaradas las librerías de clientes ROS que son llamadas al comenzar un programa.

Se declararon las dependencias en el archivo CMakeLists.txt de nuestro directorio. una vez declaradas las dependencias se declararon los programas ejecutables.

Posteriormente, construimos nuestra área de trabajo para finalmente ejecutar el script llamado “setup.bash” que establece las variables disponibles en ROS para nuestro ambiente de trabajo.

En esta sección se implementaron los siguientes programas:

1. Un simple programa “Hello”.
2. Un programa de publicación de mensajes. En esta parte realizaron las pruebas para publicar las velocidades lineares y angulares de la tortuga (Topic: geometry_msgs/Twist).
3. Un programa de suscripción. En esta parte se suscribieron las posiciones en x, y, z junto con su ángulo theta (θ). (Topic: turtlesim/Pose)).

4.4. Definición del experimento

Se escribió un programa para generar trayectorias para las tortugas de Turtlesim. El programa tiene las siguientes características:

1. Se ejecuta desde la terminal
2. Pide al usuario la posición final y el tiempo de ejecución deseados.
3. Lee la posición actual de la tortuga que publica Turtlesim.
4. Envía los mensajes de velocidad necesarios a una frecuencia de 10Hz para que la tortuga siga una trayectoria de la posición actual a posición final en un tiempo de ejecución. Durante la ejecución, se mantiene actualizada la posición actual con los datos publicados por Turtlesim.
5. Regresar al paso 2 en forma iterativa.

4.5. Simulaciones

Se considera que la ventana en la que vive la tortuga es una cuadrícula de 11x11 aproximadamente. La tortuga se ubica en la posición $X = 5.544445$, $Y = 5.544445$ con $\theta = 0$.

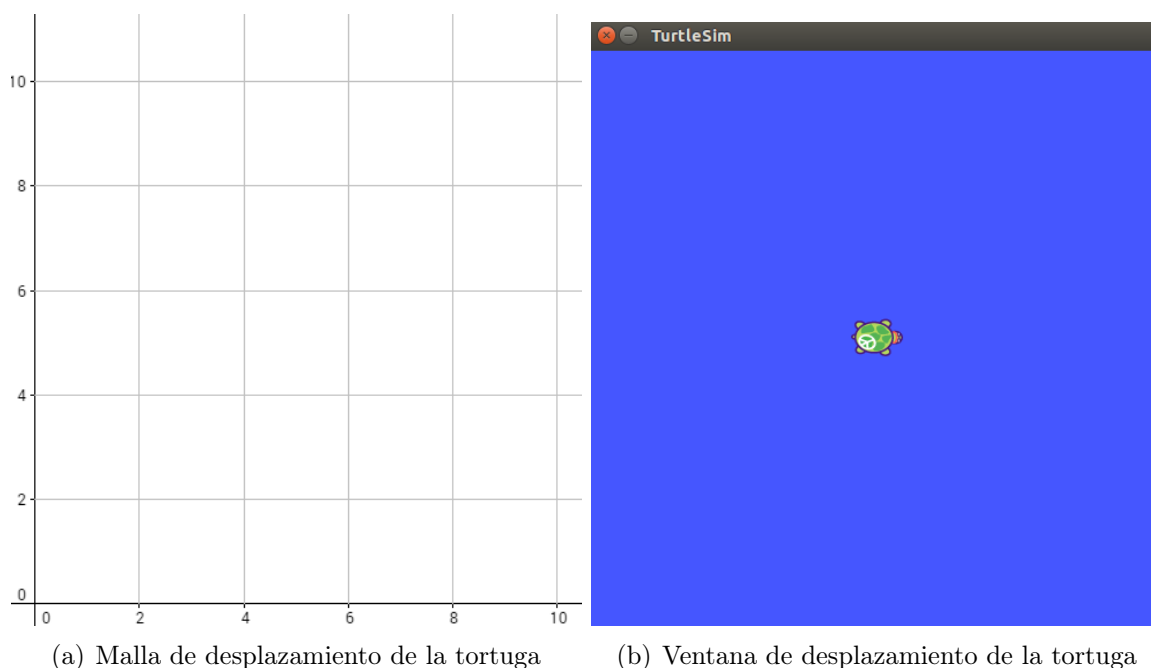
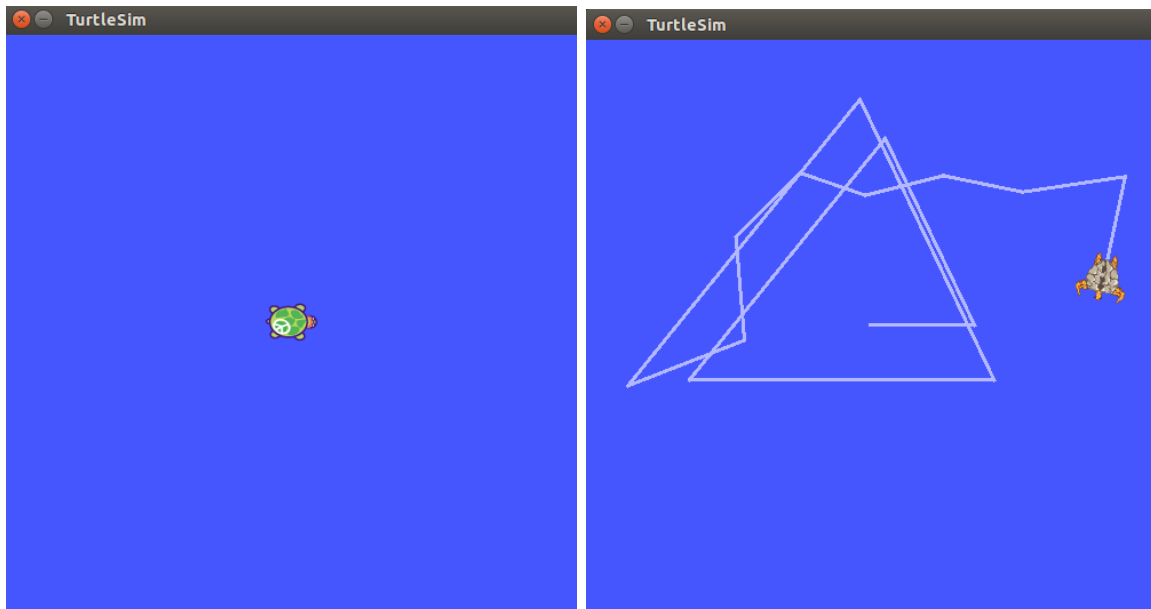
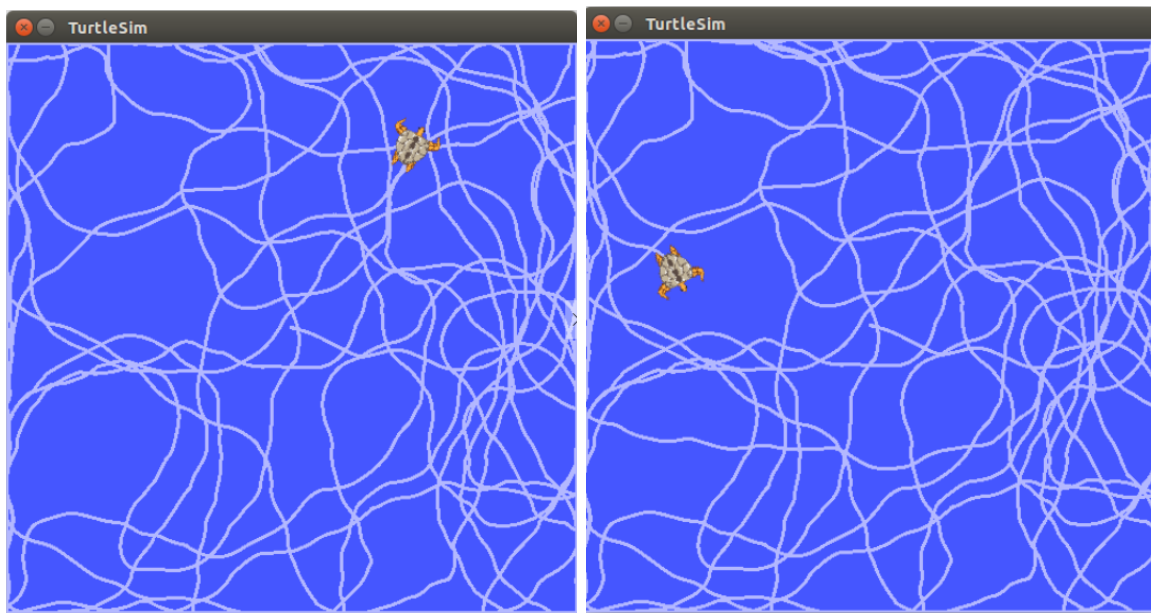


Figura 7: Turtlesim



(a) Inicio de la prueba

(b) Prueba con teclas de dirección



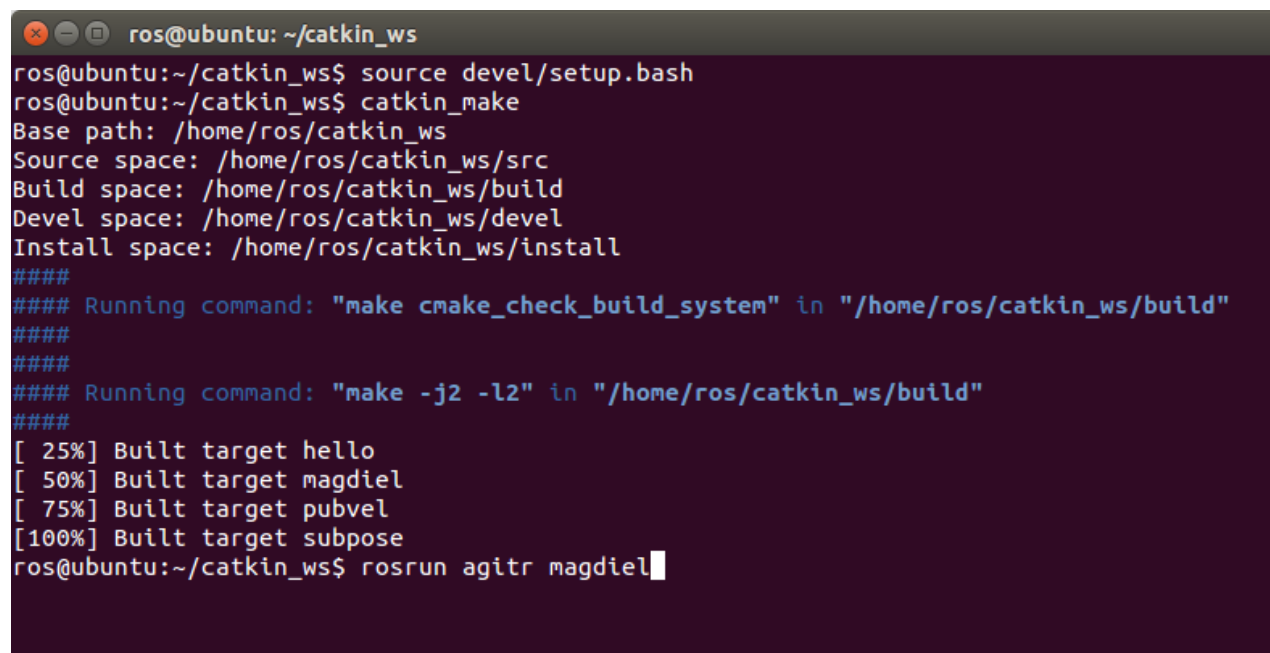
(c) 20 minutos en simulación con velocidades aleatorias

(d) 30 minutos en simulación con velocidades aleatorias

Figura 8: Turtlesim

4.6. Resultados

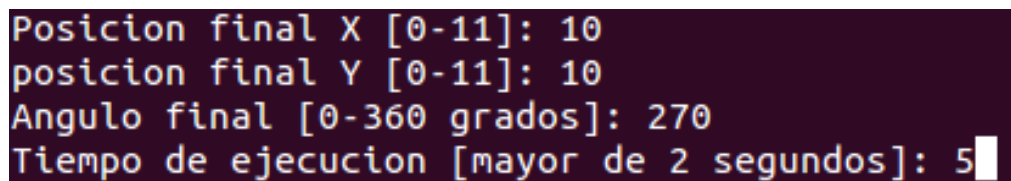
1. Se ejecuta desde la terminal El programa se ejecuta en una terminal independiente con la instrucción “roslaunch agitr magdiel” donde “agitr” es el paquete (package) y “magdiel” es el nombre del programa (ver Figura 9). En otra terminal se ejecuta el comando “roscore” y en otra terminal el comando “roslaunch turtlesim turtlesim_node”.



```
ros@ubuntu: ~/catkin_ws
ros@ubuntu:~/catkin_ws$ source devel/setup.bash
ros@ubuntu:~/catkin_ws$ catkin_make
Base path: /home/ros/catkin_ws
Source space: /home/ros/catkin_ws/src
Build space: /home/ros/catkin_ws/build
Devel space: /home/ros/catkin_ws/devel
Install space: /home/ros/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/ros/catkin_ws/build"
####
####
#### Running command: "make -j2 -l2" in "/home/ros/catkin_ws/build"
####
[ 25%] Built target hello
[ 50%] Built target magdiel
[ 75%] Built target pubvel
[100%] Built target subpose
ros@ubuntu:~/catkin_ws$ roslaunch agitr magdiel
```

Figura 9: Ejecución del programa en terminal.

2. Pide al usuario la posición final y el tiempo de ejecución deseados. El programa pide una posición final en “x” y en “y” en un rango de [0, 11] cada uno. Solicita el ángulo final deseado. Pide el tiempo de ejecución deseado en segundos (ver Figura 10).



```
Posicion final X [0-11]: 10
posicion final Y [0-11]: 10
Angulo final [0-360 grados]: 270
Tiempo de ejecucion [mayor de 2 segundos]: 5
```

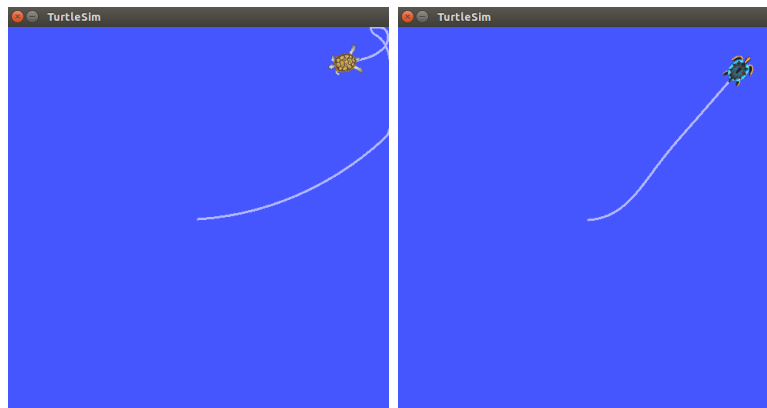
Figura 10: El programa pide al usuario la posición final, el tiempo de ejecución y el ángulo final deseado.

3. Lee la posición actual de la tortuga que publica Turtlesim. En la Figura 11 se observa la parte del código en el cual se lee la posición actual de la tortuga.

```
/**
 * Durante la ejecucion, se mantiene actualizada
 * la posiscion actual con los datos publicados por Turtlesim
 */
void posActualiza(const turtlesim::Pose::ConstPtr & pose_message){
    turtlesim_pose.x=pose_message->x;
    turtlesim_pose.y=pose_message->y;
    turtlesim_pose.theta=pose_message->theta;
}
```

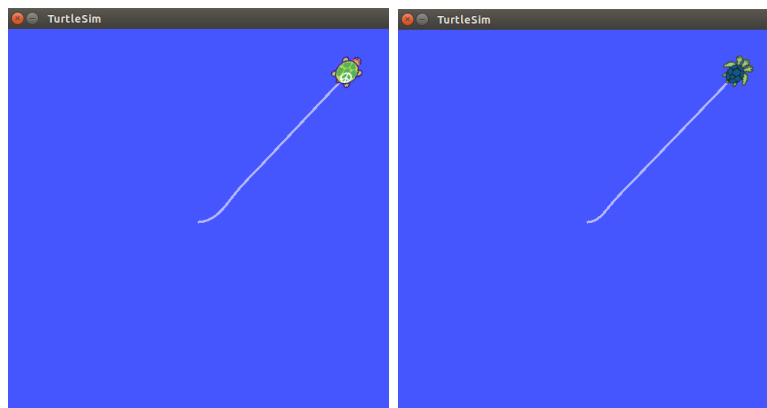
Figura 11: El programa lee la posición actual de la tortuga.

4. Envía los mensajes de velocidad necesarios a una frecuencia de 10Hz para que la tortuga siga una trayectoria recta de la posición actual a posición final en un tiempo de ejecución. Durante la ejecución, se mantiene actualizada la posición actual con los datos publicados por Turtlesim. En este punto, el programa gira para alinearse con el punto objetivo mientras que al mismo tiempo avanza.



(a) 1 segundo

(b) 5 segundos



(c) 10 segundos

(d) 15 segundos

Figura 12: Avance de (5.5444,5.5444) a (10,10)

Durante la simulación encontramos que hay un tiempo de retardo cuando se avanza a un punto objetivo. Se realizó la búsqueda experimental de un factor que considerara dicho retardo. Este factor se multiplica por el tiempo de ejecución deseado. Los resultados se muestran en las Tablas 1 y 2.

Tiempo	Factor			
	4.0		4.2	
	(0,0) a (11,11)	(11,11) a (0,0)	(0,0) a (11,11)	(11,11) a (0,0)
1 seg	0.9014	1.8007	0.9008	1.4011
5 seg	5.3003	5.1007	5.1008	4.9015
10 seg	10.8011	10.6010	10.0008	10.3010
15 seg	16.2017	16.0009	15.2017	15.5014
20 seg	21.7013	21.4021	20.4014	20.6014

Tabla 1: Tabla de factores 4.0 y 4.2

Tiempo	Factor			
	4.25		4.5	
	(0,0) a (11,11)	(11,11) a (0,0)	(0,0) a (11,11)	(11,11) a (0,0)
1 seg	0.9009	1.7019	0.9018	1.5013
5 seg	5.0023	4.8145	4.7010	4.5015
10 seg	10.1012	9.9014	9.6010	9.3008
15 seg	15.3014	15.0011	14.4008	14.2013
20 seg	20.4018	20.2020	19.3017	19.0011

Tabla 2: Tabla de factores 4.25 y 4.5

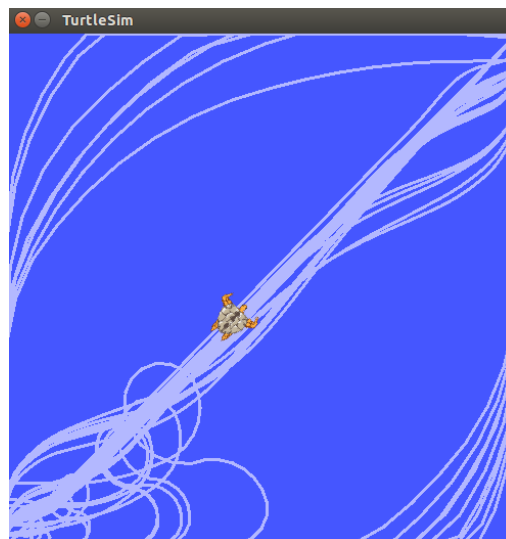
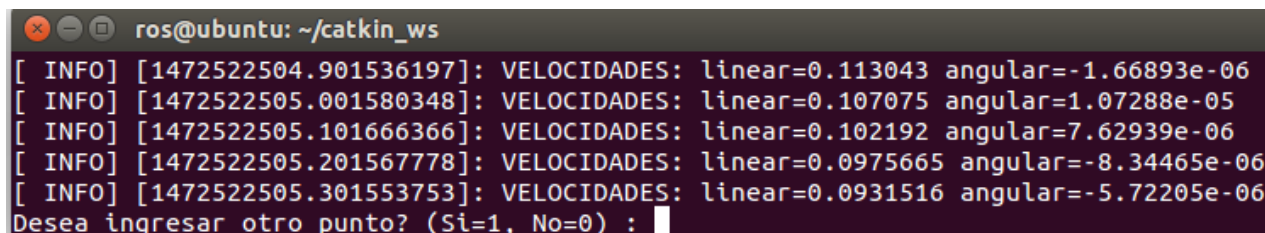


Figura 13: Ventana de simulación.

5. Regresar al paso 2 en forma iterativa. Al finalizar el tiempo de desplazamiento de la tortuga, el programa pregunta si se desea ingresar otro punto. En caso de seleccionar la opción “Sí=1” se pide al usuario la nueva posición final y el tiempo de ejecución deseados. En caso de seleccionar la opción “No=0” finaliza el programa.



```

ros@ubuntu: ~/catkin_ws
[ INFO] [1472522504.901536197]: VELOCIDADES: linear=0.113043 angular=-1.66893e-06
[ INFO] [1472522505.001580348]: VELOCIDADES: linear=0.107075 angular=1.07288e-05
[ INFO] [1472522505.101666366]: VELOCIDADES: linear=0.102192 angular=7.62939e-06
[ INFO] [1472522505.201567778]: VELOCIDADES: linear=0.0975665 angular=-8.34465e-06
[ INFO] [1472522505.301553753]: VELOCIDADES: linear=0.0931516 angular=-5.72205e-06
Desea ingresar otro punto? (Si=1, No=0) : 

```

Figura 14: Regresa al paso 2 de forma iterativa

5. Conclusiones

Con la ayuda de ROS aprendimos el concepto de trayectorias y lo pusimos en práctica. Fue un reto buscar suavizar el movimiento de la tortuga para alcanzar el objetivo. Durante la etapa de simulaciones, encontramos un factor de 4.25 que permitía llegar a la posición objetivo en el tiempo de ejecución deseado. Se agregó el factor en la velocidad lineal de la siguiente manera:

$$\text{Velocidad} = \frac{\text{distancia}}{\text{tiempo}} = \frac{\sqrt{(x^*-x)^2+(y^*-y)^2}}{\text{tiempo}*\text{factor}}$$

Las pruebas fueron satisfactorias, sin embargo, cabe mencionar que la tortuga inicia una nueva trayectoria a partir del último punto dado considerando su ángulo actual. Esta variación es de gran influencia en alcanzar el tiempo de ejecución deseado. Las simulaciones se realizaron a lo largo de toda la pantalla, es decir, que funciona bien para distancias largas (mayores a 5 unidades). Para distancias cortas o tiempos de ejecución pequeños, Se obtienen respuestas cercanas a los resultados esperados.

Como mejora del proyecto, cambiaríamos el factor que multiplica el tiempo de ejecución deseado pues lo que cambia constantemente es la distancia y el tiempo va disminuyendo. Si logramos disminuir el tiempo a la misma tasa que la distancia, el factor sería mucho menor pues se necesita para ajustar el tiempo que tarda alinearse con la posición objetivo.

NOTAS DE CLASE Después del tema “Trayectorias” visto en clase, se configuró el código de rotación y de movimiento. El factor experimental calculado en las simulaciones fue sustituido por el tiempo deseable menos el tiempo transcurrido.

$$\text{Velocidad} = \frac{\text{distancia}}{\text{tiempo}} = \frac{\sqrt{(x^*-x)^2+(y^*-y)^2}}{\text{tiempo}-(t_1-t_0)}$$

El tiempo minimo requerido es de 2 segundos pues el tiempo en que tarda en posicionarse en el ángulo final deseado es de 1 segundo. Este segundo se resta al tiempo deseado, lo cual ajusta la velocidad lineal para completar su recorrido y realizar el giro.

Ahora sin el factor experimental, la tortuga realiza un pequeño giro para alinearse con el punto objetivo. Llega al punto deseado con una tolerancia de ± 0.1 unidades en distancia y con una tolerancia de ± 0.1 segundos (ver Figura 15).

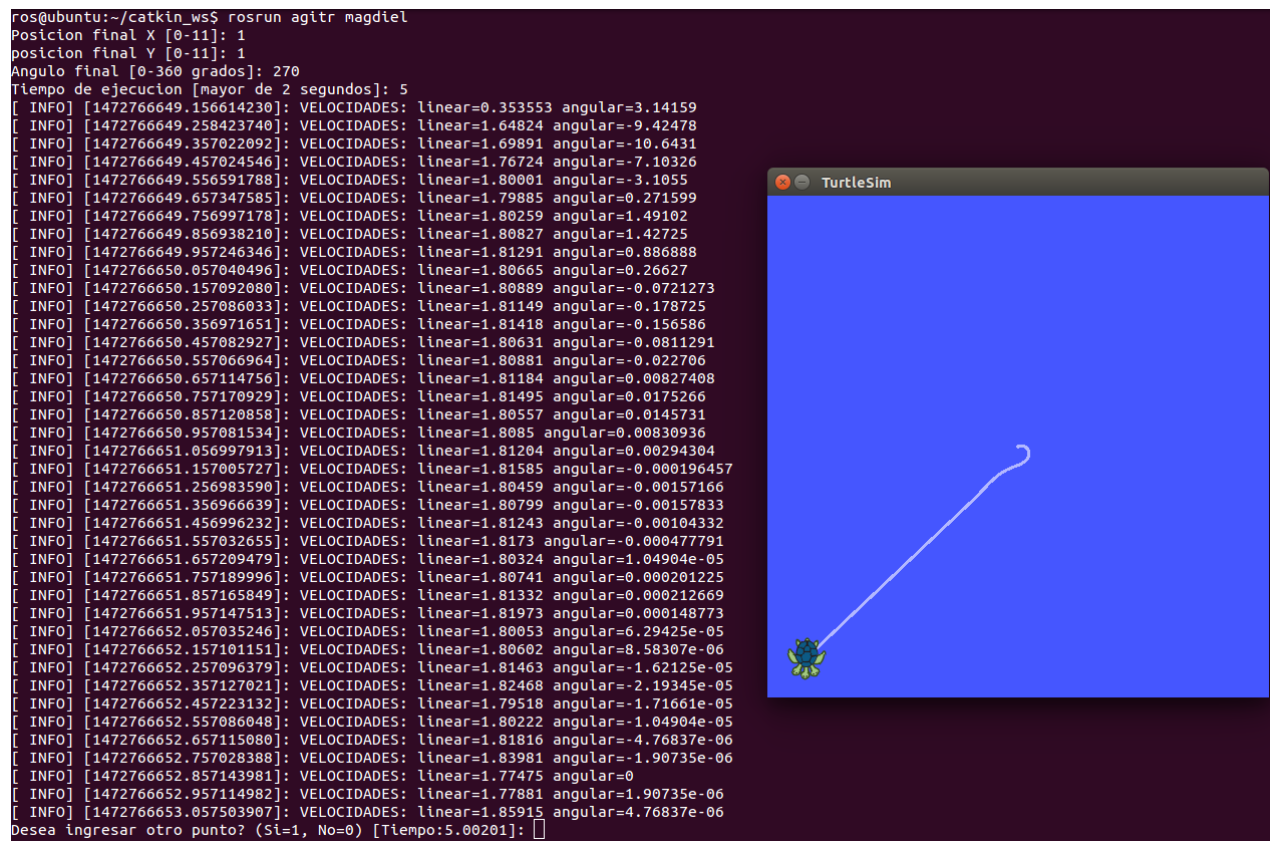


Figura 15: Simulación final.

Referencias

- [1] Jason M. O’Kane. *A Gentle Introduction to ROS*. Independently published, 2013. Available at <http://www.cse.sc.edu/~jokane/agitr/..>
- [2] Serway, Raymond A.; Jewett, John W., *Physics for Scientists and Engineers, 6ta edición*, Brooks/Cole, 2004.