Práctica 3.1

Introducción a la Programación Robótica

Control del movimiento de un robot en Gazebo (I)

Enunciado

En esta práctica revisaremos conceptos tratados anteriormente, como la comunicación a través del modelo **publicador-suscriptor,** pero en este caso dentro de un entorno de simulación 3D como Gazebo.

En particular, el objetivo de la práctica es escribir un **script que monitorice el comportamiento de un robot.** En particular, queremos conseguir que el robot emita simplemente un mensaje del tipo "Estoy cerca de 'x'" cuando se encuentre próximo a una serie de objetos en un escenario 3D dado. Para ello diseñaremos un script que desarrolle toda la lógica de la comunicación en el entorno de simulación para este modelo de comunicación de procesos.

La creación de un modelo de comunicación de tipo publicador-suscriptor va a implicar en este caso la realización de los siguientes pasos:

1. crear un nuevo paquete y la base del script
2. crear un mensaje que comunique la posición del robot
3. implementar un monitor de posiciones del resto de objetos

Para empezar, lanza el entorno de simulación **Gazebo** junto con la opción de interactuar con el robot mediante el teclado.

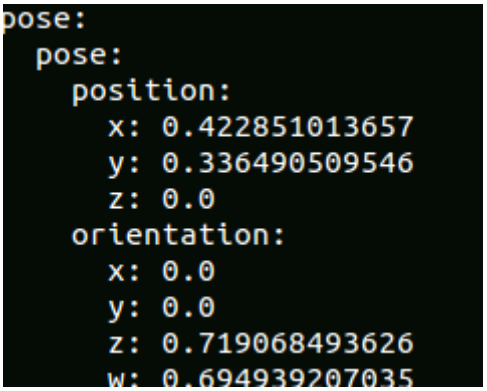
Antes de proceder con el desarrollo del script, conviene saber cómo podemos acceder al robot y a sus datos de posición.

```
# rostopic list
```

Esta información la proporciona el tópico "odom", y en particular, la posición actual se puede obtener con el comando:

```
# rostopic echo /odom
```

Se puede apreciar en la salida como este tópico incluye los datos de posición del robot en el parámetro "position" (pose/pose/position). Prueba a mover el robot y comprueba cómo los datos de posición varían.

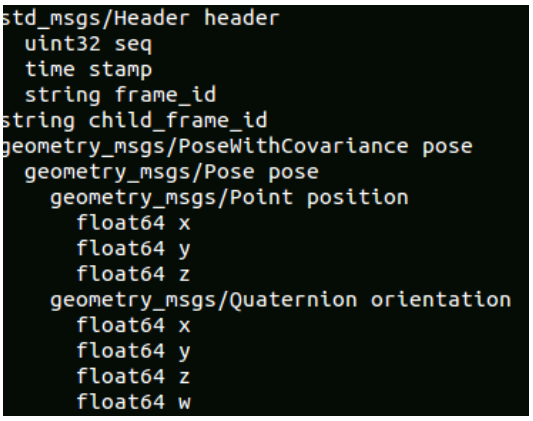


Además, resulta necesario saber qué tipo de mensaje maneja este tópico. Introduce el siguiente comando para averiguarlo:

```
# rostopic info /odom
```

El tipo de mensaje asociado a "odom" es "nav_msgs/Odometry".

```
# rosmsg show nav_msgs/Odometry
```



Como se puede observar, la salida de este comando hace mención en cierto punto a los parámetros de posición y su tipo de datos asociado, aspecto éste que nos va a ser de utilidad en breve.

>> Pasos a realizar:

1) primera versión del script (que muestra la posición del robot)

Crea un paquete con el nombre **location_monitor** dentro del "workspace", incluyendo la referencia a "nav_msgs" (necesario para obtener la posición) y a "rospy". Sitúate en el directorio "location_monitor" que se habrá generado y crea un directorio "scripts".

Dentro de *scripts*, crea un nuevo fichero **location_monitor_node.py** con el siguiente código, cuyo objetivo es publicar por consola la posición del robot.

```
#!/usr/bin/env python

import math
import rospy
import nav_msgs.msg from Odometry

def distance(x1, y1, x2, y2):
        xd = x1 - x2
        yd = y1 - y2
        return math.sqrt(xd*xd, yd*yd)

def callback(msg):
        # Guarda en la variable 'x' la posición de x que se encuentra en 'msg'
        x = msg.pose.pose.position.x
        # Guarda en la variable 'y' la posición de y que se encuentra en 'msg'
        y = msg.pose.pose.position.y
        # Muestra por consola 'x' e 'y'
        rospy.loginfo('x: {}, y: {}'.format(x,y))

def main():
        rospy.init_node('location_monitor')
        rospy.Subscriber("/odom", Odometry, callback)
        rospy.spin()

if __name__ == '__main__':
        main()
```

El contenido de este script te será familiar. Su **objetivo** es crear un nodo (location_monitor) que tomará la posición del robot suscribiéndose al tópico "Odom" y la mostrará por consola.

Fíjate en la clase "Odometry" que se importa al principio. La idea es usar el mensaje que se relaciona con esta clase. El tipo de mensaje asociado a "odom" es "nav_msgs/Odometry", como veíamos anteriormente.

Revisa el resto del contenido del script (en especial, la parte del "main") y asegúrate de que lo entiendes.

Realiza finalmente las siguientes acciones:

1. completa las 3 líneas siguiendo la indicación que se hace en el comentario.
2. ejecuta el script en una nueva consola para comprobar su funcionamiento.
3. muestra en una consola adicional los datos (posición) que está publicando el tópico "odom".

2) crear un nuevo tipo de mensaje (que comunique la distancia de un objeto)

En el código previo hemos conseguido que nuestro script muestre la posición del robot por consola. Sin embargo, esa posición aún no tiene un tópico ni un mensaje asociado mediante la cual pueda ser publicada (es decir, como un nodo que publica su posición mediante un tópico y no solo simplemente la toma del sistema, como hemos visto hace unos instantes), aspecto éste que desarrollaremos en breve.

En este punto, vamos a generar un mensaje específico, al que llamaremos "Landmarkdistance.msg", relacionado con la distancia del robot que pueda usarse posteriormente para comprobar si el robot está a menos de 1 metro del objeto más cercano.

Realiza finalmente las siguientes acciones:

1. crea un directorio **msg** dentro del directorio **location_monitor.**
2. crea un fichero con el nombre **Landmarkdistance.msg** y agrega el siguiente contenido:

```
# Name of the landmark
string name
# Distance of the landmark in meters
```

```
float64 distance
```

Como puede observarse, el mensaje guarda tanto el nombre del objeto como los datos de posición.

A continuación, modificaremos los siguientes dos ficheros, necesarios como paso previo a la compilación de código que incorpore, como es este caso, mensajes de nueva creación.

3. modifica ahora el fichero **package.xml** y añade la siguientes líneas en la zona donde existen otras declaraciones similares:

```
...
<build_depend>message_generation</build_depend>
...
<exec_depend>message_runtime</exec_depend>
```

... procediendo de igual forma con el fichero **CMakeLists.txt,** modificando estas secciones:

```
find_package(catkin REQUIRED COMPONENTS
        message_generation
        nav_msgs
        std_msgs
        rospy
)
...
add_message_files(
        FILES
        LandmarkDistance.msg
)
...
generate_messages(
        DEPENDENCIES
)
...
catkin_package(
        CATKIN_DEPENDS
        message_runtime
        nav_msgs
        std_msgs
        rospy)
```

4. finalmente, compila mediante **catkin_make** (recuerda hacerlo desde el directorio "catkin_ws")

3) implementar el monitor de posiciones

Hasta ahora, el script que tenemos es capaz de informar por consola de la posición del robot, pero sin publicarla todavía a través de un tópico creado a tal efecto.

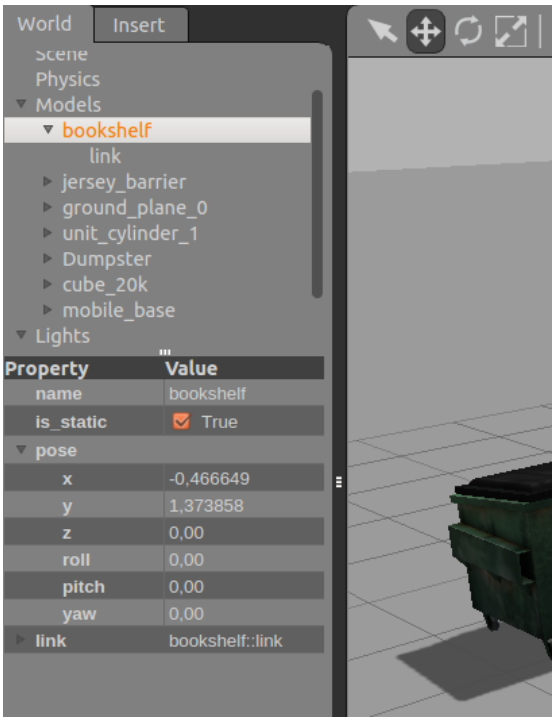
Resta por realizar los siguientes pasos:

a) averiguar cómo obtener la posición del resto de objetos.
b) calcular la distancia entre el robot y el resto de objetos según la posición cambiante del primero en el escenario.
c) publicar como tópico cuál es el objeto más cercano en cada momento.
d) imprimir por consola los casos en los que el objeto más cercano está a menos de 1 metro.

¿Cómo se puede obtener la posición del resto de objetos?

La posición de un objeto puede obtenerse desde el simulador, a través de las propiedades asociadas, cuyos valores se recogen en la parte izquierda de la pantalla.

En la imagen de la izquierda, al estar seleccionado la estantería ("bookshelf"), sus propiedades aparecen en la parte inferior. Entre ellas, se puede observar los valores de sus coordenadas de posición.



Modifica el script para añadir las referencias a los objetos del escenario junto a sus posiciones. Para ello, puedes crear una lista, a la que puedes ir agregando los diferentes objetos con sus posiciones de la siguiente forma:

```
def main():
        landmarks = []
        landmarks.append(('bookshelf', x, y))
        // añade aquí el resto de objetos a 'landmarks', sustituyendo 'x' e 'y' con los valores de posición

        rospy.init_node('location_monitor')
        rospy.Subscriber("/odom", Odometry, callback)
        rospy.spin()
...
```

Ahora llega el momento de incorporar el resto de funcionalidades al script. Esta podría ser una forma de hacerlo:

a) crea un clase llamada **LandmarkMonitor** con las siguientes características:

   i) el constructor tendrá dos atributos, por un lado, recibirá la lista con las posiciones de los objetos, y por otro, un objeto **pub** (necesario en todo publicador).

   ii) la función **callback** definida anteriormente pasará a ser un método de esta clase, realizando lo siguiente:

   - guardará la posición del robot (ya implementado)
   - calculará la posición de cada objeto del escenario (usando la función distance, implementada anteriormente)
   - informará, publicándolo, del nombre del objeto más cercano y la distancia que lo separa del mismo, usando para ello el mensaje **LandmarkDistance** creado previamente. Recuerda que mediante el comando *rostopic echo <nombre_de_topico>* puedes visualizar los datos asociados al tópico en cuestión.
   - avisará por consola cuando el objeto más cercano está a menos de 1 metro.

   iii) añade las líneas necesarias en el **main** para conseguir publicar la posición del objeto más cercano al robot.

Una vez tengas esta modificación hecha, ejecuta el script para ver qué ocurre en consola a la vez que mueves al robot a lo largo del escenario para que el mensaje generado vaya variando (es

decir, que se indique qué objeto está más próximo en cada caso).

Solución (location_monitor.py)

```
#!/usr/bin/env python

import math
import rospy
from nav_msgs.msg import Odometry
from location_monitor.msg import LandmarkDistance

def distance(x1, y1, x2, y2):
  xd = x1-x2
  yd = y1-y2
  return math.sqrt(xd*xd + yd*yd)


class LandmarkMonitor(object):
  def __init__(self, pub, landmarks):
        self._pub = pub
        self._landmarks = landmarks

  def callback(self, msg):

        x = msg.pose.pose.position.x
        y = msg.pose.pose.position.y

       # Calculamos y guardamos la posición al objeto más cercano al robot
        closest_name = None
        closest_distance = None
        for l_name, l_x, l_y in self._landmarks:
              dist = distance(x, y, l_x, l_y)
              if closest_distance is None or dist < closest_distance:
                    closest_name = l_name
                    closest_distance = dist

        # Creamos un mensaje tipo "LandmarkDistance"
        ld = LandmarkDistance()
        ld.name = closest_name
        ld.distance = closest_distance

       #Y publicamos el nombre y posición del objeto más cercano al robot ...
       # ... si la distancia es menor a 1m.
        self._pub.publish(ld)
        if closest_distance < 1:
```

```
                rospy.loginfo('I\'m close to: {}'.format(closest_name))

def main():

    landmarks = []
    landmarks.append(("Bookshelf", 0.00, 1.53))
    landmarks.append(("jersey_barrier", -4.00, -1.00))
    landmarks.append(("Dumpster", 0.99, -3.44))
    landmarks.append(("unit_cylinder_1", -2.00, 1.53))
    landmarks.append(("cube_20k", 1.38, -1.00))

    rospy.init_node("location_monitor")
    pub = rospy.Publisher('closest_landmark', LandmarkDistance, queue_size = 10)
    monitor = LandmarkMonitor(pub, landmarks)

    rospy.Subscriber("/odom", Odometry, monitor.callback)
    rospy.spin()

if __name__ == "__main__":
    main()
```