Róbotica

Práctica 9: Experimentación en ROS2

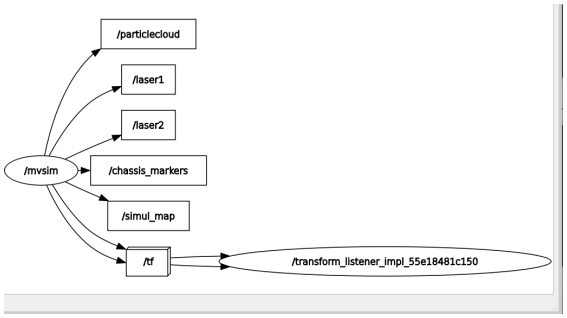
Joaquín Fernández Suárez
jkingxpf@uma.es

Francisco Velasco Romero
franVR@uma.es

INTRODUCCIÓN En este documento se profundizará en la realización de la práctica 9 de la asignatura de robótica del grado de ingeniería informática de la universidad de Málaga.

Assigment 1: Análisis de los tópicos

Al ejecutar la simulación, ejecutamos el comando `rqt_graph` que nos mostrará un grafo con los topics usados en ese momento por `MVSIM`.



Tambien podemos usar el comando de ROS2 `ros2 topic list` que muestra todos los topics que contienen la simulación.

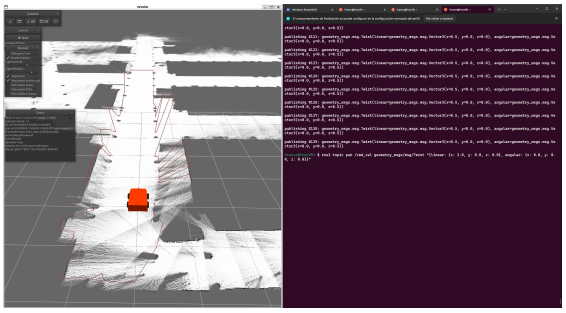
```
franvr@franVR:~/practica_ws$ ros2 topic list
/amcl_pose
/base_pose_ground_truth
/chassis_markers
/chassis_polygon
/clicked_point
/cmd_vel
/goal_pose
/initialpose
/laser1
/laser2
/odom
/parameter_events
/particlecloud
/rosout
/simul_map
/simul_map_metadata
/simul_map_updates
/tf
/tf_static
```

Lista de topics

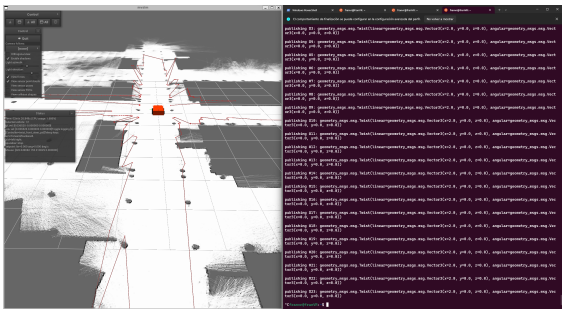
Assigment2: Control manual del robot

Para controlar el robot de forma manual podemos utilizar las teclas de w,a,s,d para mover el robot hacia delante, hacia atrás y girar hacia los lados. Este método utiliza el topic `/cmd_vel` que nos permite mover el robot por el mapa. Otra forma de controlar el robot es mandando el comando directamente al topic, para ello usaremos el siguiente comando en ROS2 para enviar la velocidad lineal y angular que queremos.

`ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "linear: x: 0.5, y: 0.0, z: 0.0, angular: x: 0.0, y: 0.0, z: 0.0"`



Posición inicial del robot



Posición final del robo

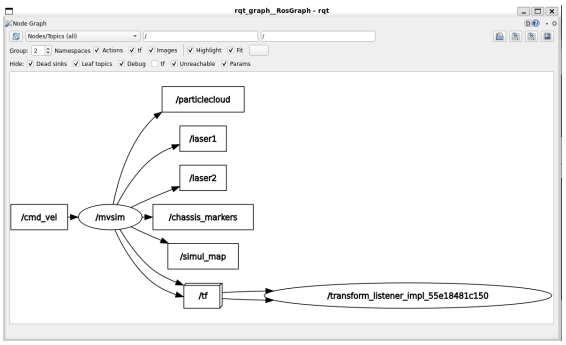


Gráfico de los topics con el comando ejecutado

Assigment 3: Implementación de navegación reactiva

En esta parte de la práctica usaremos conocimientos adquiridos en prácticas anteriores como el algoritmo *Potential Fields* que será la base de nuestro algoritmo.

Para poder comenzar primero necesitaremos recoger información del entorno del robot, para ello nos subcribiremos a los siguientes tres topics `/laser1`, `/base_pose_ground_truth`, `/goal_pose`. Para subscribirnos utilizaremos las librerías

Listing 1. Librerias subscripción

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from nav_msgs.msg import Odometry
from geometry_msgs.msg import PoseStamped,
Twist
```

Permitiendo acceder a la información recolectada por el sensor láser posicionado en la parte delantera del robot, obtener información relacionada con la posición y orientación y las coordenadas de la posición final.

También será necesario poder publicar al topic `/cmd_vel` el movimiento requerido para llegar a la posición final. Por tanto, importaremos la siguiente librería que nos permitirá mandar mensajes al topic.

Listing 2. Librerias publicar

```
from geometry_msgs.msg import PoseStamped,
Twist
```

Estas se encuentran a partir de usar los comandos `ros2 topic list -t` y `ros2 interface show <topic_type>`

```
franvr@franVR:~$ ros2 topic list -t
/amcl_pose [geometry_msgs/msg/PoseWithCovarianceStamped]
/base_pose_ground_truth [nav_msgs/msg/Odometry]
/chassis_markers [visualization_msgs/msg/MarkerArray]
/chassis_polygon [geometry_msgs/msg/Polygon]
/clicked_point [geometry_msgs/msg/PointStamped]
/cmd_vel [geometry_msgs/msg/Twist]
/goal_pose [geometry_msgs/msg/PoseStamped]
/initialpose [geometry_msgs/msg/PoseWithCovarianceStamped]
/laser1 [sensor_msgs/msg/LaserScan]
/laser2 [sensor_msgs/msg/LaserScan]
/odom [nav_msgs/msg/Odometry]
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/particlecloud [geometry_msgs/msg/PoseArray]
/rosout [rcl_interfaces/msg/Log]
/simul_map [nav_msgs/msg/OccupancyGrid]
/simul_map_metadata [nav_msgs/msg/MapMetaData]
/simul_map_updates [map_msgs/msg/OccupancyGridUpdate]
/tf [tf2_msgs/msg/TFMessage]
/tf_static [tf2_msgs/msg/TFMessage]
```

Lista de topicos con tipo de mensajes

```
franvr@franVR:~$ ros2 interface show sensor_msgs/msg/LaserScan
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

std_msgs/Header header # timestamp in the header is the acquisition time of
        builtin_interfaces/Time stamp
                int32 sec
                uint32 nanosec
        string frame_id
                        # the first ray in the scan.
                        #
                        # in frame frame_id, angles are measured around
                        # the positive Z axis (counterclockwise, if Z is up)
                        # with zero angle being forward along the x axis

float32 angle_min       # start angle of the scan [rad]
float32 angle_max       # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your scanner
                        # is moving, this will be used in interpolating position
                        # of 3d points
float32 scan_time       # time between scans [seconds]

float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]

float32[] ranges        # range data [m]
                        # (Note: values < range_min or > range_max should be discarded)
float32[] intensities   # intensity data [device-specific units].  If your
                        # device does not provide intensities, please leave
                        # the array empty.
```

Información de tópico

Con las librerías ya importadas comenzaremos a recolectar información,

Listing 3. Recepción de mensajes

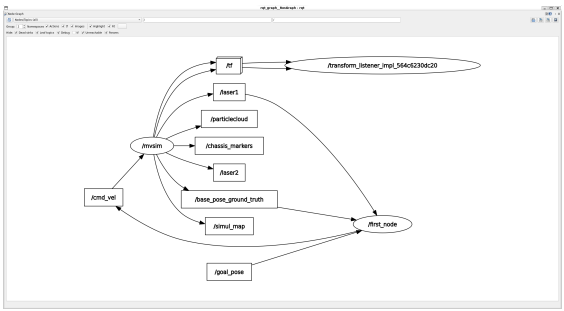
```
def callbackLaser(self, msg):
    self.laser = msg
    self.messages_received['laser'] = True
    self.calculate_and_publish()

def callbackGoalPose(self, msg):
    if self.messages_received
    ['goalPose'] == False:
        self.goalPose =
        np.vstack([[msg.pose.position.x]
        , [msg.pose.position.y]])
        self.messages_received['goalPose']
        = True
        self.calculate_and_publish()

def callbackTruePose(self, msg):
    # Almacenar datos de Odometry
    self.true_pose_data = msg
    pose_data = msg.pose.pose
    position_x = pose_data.position.x
    position_y = pose_data.position.y

    # Supongamos que tienes un
    mensaje Odometry llamado "odometry_msg"
    q0 = msg.pose.pose.orientation.w
    q1 = msg.pose.pose.orientation.x
    q2 = msg.pose.pose.orientation.y
    q3 = msg.pose.pose.orientation.z

    # Calcula los  ngulos  de
    Euler (yaw, pitch, roll)
    yaw_angle = np.arctan2
    (2 * (q0*q3 + q1*q2),
    1 - 2 * (q2**2 + q3**2))

    self.trueTheta = yaw_angle
    self.truePose =
    np.vstack([[position_x],[position_y]])
    self.messages_received['truePose']
    = True
    self.calculate_and_publish()
```

En esta parte del código recogemos la información aportada por los topics subscritos y empezamos a ejecutar el algoritmo, pero ello podría producir errores por la falta de datos. Para prevenir este error haremos que el código ”espere” a que se haya recolectado toda la información para poder ejecutar el algoritmo.

Listing 4. Código de espera

```
def calculate_and_publish(self):
    if all(self.messages_received.values()):
        # Todos los mensajes han sido
        #recibidos,
        #ahora puedes calcular y publicar
        RadiusOfInfluence = 4

        KObstacles =
        self.calculate_obstacles
        (self.laser.ranges, RadiusOfInfluence)

        v, theta =
        self.total_force(self.truePose,
        self.goalPose, self.laser,
        RadiusOfInfluence, KObstacles, 1.75)

        if self.destinoFinal == False:
            self.publish_cmd_vel(v, theta)
        # Reiniciar las banderas y
        #esperar para recibir nuevos valores

        self.messages_received =
        {'laser': False, 'truePose': False}
        self.laser = None
        self.truePose = None
```

Con todo esto ya podremos ejecutar el algoritmo de *Potential Fields* recogido en las funciones `repulsive_force` `attractive_force`, `total_force`.Durante la ejecución nuestro nodo aparecerá reflejado en `rqt_graph` junto a los nodos topics de la simulación.



En este grafo se ve como nuestro nodo esta subscrito a los topics `/laser1`, `/base_pose_ground_truth`,`/goal_pose` y como publica a `/cmd_vel` que a su vez publica en `mvsim`

Para finalizar veremos un video donde se ve el robot avanzando desde el punto inicial al punto final sin chocar con ningún obstáculo usando el código desarrollado en la práctica.