

Họ và tên: Đỗ Đức Mạnh

Mã sinh viên: 20020688

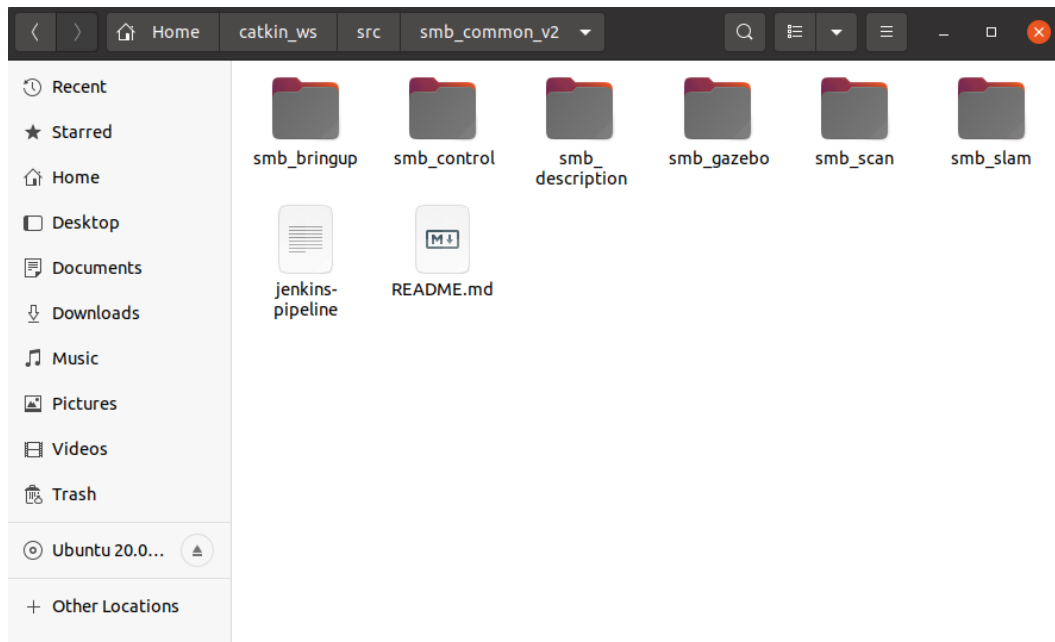
Báo cáo thực hành tuần 3

Môn học: Lập trình ROS

Bài 1: Yêu cầu: Cài đặt file *smb_common_v2* mới.

B1: Tải file *smb_common_v2* từ course.

B2: Giải nén file đó tại đường dẫn `/home/manh/catkin_ws/src`, ta thu được các file như *Hình 1*. Ngoài các file gốc như *smb_control*, *smb_description*, *smb_gazebo* tôi có tạo thêm các file để thực hiện các bài tập sau.

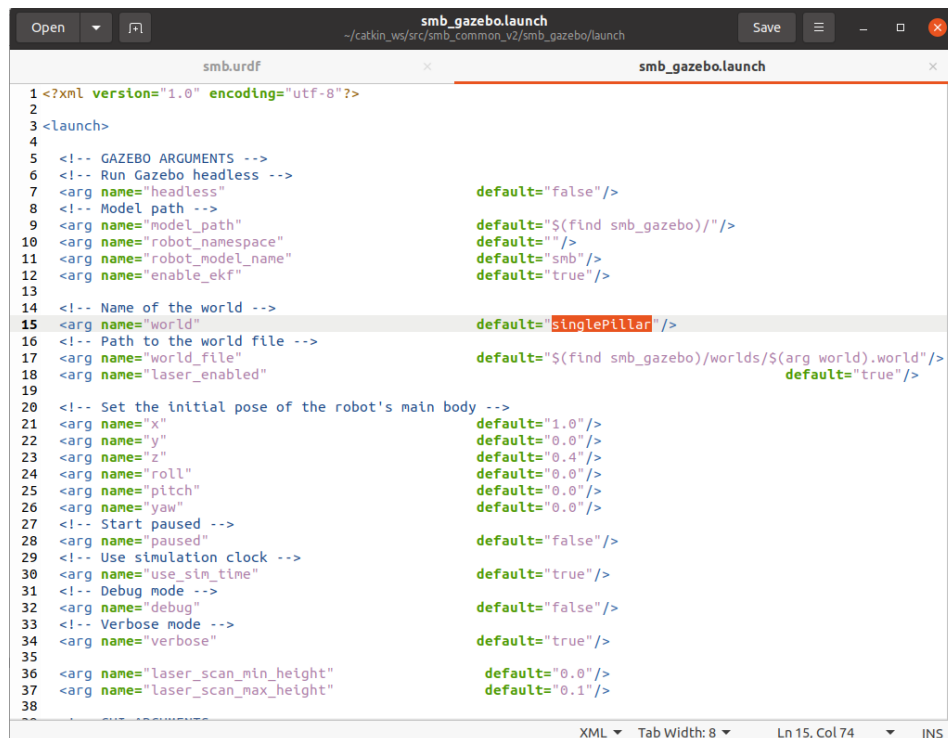


Hình 1. Các file trong smb_common_v2.

B3: Biên dịch file trên bằng câu lệnh `$ catkin_make`.

Bài 2: Yêu cầu: Thay đổi môi trường tại file *smb_gazebo.launch*.

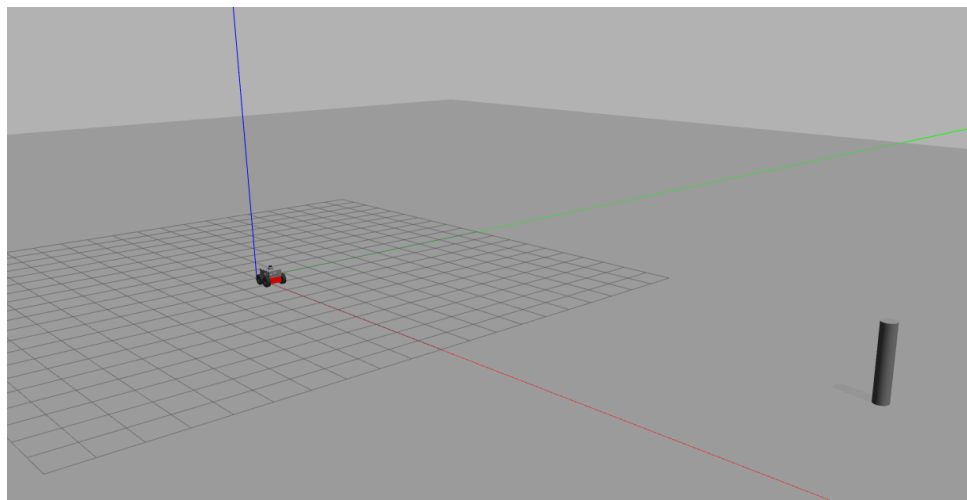
B1: Tại file *smb_gazebo.launch* dòng thứ 15, tôi thay đổi biến môi trường *singlePillar* như *Hình 2*.



```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <launch>
4
5 <!-- GAZEBO ARGUMENTS -->
6 <!-- Run Gazebo headless -->
7 <arg name="headless" default="false"/>
8 <!-- Model path -->
9 <arg name="model_path" default="$(find smb_gazebo)/"/>
10 <arg name="robot_namespace" default="/">
11 <arg name="robot_model_name" default="smb"/>
12 <arg name="enable_ekf" default="true"/>
13
14 <!-- Name of the world -->
15 <arg name="world" default="singlePillar"/>
16 <!-- Path to the world file -->
17 <arg name="world_file" default="$(find smb_gazebo)/worlds/$(arg world).world"/>
18 <arg name="laser_enabled" default="true"/>
19
20 <!-- Set the initial pose of the robot's main body -->
21 <arg name="x" default="1.0"/>
22 <arg name="y" default="0.0"/>
23 <arg name="z" default="0.4"/>
24 <arg name="roll" default="0.0"/>
25 <arg name="pitch" default="0.0"/>
26 <arg name="yaw" default="0.0"/>
27 <!-- Start paused -->
28 <arg name="paused" default="false"/>
29 <!-- Use simulation clock -->
30 <arg name="use_sim_time" default="true"/>
31 <!-- Debug mode -->
32 <arg name="debug" default="false"/>
33 <!-- Verbose mode -->
34 <arg name="verbose" default="true"/>
35
36 <arg name="laser_scan_min_height" default="0.0"/>
37 <arg name="laser_scan_max_height" default="0.1"/>
38
```

Hình 2. Thay đổi biến môi trường trong `smb_gazebo.launch`.

B2: Chạy file `smb_gazebo.launch` để quan sát môi trường `singlePillar` như Hình 3.



Hình 3. Môi trường `singlePillar`.

Bài 3: Yêu cầu: Tìm ra vị trí của cây cột so với vị trí của robot .

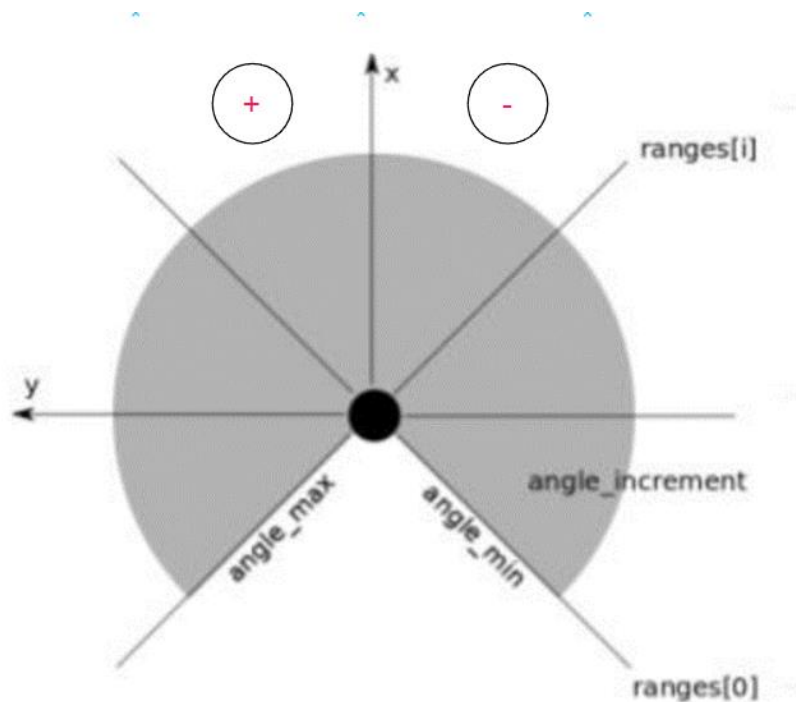
B1: Đọc thông tin từ topic / scan khi chạy file smb_gazebo.launch.

[illegible]

B2: Tạo file `find_pillar.py` để xác định vị trí trục của cây cột.

```
smb_navigation > src > ➤ find_pillar.py
1 import rospy
2 from sensor_msgs.msg import LaserScan
3
4 def callback(msg):
5     sum = 0
6     count = 0
7     for i in range(302):
8         if (msg.ranges[i]<50):
9             sum = sum + i
10            count = count + 1
11
12    # Xac dinh truc cua cay cot nam o goc bao nhieu so voi robot
13    central = int((int(sum / count) * 270 / 302) - 135)
14
15    # In ra vi tri cua truc cay cot voi vi tri goc va khoảng cách so voi robot
16    print("pillar o goc: {}".format(central))
17    print("khoảng cách đến pillar: {}(m)".format(msg.ranges[int(sum/count)]))
18
19 rospy.init_node('find pillar')
20 sub = rospy.Subscriber('scan',LaserScan,callback)
21 rospy.spin()
```

Hình 5. File `find_pillar.py`.



Hình 6. Trục tọa độ trên robot.

```
manh@manh:~/catkin_ws$ rosrund smb_navigation find_pillar.py
pillar o goc: 51(do)
khoang cach den pillar:19.49006462097168(m)
-----
pillar o goc: 51(do)
khoang cach den pillar:19.49388885498047(m)
-----
```

Hình 7. Vị trí của cây cột so với robot hiển thị trên terminal.

Bài 4: Yêu cầu: Tạo một publisher cho topic /cmd_vel để cung cấp vận tốc cho robot.

B1: Tìm kiểu dữ liệu cung cấp cho topic /cmd_vel bằng câu lệnh: \$ rostopic type /cmd_vel. Hình 8a và Hình 8b thể hiện kiểu dữ liệu cung cấp vận tốc cho robot.

```
manh@manh:~/catkin_ws$ rostopic type /cmd_vel
geometry_msgs/Twist
```

Hình 8a. Tên kiểu dữ liệu cấp vận tốc cho robot.

```

manh@manh:~/catkin_ws$ rosmmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z

```

Hình 8b. Kiểu dữ liệu cấp vận tốc cho robot.

B2: Tạo file `publisher_vel.py` để cấp vận tốc cho topic `/cmd_vel`.

Hình 9a và Hình 9b biểu diễn file `publisher_vel.py`. Từ B1, tôi xác định được kiểu dữ liệu cấp cho topic `/cmd_vel` và sử dụng thư viện chứa kiểu dữ liệu đó trong file này. Tôi cung cấp vận tốc cho xe bằng cách nhập lần lượt 6 thông số của vận tốc tuyến tính và vận tốc quay từ bàn phím. Ngoài ra tôi có thể điều khiển xe bằng cách nhập các nút từ bàn phím.

```

smb_navigation > src > publisher_vel.py
1  #!/usr/bin/env python3
2  import rospy
3  from geometry_msgs.msg import Twist
4
5  class movement:
6      def __init__(self):
7          rospy.init_node('pub_move_robot', anonymous=False)
8          self.pub_move = rospy.Publisher("/cmd_vel", Twist, queue_size=10)
9          self.move = Twist()
10
11      def publish_vel(self):
12          self.pub_move.publish(self.move)
13
14      def move_forward(self):
15          self.move.linear.x = 0.5
16          self.move.angular.z = 0.0
17
18      def move_left(self):
19          self.move.linear.x = 0.0
20          self.move.angular.z = 0.2
21
22      def move_right(self):
23          self.move.linear.x = 0.0
24          self.move.angular.z = -0.2
25
26      def stop(self):
27          self.move.linear.x = 0.0
28          self.move.angular.z = 0.0
29
30      def move_backward(self):
31          self.move.linear.x = -0.5
32          self.move.angular.z = 0.0
33
34      def pub_vel(self, li_x, li_y, li_z, an_x, an_y, an_z):
35          self.move.linear.x = li_x
36          self.move.linear.y = li_y
37          self.move.linear.z = li_z
38          self.move.angular.x = an_x
39          self.move.angular.y = an_y
40          self.move.angular.z = an_z
41

```


Vì vậy, tôi đã dùng phương pháp khác là tôi sẽ chọn vị trí tâm cây cột là trung bình cộng của các tia thu được khoảng cách và khoảng cách từ robot đến tâm cây cột là trung bình cộng các khoảng cách thu được. Điều này đảm bảo thu được giá trị gần đúng của vị trí cây cột so với robot. *Hình 11* là file `p_controller.py` tôi thể hiện phương pháp này.

```
smb_navigation > src > p_controller.py
1  #!/usr/bin/env python3
2  import rospy
3  from geometry_msgs.msg import Twist
4  from sensor_msgs.msg import LaserScan
5
6  class controller:
7      def __init__(self):
8          rospy.init_node('controller', anonymous=False)
9          self.sub = rospy.Subscriber('/scan', LaserScan, self.callback)
10         self.data = LaserScan()
11         self.pub_move = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
12         self.move = Twist()
13
14     def callback(self, msg):
15         self.data = msg
16         self.sum_ang = 0
17         self.sum_dis = 0
18         self.count = 0
19         for i in range(302):
20             if (self.data.ranges[i] < 50):
21                 self.sum_ang = self.sum_ang + i
22                 self.sum_dis = self.sum_dis + self.data.ranges[i]
23                 self.count = self.count + 1
24         self.ang = float(float(self.sum_ang / self.count) * 270 / 362) - 135
25         self.dis = float(self.sum_dis / self.count)
26         print("pillar o goc: {}".format(self.ang))
27         print("khoang cach den pillar: {}(m)".format(self.dis))
28         print("-----")
29         self.kp = 0.05
30         if self.dis < 5:
31             self.move.linear.x = 0
32         else:
33             self.move.linear.x = self.kp * self.dis
34
35         if abs(self.ang) < 1:
36             self.move.angular.z = 0
37         else:
38             self.move.angular.z = self.kp * self.ang
39
40         self.pub_move.publish(self.move)
41         print("li_x: {}".format(self.move.linear.x))
42         print("an_x: {}".format(self.move.angular.z))
43         print("=====")
44
45 if __name__ == "__main__":
46     ctl = controller()
47
48     rospy.spin()
49
50
```

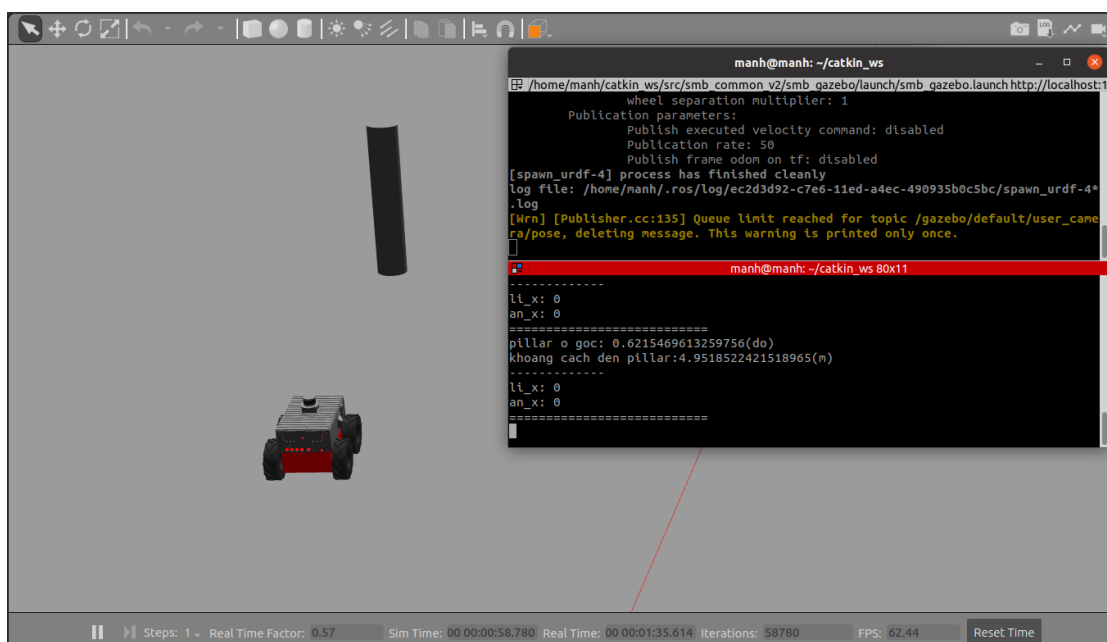
Hình 11. File `p_controller.py`.

B2: Sử dụng P controller để cấp vận tốc cho robot đi đến cây cột theo hai giá trị góc và khoảng cách cây cột.

Để di chuyển tới vị trí cây cột nằm trước mặt robot, tôi sử dụng sai số giữa góc cây cột và trục Ox của robot để làm đầu vào cho P controller và sau đó cấp vận tốc quay vào angular.z của robot.

Để robot cách cây cột 5(m), tôi sử dụng khoảng cách tới cây cột làm đầu vào cho P controller và cấp vận tốc tuyến tính vào linear.x của robot.

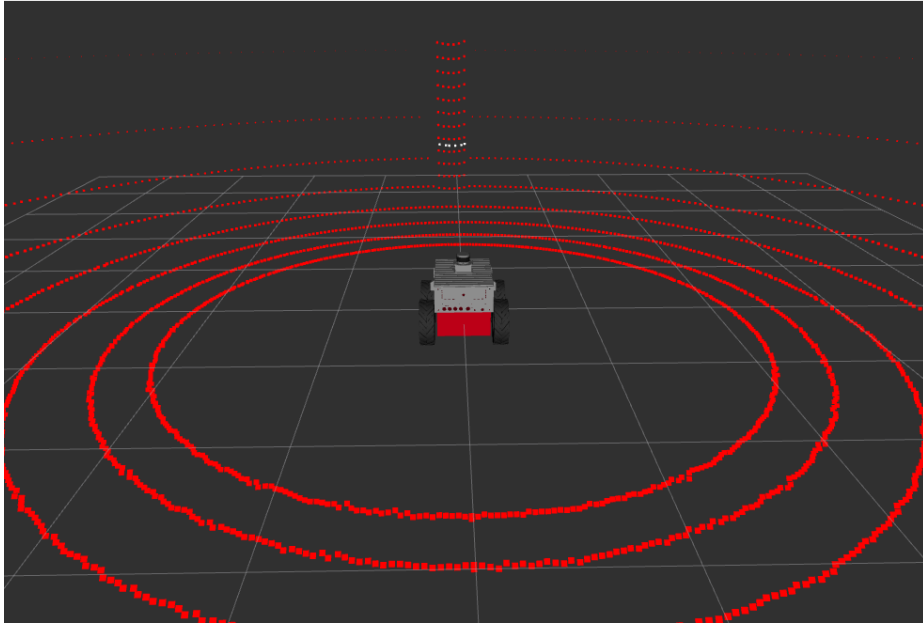
Hình 11 thể hiện phần P controller cho robot trong hàm callback(). Hình 12 thể hiện kết quả khi robot sử dụng P controller.



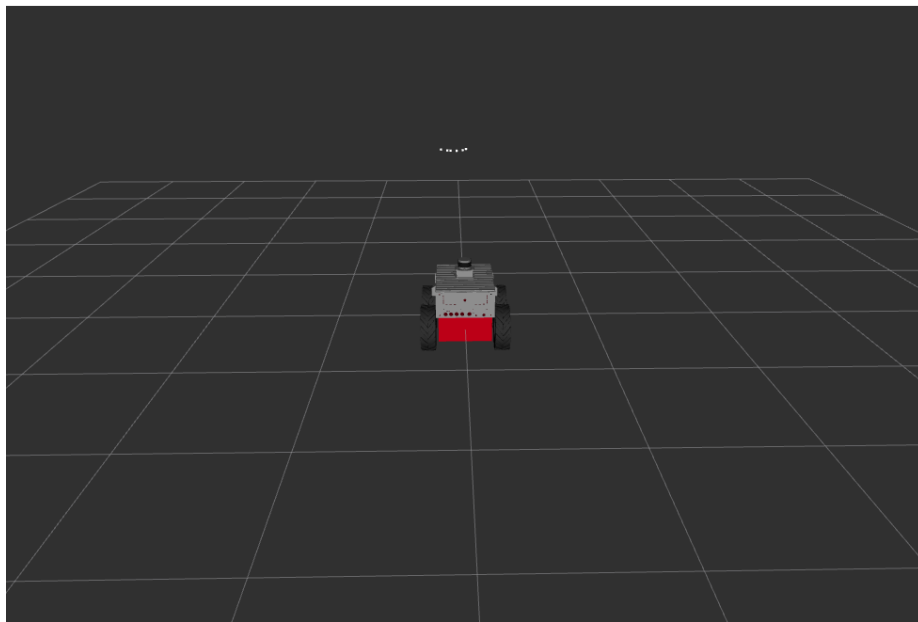
Hình 12. Robot di chuyển đến cách cây cột 5(m) và cây cột nằm trên trục Ox của robot.

Bài 6: Yêu cầu: Trực quan hóa trên Rviz.

Sau khi thực hiện **Bài 5**, trực quan hóa trên Rviz tôi làm tương tự như trong các tuần trước. Khởi động Rviz và thêm các topic để hiển thị các thành phần trong môi trường mô phỏng. Hình 13a thể hiện các đám mây điểm 3D robot thu được, có thể thấy cây cột đang ở trước mặt robot. Hình 13b thể hiện đám mây điểm 2D thu từ laser với màu trắng.



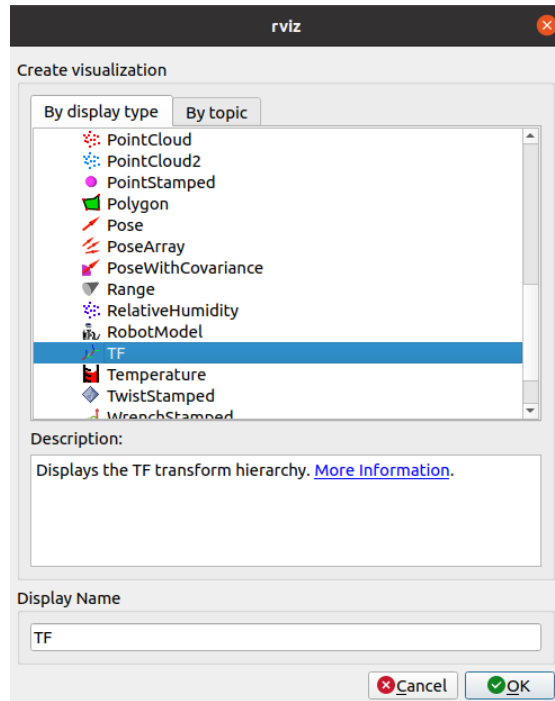
Hình 13a. Đám mây điểm 3D thu được từ topic /rslidar.



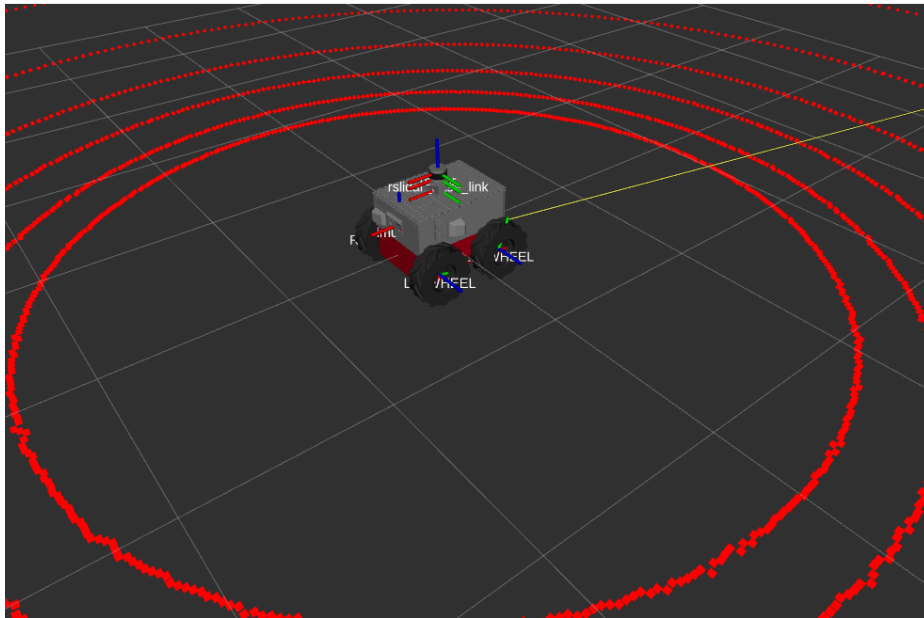
Hình 13b. Đám mây điểm 2D thu được từ topic /scan.

Bài 7: Yêu cầu: Thêm cấu hình TF trên Rviz.

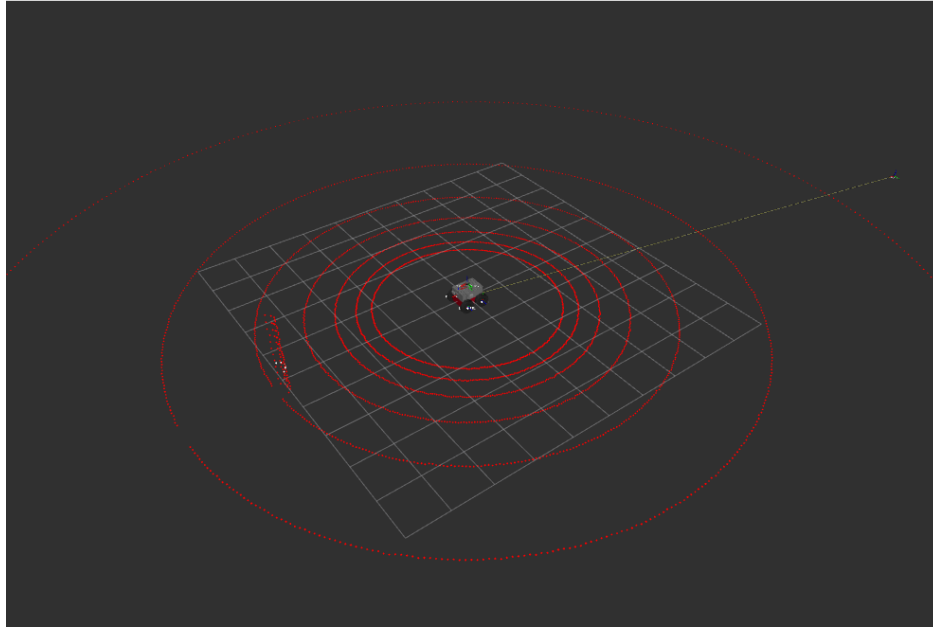
Để thêm cấu hình TF trên Rviz tôi thêm TF tại phần “Add” trong Rviz như *Hình 14*. Tôi sẽ thu được kết quả như *Hình 15a* và *Hình 15b*. *Hình 14a* thể hiện các TF gắn trên robot và *Hình 14b* thể hiện thêm TF lại gốc tọa độ của môi trường.



Hình 14. Thêm cấu hình TF.



Hình 15a. Các TF trên robot.



Hình 15b. TF tại gốc tọa độ của môi trường.s

Bài 8: Yêu cầu: Thêm Marker trong Rviz để trực quan vị trí cây cột ước tính.

Để trực quan vị trí cây cột ước tính thì tôi sử dụng thư viện `visualization/Marker`. Thư viện này cho phép thêm các khối như hình trụ, hình cầu,... cùng các thông số màu sắc và vị trí để hiển thị trên Rviz. *Hình 16* thể hiện các thông số thể hiện trên cung cấp được lấy từ trang chủ *ros.org*.

```

# See http://www.ros.org/wiki/rviz/Display

uint8 ARROW=0
uint8 CUBE=1
uint8 SPHERE=2
uint8 CYLINDER=3
uint8 LINE_STRIP=4
uint8 LINE_LIST=5
uint8 CUBE_LIST=6
uint8 SPHERE_LIST=7
uint8 POINTS=8
uint8 TEXT_VIEW_FACING=9
uint8 MESH_RESOURCE=10
uint8 TRIANGLE_LIST=11

uint8 ADD=0
uint8 MODIFY=0
uint8 DELETE=2
uint8 DELETEALL=3

Header header # h
string ns # N
int32 id # obj
int32 type # Typ
int32 action # 0
geometry_msgs/Pose pose
geometry_msgs/Vector3 scale
std_msgs/ColorRGBA color # C
duration lifetime # H
bool frame_locked # I

#Only used if the type specified has some
geometry_msgs/Point[] points
#Only used if the type specified has some
#number of colors must either be 0 or eq
#NOTE: alpha is not yet used
std_msgs/ColorRGBA[] colors

# NOTE: only used for text markers
string text

# NOTE: only used for MESH_RESOURCE marker
string mesh_resource
bool mesh_use_embedded_materials

```

Hình 16. Các dữ liệu trong thư viện *visualization/Marker*.

Từ vị trí và góc của cây cột so với robot tính toán từ các bài trên. Tôi sẽ tính toán vị trí cây cột theo trục Ox, Oy so với khung tọa độ của robot, từ đó có thể trực quan cây cột trên Rviz.

B1: Chỉnh sửa file `p_controller.py`, thêm phần trực quan thông qua thư viện `visualization/Marker`.

Hình 17 thể hiện sự chỉnh sửa của tôi tại file `p_controller.py`. Tôi sẽ đẩy dữ liệu về marker lên topic `/marker_basic` và sẽ hiển thị topic này trên Rviz. Marker này sẽ có những thông số như khung gốc (`frame_id`), kiểu dáng là hình trụ (`type = CYLINDER`), vị trí (`position`), màu sắc (`color`),... Tôi chọn khung `base_inertia` vì lấy khung robot làm gốc để thể hiện vị trí của cây cột.

```

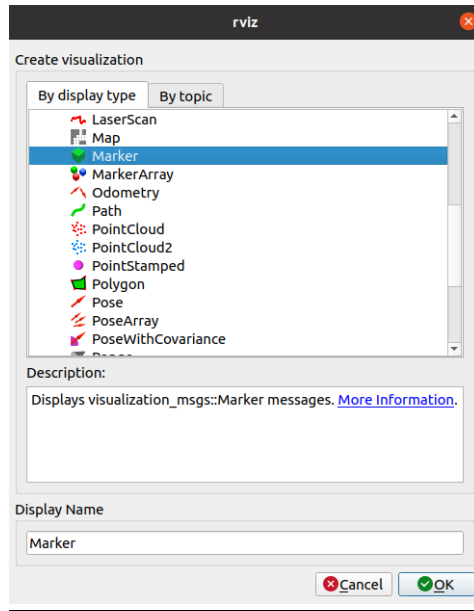
smb_navigation > src > p_controller.py
12     rospy.init_node('controller', anonymous=False)
13     self.sub = rospy.Subscriber('/scan', LaserScan, self.callback)
14     self.data = LaserScan()
15     self.pub_move = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
16     self.move = Twist()
17
18     #marker
19     self.marker_objectlsher = rospy.Publisher('/marker_basic', Marker, queue_size=1)
20     self.marker_object = Marker()
21
22     def callback(self, msg):
23         self.data = msg
24         self.sum_ang = 0
25         self.sum_dis = 0
26         self.count = 0
27         for i in range(360):
28             if (self.data.ranges[i]<50):
29                 self.sum_ang = self.sum_ang + i
30                 self.sum_dis = self.sum_dis + self.data.ranges[i]
31                 self.count = self.count + 1
32         self.ang = float(float(self.sum_ang / self.count) * 270 / 360) - 135
33         self.dis = float(self.sum_dis/self.count)
34         print("pillar o goc: {}".format(self.ang))
35         print("khoảng cách đến pillar: {}".format(self.dis))
36         print("-----")
37         self.kp = 0.05
38         if self.dis < 5:
39             self.move.linear.x = 0
40         else:
41             self.move.linear.x = self.kp * self.dis
42
43         if abs(self.ang) < 1:
44             self.move.angular.z = 0
45         else:
46             self.move.angular.z = self.kp * self.ang
47
48         self.pub_move.publish(self.move)
49         print("li x: {}".format(self.move.linear.x))
50         print("an x: {}".format(self.move.angular.z))
51         print("=====")
52
53         #marker
54         self.marker_object.header.frame_id = "base_inertia"
55         self.marker_object.header.stamp = rospy.Time.now()
56         self.marker_object.id = 0
57         self.marker_object.type = Marker.CYLINDER
58
59         self.marker_object.pose.position.x= self.dis * math.cos(self.ang * 3.14/180)
60         self.marker_object.pose.position.y= self.dis * math.sin(self.ang * 3.14/180)
61         self.marker_object.pose.position.z=0
62         self.marker_object.pose.orientation.x = 0.0
63         self.marker_object.pose.orientation.y = 0.0
64         self.marker_object.pose.orientation.z = 0.0
65         self.marker_object.pose.orientation.w = 1.0
66
67         self.marker_object.scale.x = 0.3
68         self.marker_object.scale.y = 0.3
69         self.marker_object.scale.z = 1.0
70
71         self.marker_object.color.r = 0.0
72         self.marker_object.color.g = 0.0
73         self.marker_object.color.b = 1.0
74         self.marker_object.color.a = 1.0
75
76         self.marker_object.lifetime = rospy.Duration(0)
77
78         self.marker_objectlsher.publish(self.marker_object)

```

Hình 17. Thêm phần hiển thị Marker trong *p_controller.py*.

B2: Hiển thị marker trên Rviz.

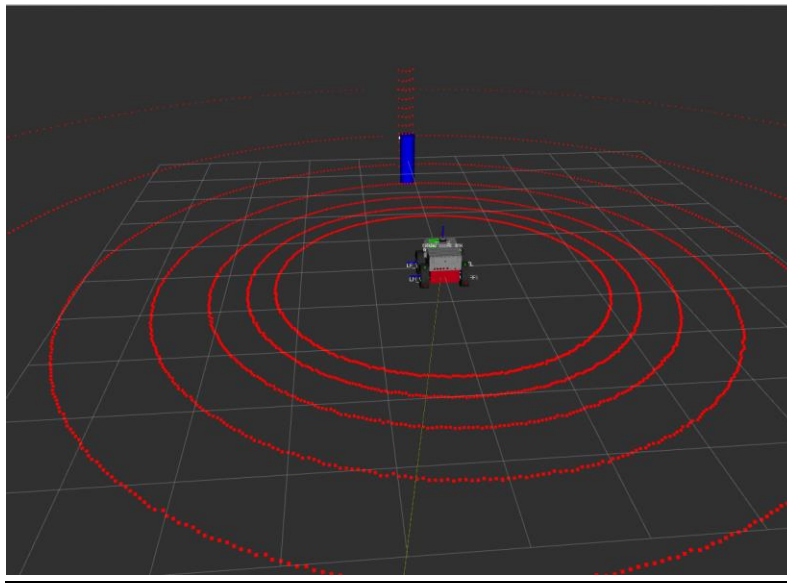
Tôi sẽ thêm Marker như thêm cấu hình TF ở **Bài 7**. Hình 18 thể hiện cách thêm cấu hình Marker. Cần chọn đúng topic là /marker_basic để hiển thị.



Hình 18. Thêm cấu hình Marker.

B3: Xem kết quả sau khi thêm Marker.

Hình 19 thể hiện kết quả, Marker đã trùng với đám mây điểm của cây cột.



Hình 19. Trực quan hóa Marker trên Rviz.

