

Họ và tên: Đỗ Đức Mạnh

Mã sinh viên: 20020688

Báo cáo Bài tập tuần 4

Môn: Lập trình ROS

Bài 1: Yêu cầu: Sử dụng `rqt_multiplot` để trực quan hóa đường đi của robot trên mặt phẳng Oxy.

Công cụ `rqt_multiplot` để trực quan hay vẽ lại một số thông tin dưới dạng đồ thị.

B1: Tải gói `rqt_multiplot`.

Để tải gói `rqt_multiplot`, thực hiện câu lệnh:

```
$ sudo apt install ros-noetic-rqt-multiplot*
```

B2: Khởi chạy môi trường, chuẩn bị trực quan hóa trên `rqt_multiplot`.

Khởi chạy môi trường cùng robot như các bài tập trước.

Để trực quan hóa trên `rqt_multiplot`, thực hiện câu lệnh:

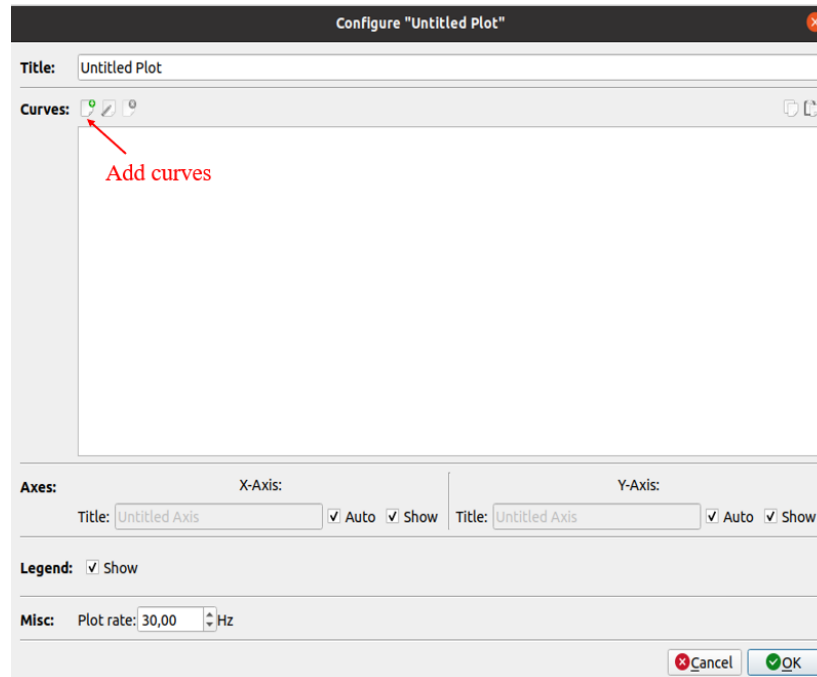
```
$ rosrun rqt_multiplot rqt_multiplot
```

Màn hình sẽ hiện như *Hình 1*.

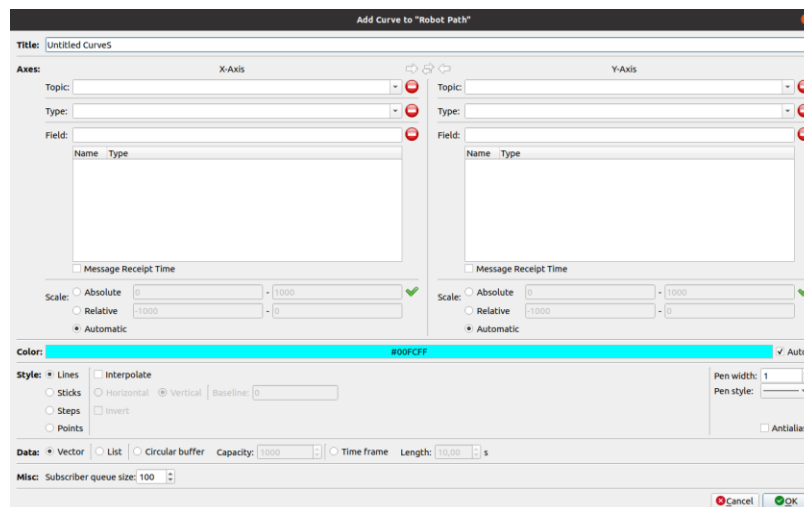


Hình 1. Giao diện của rqt_multiplot.

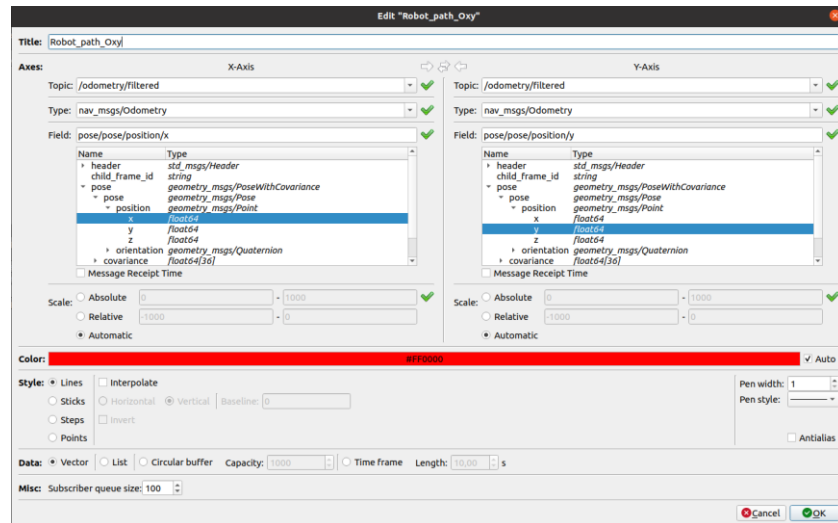
Sau đó nhấn icon “Config plot” hiển thị như *Hình 2a*. Tiếp tục nhấn icon “Add curves”, màn hình sẽ hiển thị như *Hình 2b*. Thông tin đường đi của robot được thể hiện tại topic /odometry/filtered, topic này dựa trên số vòng quay của bánh xe để ước tính ra vị trí của robot. Để trực quan hóa trên mặt phẳng Oxy thì cần gọi thông tin của pose/position/x và pose/position/y từ msg nav_msgs/Odometry của topic /odometry/filtered như *Hình 2c*. Sau đó lưu và quay lại giao diện như *Hình 1*.



Hình 2a. Giao diện thêm/bớt đồ thị.



Hình 2b. Giao diện chọn thông tin cho đồ thị.



Hình 2c. Chọn thông tin cho hệ tọa độ Oxy.

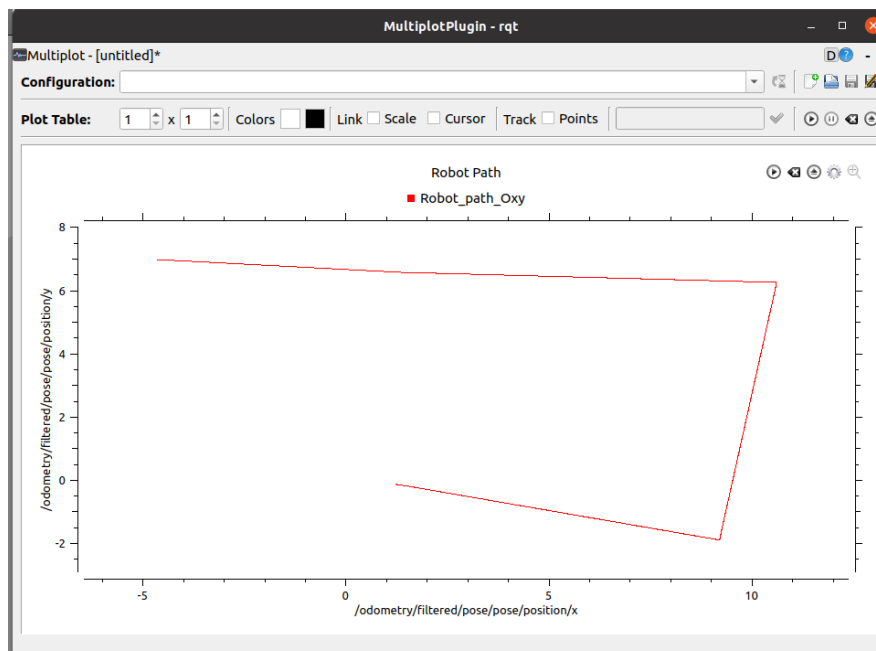
B3: Điều khiển robot và trực quan hóa đường đi.

Sử dụng bàn phím để điều khiển robot qua câu lệnh:

```
$ rosrn teleop_twist_keyboard teleop_twist_keyboard.py
```

Nhấn icon “Run” trên Hình 1 để trực quan hóa đường đi của robot trên mặt phẳng Oxy, thu được Hình 3.

Nhận xét: Tôi thực hiện di chuyển robot trên đường thẳng theo 3 giai đoạn song song với trục Ox và Oy; tuy nhiên có thể thấy đường đi robot bị lệch, kết quả việc ước lượng tại Hình 3 bị ảnh hưởng do sai số tích lũy của phép đo odometry.



Hình 3. Trực quan hóa ước tính đường đi của Robot.

Bài 2: Yêu cầu: Sử dụng `rosvbag` để lưu lại thông tin cảm biến trong quá trình 15s robot di chuyển.

Gói `rosvbag` là một công cụ giúp ghi lại các thông tin, dữ liệu từ các topic và có thể sử dụng lại mỗi khi gọi lên.

B1: Khởi chạy môi trường, xác định các topic thông tin cảm biến.

Liệt kê topic thông tin cảm biến qua câu lệnh:

```
$ rostopic list
```

Thông tin đo cảm biến bánh xe tại topic: `/odometry/filtered`

Thông tin đo cảm biến IMU tại topic: `/imu0`

Thông tin đo cảm biến laser tại topic: `/scan`

B2: Lưu thông tin cảm biến.

Sử dụng gói `rosvbag` lưu thông tin cảm biến qua câu lệnh:

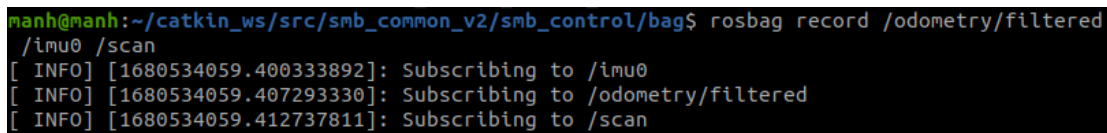
Cách 1: Lưu tất cả các topic:

```
$ rosvbag record -a
```

Cách 2: Chỉ lưu các topic thông tin cảm biến:

```
$ rosvbag record /odometry/filtered /imu0 /scan
```

Hình 4 thể hiện terminal sử dụng cách 2 để lưu lại thông tin cảm biến.



```
manh@manh:~/catkin_ws/src/smb_common_v2/smb_control/bag$ rosvbag record /odometry/filtered /imu0 /scan
[ INFO] [1680534059.400333892]: Subscribing to /imu0
[ INFO] [1680534059.407293330]: Subscribing to /odometry/filtered
[ INFO] [1680534059.412737811]: Subscribing to /scan
```

Hình 4. Giao diện terminal sử dụng rosvbag.

B3: Sử dụng bàn phím điều khiển robot.

Sử dụng `teleop_keyboard` để điều khiển robot trong 15s và sau đó dừng lưu `rosvbag`.

File bag được lưu với tên: `2023-04-03-22-00-59.bag` .

Bài 3: Yêu cầu: Sử dụng `rqt_multiplot` để trực quan hóa đường đi của robot trên mặt phẳng Oxy thông qua file bag lưu ở Bài 2.

File bag khi được sử dụng lại sẽ gọi lại các topic đã lưu cùng với các thông tin của topic.

B1: Chạy file bag đã lưu từ **Bài 2**.

Trước hết cần gọi ros Master bằng câu lệnh:

```
$ roscore
```

Chạy file bag bằng câu lệnh:

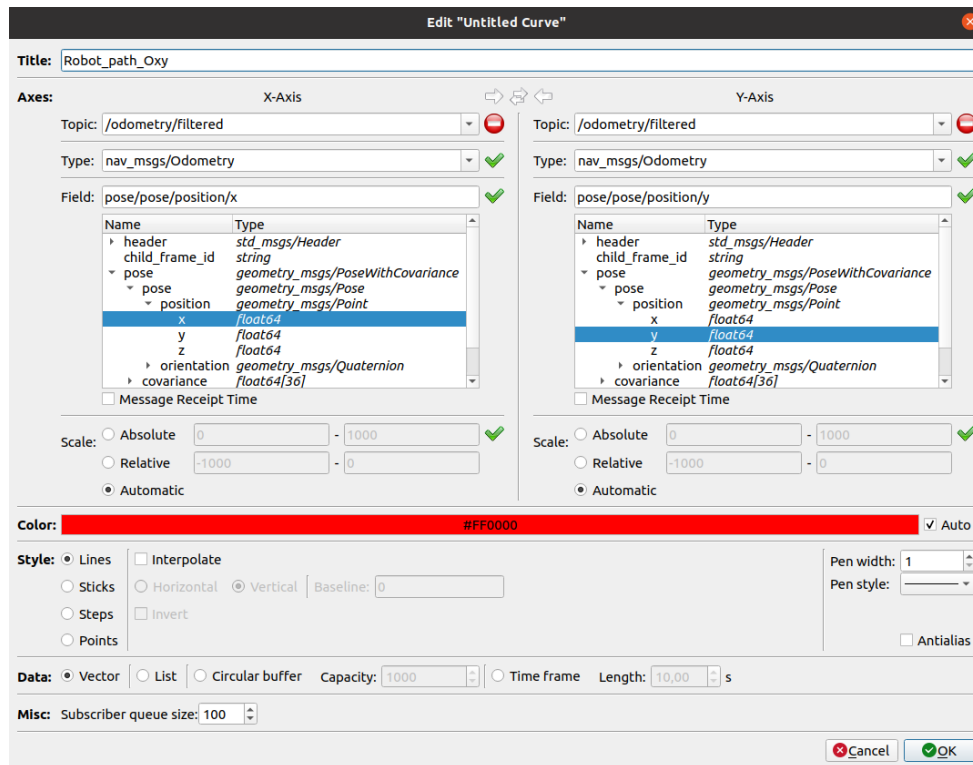
```
$ rosbag play 2023-04-03-22-00-59.bag
```

Có thể kiểm tra lại các topic qua câu lệnh:

```
$ rostopic list
```

B2: Chạy `rqt_multiplot` để trực quan hóa.

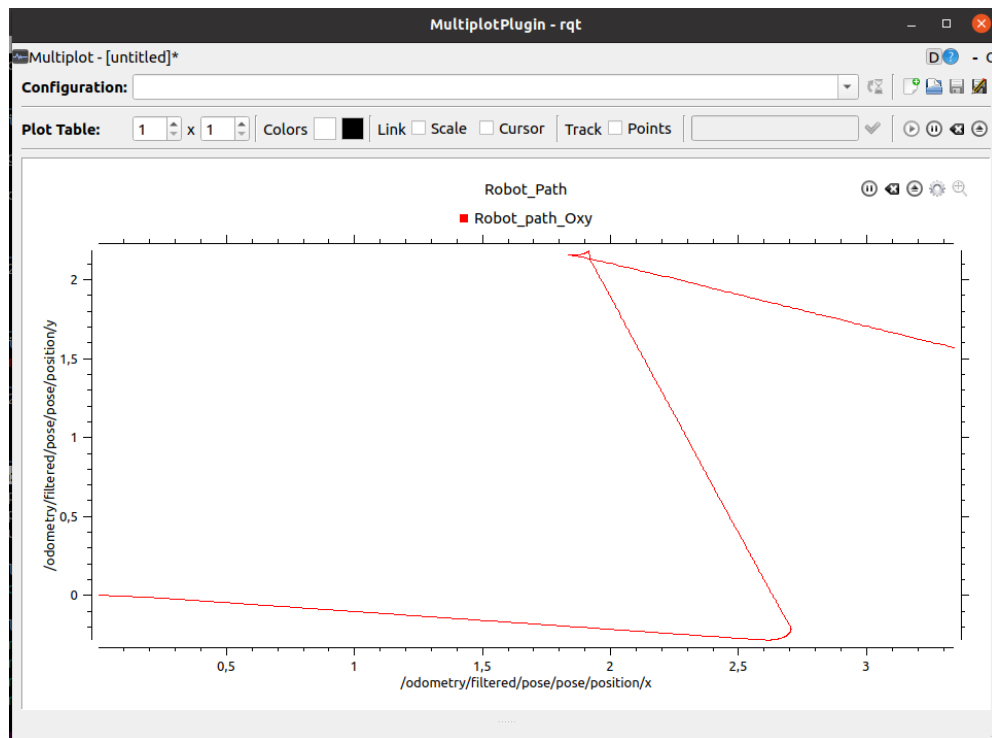
Tôi sẽ sử dụng lại cấu hình từ **Bài 1** để trực quan hóa đường đi robot trên mặt phẳng Oxy như *Hình 5*.



Hình 5. Cấu hình đồ thị đường đi robot.

B3: Kết quả trực quan hóa.

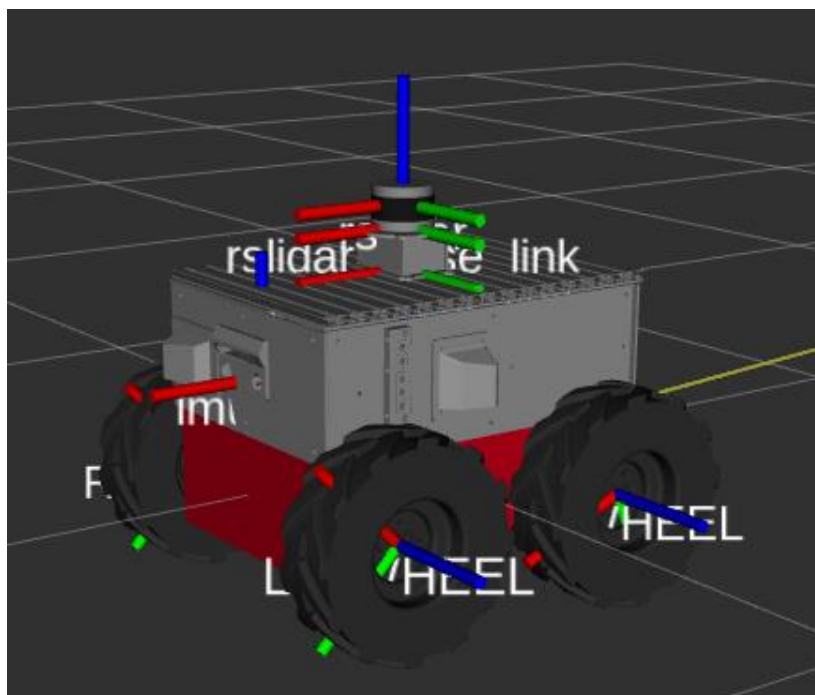
Đường đi của robot qua file bag được trực quan hóa trên mặt phẳng Oxy tại *Hình 6*.



Hình 6. Đồ thị đường đi của Robot trong mặt phẳng Oxy.

Bài 4: Yêu cầu: Trực quan hóa chuyển động của Robot trong Rviz sử dụng TF markers.

Tại Rviz, tôi thêm topic TF để trực quan các trục trên robot như trục tọa độ bánh xe, trục tọa độ thân robot, trục tọa độ laser. Khi robot chuyển động, trục tọa độ bánh xe cũng quay theo chiều quay của bánh xe robot như Hình 7.



Hình 7. Trực quan hóa chuyển động robot bằng TF.

Để đẩy các thông số robot_description lên máy chủ thì tại file control.launch đã có node thực hiện tác vụ này. Hình 8 là node thể hiện trạng thái robot trong file control.launch.

```
<!-- Load robot description for gazebo -->
<include file="$(find smb_description)/launch/load.launch">
  <arg name="simulation" value="$(arg simulation)"/>
  <arg name="description_name" value="$(arg robot_description)"/>
  <arg name="description_file" value="$(arg description_file)"/>
  <arg name="wheel_joint_type" value="continuous"/>
  <arg name="robot_namespace" value="$(arg robot_namespace)"/>
</include>

<node name="smb_robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publi
  <param name="publish_frequency" value="50"/>
  <param name="use_tf_static" value="true"/>
  <remap from="robot_description" to="$(arg robot_description)"/>
</node>
```

Hình 8. Node thể hiện trạng thái robot và các thông số đưa lên máy chủ.

Bài 5: Yêu cầu: Thực hiện một service server để cho phép bắt đầu hoặc dừng lại robot.

Khi sử dụng service server, robot sẽ hoạt động với 2 chế độ 1(true) hoặc 0(false). Với chế độ True, robot sẽ bắt đầu di chuyển và ngược lại. Ngoài ra, khi robot đang di

chuyển hoặc đang dừng thì robot cũng sẽ thông báo lên máy chủ. Hình 9 thể hiện file `service_server.cpp` thực hiện nhiệm vụ đã mô tả.

```
1 #include<ros/ros.h>
2 #include<std_srvs/SetBool.h>
3
4 bool cur_rb = false;
5
6 bool changes_state_robot(std_srvs::SetBool::Request &req, std_srvs::SetBool::Response &res){
7     if (req.data == 0){
8         if(cur_rb==0){
9             res.success = 1;
10            res.message = "Robot is already stopped";
11        }
12        else{
13            cur_rb = 0;
14            res.success = 0;
15            res.message = "Stopping robot";
16        }
17    }
18    else{
19        if(cur_rb==1){
20            res.success = 1;
21            res.message = "Robot is already running";
22        }
23        else{
24            cur_rb = 1;
25            res.success = 0;
26            res.message = "Starting robot";
27        }
28    }
29    return true;
30 }
31
32 int main(int argc, char **argv){
33     ros::init(argc, argv, "changes_state_robot_server");
34     ros::NodeHandle srv;
35
36     ros::ServiceServer service = srv.advertiseService("changes_state_robot",changes_state_robot);
37
38     ros::spin();
39
40     return 0;
41 }
```

Hình 9. File `service_server.cpp` thực hiện thay đổi trạng thái robot.

Kết quả trên terminal được thể hiện ở Hình 10. Ban đầu, robot đang đứng yên, tôi ra lệnh robot dừng lại thì nhận được thông báo robot đã đang dừng. Và khi tôi ra lệnh True thì robot bắt đầu di chuyển. Biến `success` thể hiện giá trị so sánh giữa trạng thái hiện tại và trạng thái thay đổi.

```
manh@manh:~/catkin_ws$ rosservice info /changes_state_robot
Node: /changes_state_robot_server
URI: rosrpc://manh:45045
Type: std_srvs/SetBool
Args: data
manh@manh:~/catkin_ws$ rosservice call /changes_state_robot "data: false"
success: True
message: "Robot is already stopped"
manh@manh:~/catkin_ws$ rosservice call /changes_state_robot "data: true"
success: False
message: "Starting robot"
manh@manh:~/catkin_ws$ rosservice call /changes_state_robot "data: false"
success: False
message: "Stopping robot"
manh@manh:~/catkin_ws$
```


Hình 10. Kết quả thực hiện thay đổi trạng thái robot.

Bài 6: Yêu cầu: Chạy môi trường robot và sử dụng service trên để bắt đầu hoặc dừng robot.

Để thực hiện yêu cầu đề bài, từ **Bài 5** tôi sẽ kiểm tra trạng thái robot bằng cách kiểm tra vận tốc tịnh tiến và vận tốc xoay của robot. Từ đó có thể điều khiển robot đứng yên hoặc dừng lại.

Hình 11a và Hình 11b thể hiện file code *service_server.cpp*. Trước hết kiểm tra trạng thái robot từ vận tốc tịnh tiến và vận tốc góc từ topic */cmd_vel*, khi nhận lệnh thì robot sẽ thay đổi vận tốc để thực hiện nhiệm vụ từ câu lệnh.

```
1  #include<ros/ros.h>
2  #include<std_srvs/SetBool.h>
3  #include<geometry_msgs/Twist.h>
4
5  bool cur_rb;
6  ros::Publisher pub;
7  ros::Subscriber sub;
8
9  void callback(const geometry_msgs::Twist msg){
10     if(msg.linear.x == 0 && msg.angular.z == 0){
11         cur_rb = 0;
12     }
13     else{
14         cur_rb = 1;
15     }
16 }
17
18 bool changes_state_robot(std_srvs::SetBool::Request &req, std_srvs::SetBool::Response &res){
19     geometry_msgs::Twist vel;
20     if (req.data == 0){
21         if(cur_rb==0){
22             res.success = 1;
23             res.message = "Robot is already stopped";
24         }
25         else{
26             cur_rb = 0;
27             res.success = 0;
28             res.message = "Stopping robot";
29             vel.linear.x = 0.0;
30             vel.angular.z = 0.0;
31         }
32     }
33     else{
34         if(cur_rb==1){
35             res.success = 1;
36             res.message = "Robot is already running";
37         }
38         else{
39             cur_rb = 1;
40             res.success = 0;
41             res.message = "Starting robot";
42             vel.linear.x = 0.5;
43             vel.angular.z = 0.0;
44         }
45     }
46     pub.publish(vel);
47     return true;
48 }
```

Hình 11a. Các hàm trong file *service_server.cpp*.

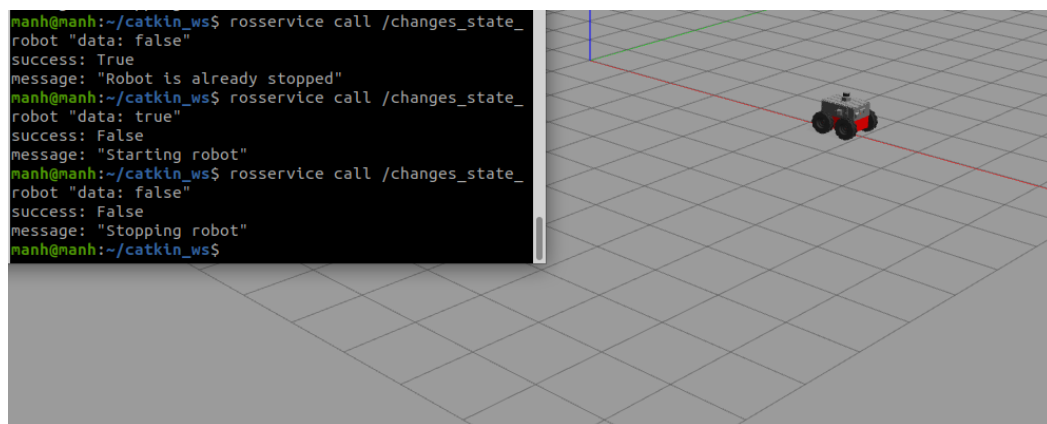
```

50 int main(int argc, char **argv){
51     ros::init(argc, argv, "changes_state_robot_server");
52     ros::NodeHandle srv;
53
54     ros::ServiceServer service = srv.advertiseService("changes_state_robot", changes_state_robot);
55
56     pub = srv.advertise<geometry_msgs::Twist>("cmd_vel", 1000);
57     sub = srv.subscribe("cmd_vel", 1000, callback);
58
59     ros::spin();
60     return 0;
61 }

```

Hình 11b. Hàm main trong service_server.cpp.

Khi robot nhận được lệnh True thì robot đã di chuyển theo đường thẳng và khi nhận được lệnh False, robot đã dừng lại như *Hình 12*.



Hình 12. Mô phỏng robot khi nhận được lệnh từ service.