ELEE 4200

Autonomous Mobile Robots

Homework No:2

Drive Robot with Feedback

Author 1 :
Name :Mi Liang
T.No.:02157118

Author 2:
Name :Wang JIngya
T.No.: 02157119

Instructor:
Dr. Mohan Krishnan

September 29, 2019

Grading Rubric - Information for Students

Grading rubric that will broadly apply to assessing performance. That is, assessment exercises that are associated with a predominantly qualitative rather than quantitative character. If, for a particular exercise, there is substantial deviation from the scheme outlined below, I will let you know! Note: Be aware that if your "Quality of Presentation" is poor, it could impact scores assigned for the other two categories! (DO NOT EDIT THIS PAGE)

		Level of accomplishment					
		0	**1**	**2**	**3**	**4**	**5**
Aspect of effort	**Extent of Completion of Technical Requirements**	Achieved none of the objectives or did not submit	Achieved very few of the objectives	Achieved some of the objectives	Achieved most of the objectives	Achieved almost all the objectives	Demonstrated additional initiative beyond what was required.
	Quality of Analysis & Conclusions	Inadequate	Poor	Below average	Average	Good	Insightful!
	Quality of Presentation	Inadequate	Poor	Below average	Average	Good	Exceptional!

Figure 1: Level of accomplishments

- **Student's overall Level/Score (To be entered by Professor/GTA):**.

Contents

1 Abstract **3**

2 Introduction **3**

3 Methodology **3**

4 Results **5**

5 Discussions and Conclusion **10**

A Appendix **11**
A.1 List of attached files 12

List of Figures

1 Level of accomplishments 1
2 The figure for method 1 interpretation 4
3 The code for the second method 5
4 The code for the third method 5
5 The result of using time to drive robot 6
6 The result of using odometry feedback 6
7 The result of using the "model state" topic as feedback 7
8 The explain of TA's method 7
9 The explain of TA's method 8
10 The explain of TA's method 8
11 The sudden change 9
12 The solution part code 10
13 The explain to the problem 11
14 The solution code to the problem 11

List of Tables

1 Abstract

The Homework2 is the advanced version of Homework1, which we need to drive the robot by using feedback to guide it. And the path is a triangular path which is an isosceles triangle with angles of $30°$**,**$30°$ **and** $120°$ **with the equal sides being 5 meters each in length.**

2 Introduction

- The final goal is to plot a figure which seems like an isosceles triangle with angles of $30°$,$30°$ and $120°$ with the equal sides being 5 meters each in length.

  The specific requirements are:
  Use three strategies to draw the figures:
  1. Identify the linear velocity and the angular velocity, control the time of each line segments, and let the final result come put as an isosceles triangular.
  2. Driving the robot by using the odometry feedback, what's more, we cannot control the time to swerve at the corner but to use feedback to control.
  3. Driving the robot by using the model state feedback, what's more, we cannot control the time to swerve at the corner but to use feedback to control.

  After the three program, we also are asked to do other two tasks:
  1.Search for the command of reading the programming environment time;
  2.Compare the results of each method, and find the characters for each method.

3 Methodology

In the beginning of the code, we should initialize the ROS, create a publisher to the topic, and create the messages with the right format to the topic. In this time, we should also create a subscriber to the topic for the second and the third method.

And this time, we learned how to reset the position of the robot(the specific codes is in the appendix.)

The logic of the first method:

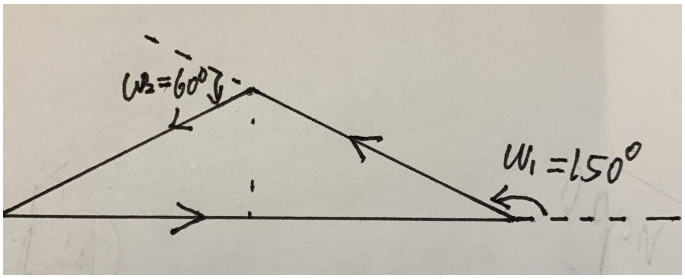

Figure 2: The figure for method 1 interpretation

Firstly, we should define the linear velocity and the angular velocity. Secondly, according to the follow knowledge background is that:

$$\delta s = v \times \delta t$$

According to the formula, because we have already know the length of three line segments, so that we can calculate the $\delta t = \delta s/v$;Then define the t for each line segments.

Thirdly, at the corner, because we have already define the angular velocity, we can also calculate the δt in order to make the robot turning as the required angles. The formula using is

$$\delta\theta = \omega \times \delta t$$

The logic of the second method:

Firstly, receive the initial position of the robot, and save it in a variable. Then receive each step's position information and receive its x axis position, y axis position and z axis angular velocity in three array.

Secondly, use the formula which can calculate the distance between two points $P1P2 = \sqrt{}((x1 - x2)^2 + (y1 - y2)^2)$. and using the receiving information to calculate how far the each step's position point from the initial points. And judging the distance each step, if the distance fit with the requirements 5 meters. Stop the robot.

Thirdly, when we finish the first line segment, it is time to turn an

angle and let the result output as 30°. The idea is to use the function taught by TA.

function quat2eul([W X Y Z]), which is to change the angle.
But because the initial angle we don't know,we use the same method as the line segment to solve the problem; we give the initial angle to the new variable and make the subtraction between each step's angle and the initial angle. And when the output equal to the value what we want, stop at this direction.
Fourthly, combine the three line segments and three angle change in order to the final code.

The logic of the third method: The basic logic is same as the second method, however, it has a big distinction between the two methods. The CITATION FORMAT in the program is different. Calling the information in the second method, we can directly use ".Pose.Pose.Position." as the HW1. However, the calling commands in the method three is ".Pose(2,1).Position". You can see the more details information in the following figures

```
velmsg.Linear.X = 0;
velmsg.Angular.Z =0;
```

Figure 3: The code for the second method

```
robot_state_begain_x = model_states.Pose(7,1).Position.X;
robot_state_begain_y = model_states.Pose(7,1).Position.Y;
```

Figure 4: The code for the third method

4 Results

The result of the first method:
Because the first method is using time to control the robot, the control precision is not as good as the other two method. Additionally, the

robot can not run in a straight line because of the inertia.

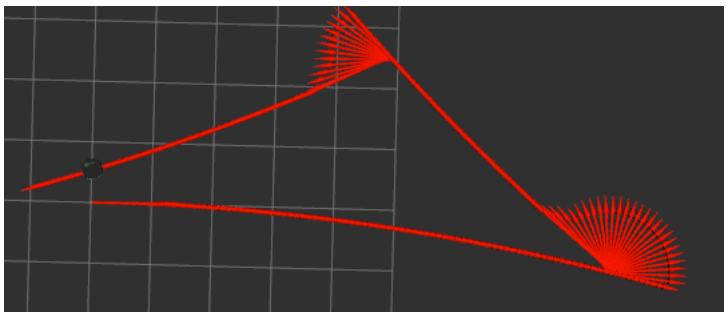

Figure 5: The result of using time to drive robot

The result of the second method:
The result is completely same as our prediction through the adjustments during the running process by using odometry feedback.

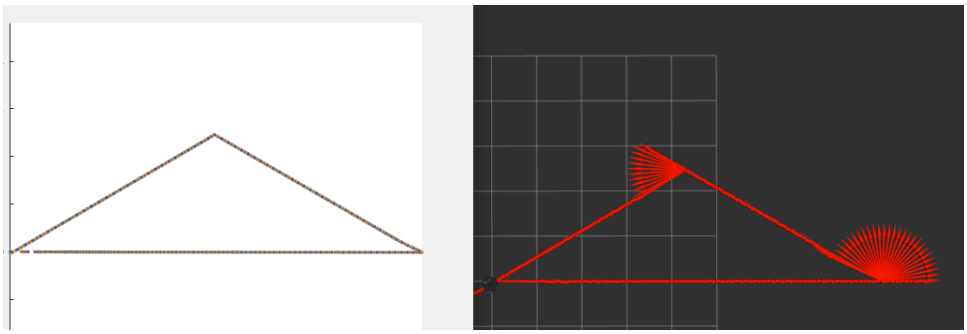

Figure 6: The result of using odometry feedback

The result of the third method:
The result is completely same as our prediction through the adjustments during the running process by using 'model state" topic feedback.
The specified adjustment process:
As we all know, the robot can not run as a straight line during the process because of the inertia. So, if we want to improve the figures and make the line to become straight, the TA told us a method is that, comparing every line segment's direction with the initial direction which we command the robot to go. For example, as the figure 8

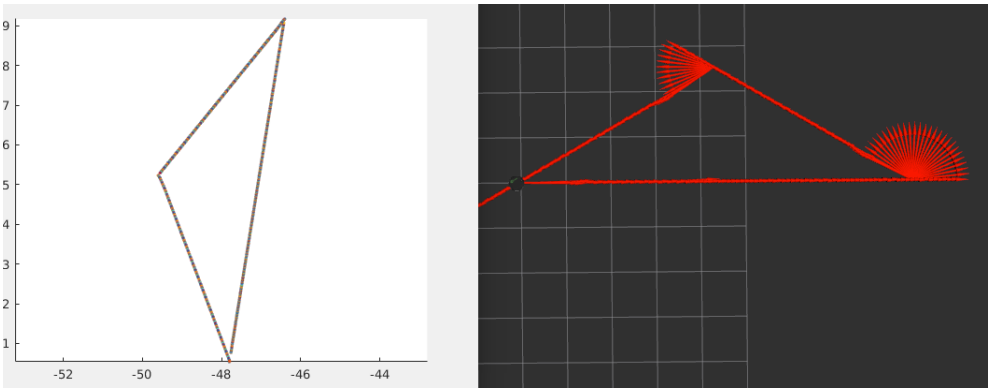

Figure 7: The result of using the "model state" topic as feedback

and figure 9 shown below, if the initial direction is A, and the next line segment's direction is B. If B is lower than A, we need to give the robot a positive angular value, which is to let it rotate counterclockwise. If B is higher than A, we need to give the robot a negative angular value, which is to let it rotate clockwise.

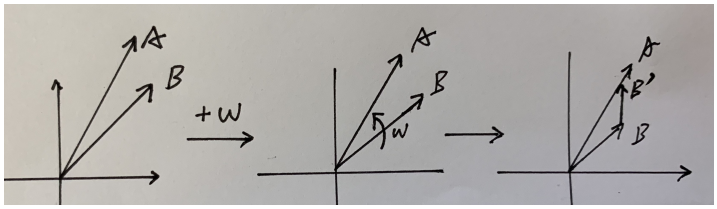

Figure 8: The explain of TA's method

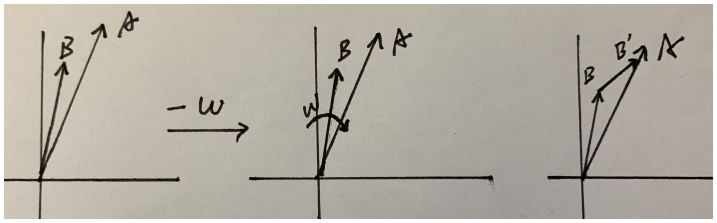

Figure 9: The explain of TA's method

So, the initial code is like the following:

```
W=model.Pose(2,1).Orientation.W;
X=model.Pose(2,1).Orientation.X;
Y=model.Pose(2,1).Orientation.Y;
Z=model.Pose(2,1).Orientation.Z;
angle=quat2eul([W X Y Z]);
if angle(1)==angle0(1)
       velmsg.Linear.X=0.8;
       velmsg.Angular.Z=0;
       send(robot,velmsg);
       x=[x,model.Pose(2,1).Position.X];
       y=[y,model.Pose(2,1).Position.Y];
model=receive(model_subs);
elseif angle(1)<angle0(1)
       velmsg.Linear.X=0.2;
       velmsg.Angular.Z=0.1;
       send(robot,velmsg);
       x=[x,model.Pose(2,1).Position.X];
       y=[y,model.Pose(2,1).Position.Y];
model=receive(model_subs);
elseif angle(1)>angle0(1)
       velmsg.Linear.X=0.2;
       velmsg.Angular.Z=-0.1;
       send(robot,velmsg);
       x=[x,model.Pose(2,1).Position.X];
       y=[y,model.Pose(2,1).Position.Y];
model=receive(model_subs);
end
end
```

Figure 10: The explain of TA's method

However, there is a bug at the moment which the angular value turns from positive to negative or turns from negative to positive.
For example, as the following figure 11 shown below:

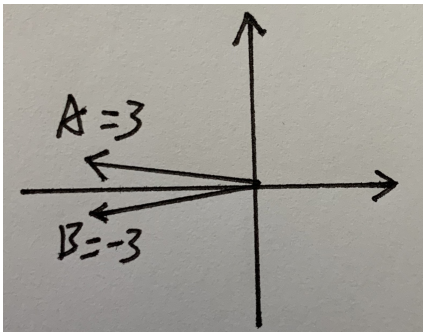

Figure 11: The sudden change

A direction = +3,B direction = -3; the robot judge the situation as A <B ; and give the robot positive value and continue add it until B = +3, so it will lead the robot rotate at the point.
Our solution is to separate such situations from the whole judgements and set the codes like the following figure.

```
if angle_begin==angle_change
velmsg.Linear.X=0.3;
velmsg.Angular.Z=0;
else
    if (angle_begin(1)>=0)&&(angle_change(1)>=0)
        if angle_begin(1) > angle_change(1)
             velmsg.Linear.X=0.3;
             velmsg.Angular.Z=0.05;
        else
             velmsg.Linear.X=0.3;
             velmsg.Angular.Z=-0.05;
        end
    elseif(angle_begin(1)>=0)&&(angle_change(1)<=0)
             velmsg.Linear.X=0.3;
             velmsg.Angular.Z=-0.05;
    elseif(angle_begin(1)<=0)&&(angle_change(1)>=0)
             velmsg.Linear.X=0.3;
             velmsg.Angular.Z=-0.05;
    elseif(angle_begin(1)<=0)&&(angle_change(1)<=0)
        if angle_begin(1) > angle_change(1)
             velmsg.Linear.X=0.3;
             velmsg.Angular.Z=0.05;
        else
             velmsg.Linear.X=0.3;
             velmsg.Angular.Z=-0.05;
        end
     end
end
```

Figure 12: The solution part code

5 Discussions and Conclusion

1.About how to get reading of time in the programming environment

The command we should use is $rostime('now')$: for example, in our programm:

s1 = rostime('now')

{*programmcode*; }

s2 = rostime('now') t = s2-s1;

2.The problem occur in the process:

Because the bug which mentioned in the results part we didn't take into account, the other problem happens:

Let's take an example to explain:

Look at the following figure 13:

At the turning corner, if we define the turning condition like A-B≥C (C is the define angle like 150°), such as we define the C = 5 rad;

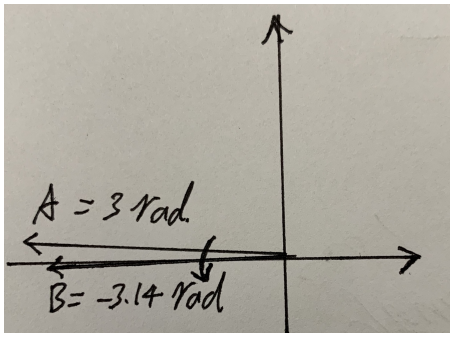

Figure 13: The explain to the problem

If A = 3 rad,and when the robot crosses the x axis which is the condition that B suddenly changed into 3.14. At the moment,B = -3.14 rad; A-B = 6 rad C = 5 rad;the condition we need to break out. However, the real turning angle is just 0.14 rad, the condition judged result is 6 rad. Obviously, the result is wrong.
The solution code is like these:

```
145 -       if (angle_begin(1)>=0)&&(angle_change(1)>=0)
146 -           angle=abs(angle_begin(1)-angle_change(1));
147 -       elseif(angle_begin(1)>=0)&&(angle_change(1)<=0)
148 -           angle=(3.14-angle_begin(1))+(3.14+angle_change(1));
149 -       elseif(angle_begin(1)<=0)&&(angle_change(1)>=0)
150 -           angle=-angle_begin(1)+angle_change(1);
151 -       elseif(angle_begin(1)<=0)&&(angle_change(1)<=0)
152 -           angle=angle_change(1)-angle_begin(1);
```

Figure 14: The solution code to the problem

References

A Appendix

The following are the matlab files.
The order is like the following(the code page is in the behind pages):

A.1 List of attached files

1. MATLAB Method1
2. MATLAB Method2
3. MATLAB Method3