Analysis and Implementation of Proportional Controller Greedy Approach for Wheeled Robot Motion Planner in A Simulated Environment Using ROS (Robot Operating System) and Gazebo Simulator

Farrel Ahmad - 13520110
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520110@std.stei.itb.ac.id

***Abstract*—Background : Movement in robotics is important basic ability to move something to certain setpoint. To reach the setpoint as quick as possible and as accurate as possible, a control system algorithm is needed to provide the steps needed.**

Methods : The Proportional Controller is a control system method derived from PID Controller but without I and D controller. The Proportional Controller algorithm is based on greedy approach. ROS will the main program and where the algorithm is implemented. The ROS program will then be connected to Gazebo Simulator to simulate the ROS program.

Result : The proportional controller is able to make the robot move from start position to goal position. However, Kp constant of linear velocity and Kp constant of angular velocity must be adjusted such that the robot will not overshoot or lost control during the process of getting towards goal position.

Conclusion : From the result, the proportional controller algorithm has proven its ability to reach its setpoint from start position.

Keywords—Motion Planning, Control System, PID Controller

I. BACKGROUND

Movement is one of the most basic abilities in robotics. The robot's movement is used to finish certain task. Such task as simple as moving forward by using robot's foot or by using wheel. There are two main components of robot's movement, path planner and motion planner. Path planner generates positions needed to reach the final goal position while the motion planner generates what linear and angular speed needed for the robot to move towards the goal position. For example, if the robot wants to move from (0,0) to (1,1) and then to (1,2). The robot must know what speed needed from (0,0) to (1,1) and then finally to (1,2). The speed of the robot must be controlled precisely because the robot's movement must be quick and accurate without any overshoot and undershoot. Overshoot is a term when the current state has passed the setpoint. Undershoot is a term when the current state has not passed the setpoint. For example, the robot wants to move from x = 0.0 m to x = 0.15 m. If the robot stopped at x = 0.17 m then this is overshoot. If the robot stopped at x = 0.13 m then this is undershoot.

This is what the proportional controller used for. In this case, the proportional controller implemented in motion planner will execute the path by generating the speed needed step by step for the robot towards each position command from path planner. Proportional controller is based on greedy approach of minimizing position error by moving towards the goal.

Programming domain in robotics is divided into two domains. These are high-level implementation and low-level implementation. The high-level mostly is non-hardware related while the low-level mostly is hardware related. Each has its own purpose has they work in parallel

The high-level implementation usually consists of localization (the robot's implementation of the surrounding environment), perception/vision (real-time image processing for localization), and navigation. The low-level implementation usually consists of hardware related such as controlling motor's PWM (Pulse Width Modulation) or receiving video/continuous real-time image from camera.

The term navigation in this paper covers the process of path planning and motion planning. The path planner gives position command (generated either using A*, Dijkstra, etc.), and the motion planning convert the position command to the robot velocity command.

The velocity command will then receive by the low-level implementation software. For example, if the robot has two wheels, the software must convert the velocity command into wheel velocity. Usually in microcontroller, the wheel velocity is then converted into PWM needed to move the motor to certain speed.

Proportional Controller is common for motion planning because it is derived from PID Controller. Other common motion planning method are trapezoidal profile motion planner, S curve profile motion planner. For trapezoidal profile, as the name suggests, the motion profile is to accelerate constantly to certain speed and decelerate constantly to stop. S curve is similar to trapezoidal motion profile but constant jerk (m/s^3) instead of constant acceleration. Proportional controller is a little bit different, it is not based on constant acceleration or constant jerk, but based on current error value between setpoint and goal, in this case is error between goal position and current position.

II. METHODS

A. Greedy Algorithm Theory

Greedy Algorithm is an algorithm approach to find certain solution step by step such that it will find the current best local optimum in hope of global optimum [1]. For every step, find the best solution in current state.

There are 5 elements of greedy algorithm:

1. Candidate Set : a set of candidates that will be chosen for every step.
2. Solution Set : a set of candidates that has been chosen.
3. Solution Function : a function that checks whether the candidate that has been chosen gives a solution.
4. Selection Function : a function that chooses a candidate based on greedy strategy.
5. Feasibility Function : a function that checks whether the candidate chosen is feasible to be inserted into solution set.
6. Objective Function : maximizing or minimizing function.

The main algorithm is as follows :

1. While solution set is not found, find solution from candidate set.
2. Choose a candidate from candidate set using selection function and objective function
3. Check whether candidate is feasible using feasibility function
4. If feasible then insert to solution set.
5. if not feasible then find another candidate.
6. Repeat until solution set is found.
7. When solution set is found, use solution function to check whether solution set is valid.
8. If valid, then solution is found.
9. If not valid, then solution is not found.

B. Proportional Controller Algorithm

Proportional controller is a control system method derived from PID Controller. PID stands for Proportional Integral Derivative. The goal of PID Controller is to reach certain setpoints, such speed, and position. In this paper, the setpoint is position. Given an input of position command, the algorithm will process error of current position and position command and then output the speed needed for current state.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$

Fig.1 **PID Controller Formula**
Source:
https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID#22

PID has three constants, these are Kp, Ki, and Kd. These three constants are something tunable. The Kp, Ki, and Kd can be changed to get the best output without any overshoot and undershoot. The formula can be translated to the algorithm called Discrete PID algorithm. The algorithm can be approximated using this pseudocode.

```
previous_error = 0
integral = 0
start:
  error = setpoint - measured_value
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto start
```

Fig.2 **Discrete PID Algorithm using Approximation**
Source:
https://shikinzhang.github.io/2016/07/25/PD-Controller/

Fig.2 PID Algorithm shows the approximation of the formula in Fig.1. The dt is delta time, in this case is the robot's compute period. Robot processing time is usually fixed using compute period. For example, if the compute period of the robot is 20 milliseconds, each iteration happens in 20 milliseconds. The algorithm explanation in Fig.2 is as follows:

1. Compute error = (goal – setpoint)
2. Compute integral using approximation
3. Compute derivative using approximation
4. Output is Kp * error + Ki * integral + Kd * derivative

In Fig.1, if the Ki and Kd is set to 0, the controller still works fine, and this is where Proportional Controller or P controller comes from. P Controller is PID without I and D. However, in theory, Integral Controller is used to eliminate steady state error and Derivative Controller is used to minimize overshoot and undershoot. Without I and D, the controller will have some minor issue with having steady state error or over/undershoot under certain conditions.

The P Controller pseudocode will be as follows:

```
start:
    error = setpoint – current
    output = Kp * error
```

For example, the robot must move from x = 0 m to x = 0.1 m with Kp = 1, compute period 20 ms, output as speed. Assuming the displacement is calculated approximately using compute period. For this example, the pseudocode is as follows:

```
start:
    current =
          current +
          (output * compute_period)
    error = setpoint – current
    output = Kp * error
    goto start
```

The iteration will be as follows:

1. Kp = 1
   curr_x = 0.0 m
   goal_x = 0.1 m
   output = 1 * (0.1 – 0.0) = 0.1 m/s

2. Kp = 1
   curr_x = 0.0 + (0.1 * 0.020)= 0.002 m
   goal_x = 0.1 m
   output = 1 * (0.1 – 0.002) = 0.098 m/s

3. Kp = 1
   curr_x = 0.002 + (0.098 * 0.020) = 0.00396 m
   goal_x = 0.1 m
   output = 1 * (0.1 – 0.00396) = 0.09604 m/s

4. Kp = 1
   curr_x = 0.00396 + (0.09604 * 0.020) = 0.0058808 m
   goal_x = 0.1 m
   output = 1 * (0.1 – 0.0058808 m) = 0.0941192 m/s

5. The iteration will continue and eventually stop because the error is getting smaller resulting the smaller output. When the error is zero, the output will also be zero and the robot will stop and the goal position.

If Kp is set bigger, the output in the first iteration will be bigger and eventually will reach setpoint more quickly. However, in real application, this will result in high overshoot over the setpoint.

The greedy approach of the algorithm is shown by the logic of "increase the speed when error is getting larger because the setpoint is still far away" and "decrease the speed when error is getting smaller because the setpoint is getting close and it needs to slow down." The optimization task of the algorithm is to minimize error between goal position and current position. In details, the greedy elements analysis is as follows:

1. Candidate set : list of any speed command to the robot.
2. Solution set : list of speed command that has been chosen.
3. Solution function : valid if the solution set will make final error <= desired minimum error to stop the robot (for example current error <= 0.01 m).
4. Selection function : Kp * error.
5. Feasibility function : candidate is valid if the command will not make the robot overshoot/undershoot.
6. Objective function : minimizing error of position.

However, in simulator, the error of the robot will be computed using actual position of the robot instead of using approximate displacement with compute period as position's reading.

C. ROS (Robot Operating System)

The ROS (Robot Operating System) is a set of libraries and tools that can help users build robot program mainly in C++ or python or both [2]. Although the name has "Operating System" in it. It is not an operating, but a framework that is commonly used in robotics world.

The program works by having packages and each packages has nodes. These nodes can communicate to each other by publish-subscribe method and server-client (service-request) method. The term node in this paper will be the main program of the robot where the algorithm is implemented.

Whether publish-subscribe or service-client method, both has the address of where the message is sent called rostopic. Rostopic can be imagine as the mailbox for the message where every other node can read the content of the message.

In publish-subscribe method. A node can contain either publisher or subscriber or both. A node that has publisher will send or publish the message to certain rostopics. For example, a velocity controller using P-Controller publishes velocity message to the robot to rostopic /robot1/cmd_vel. The publisher constantly publishes the message every compute period time. For example, if the node rate is 50 Hz, then every 1/50 s = 20 ms the message is published to rostopic /robot1/cmd_vel. The microcontroller can also have a ROS program with node containing subscriber to /robot1/cmd_vel. Every 20 ms the microcontroller will receive new cmd_vel message and then the message will be converted to motor

command that will move the motor and reach the intended velocity.

In service-client method. The workflow is similar to publish-subscribe method. However, service-client method is not constantly publishing the message to rostopics. The client calls once the server with a request then the server will process the request using the callback that has been created and return the result back to the client. The simple example of service-client method in ROS is calling the server in Gazebo Simulator to reset the world to the initial state when the program is first launched.

The program must be compiled using Catkin before use. Catkin is an official build system of ROS that is a CMake with added python script to provide extra functionality for the CMake normal workflow.

D. Gazebo Simulator

Gazebo Simulator is an open-source libraries containing set of development libraries including simulation for robotics development [3]. It has the capability of simulating physics, sensors, and 3D rendering. It can simulate accurate robot movements and simulate as in controlling real robot. During robot development, it is important to use simulator to simulate the performance of the robot first before testing it directly to the robot. Testing it directly in the robot is riskier because if there is a bug, the robot can be in a state of unwanted behavior such as going out of control or anything dangerous. The ROS program that has been made can be connected to Gazebo Simulator to test the code.

Gazebo Simulator is a platform to simulate low-level implementation of the robot. Usually, hardware related commands are sent to robot via microcontroller. The simulator can simulate the microcontroller and simulate the received command from ROS into hardware execution. Gazebo also has a built-in physics engine that the motion or robot movements follow the physics of real life.

E. Robot Specification

The robot that will be used in a simulator is based on real world robot called ROBOTIS TurtleBot3 Burger. This is a small kit robot that is usually used for educational purpose. The robot has a dimension of 13.8 cm x 17.8 cm x 19.2 cm (L x W x H).

III. RESULT

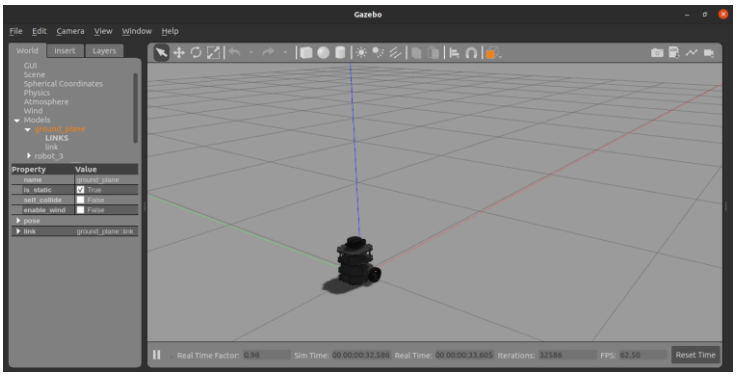


Fig.3 **Gazebo World**

The program is implemented using empty world of gazebo. The empty world contains grid tiles with the size of 1m x 1m. These tiles can be an approximate measurement of position with visible eye. However, Gazebo provide real-time current robot position and this real-time position is used in the computation. The program's compute period is set to 50 Hz. Thus, each iteration happens for 20 ms.

There are two types of proportional controller used in the program. The linear velocity proportional controller and angular velocity proportional controller. Each has its own Kp constant. The velocity proportional controller will control the forward motion of the robot while the angular proportional controller will control the angular motion of the robot.

The command given/published to the robot is in Twist message. Twist is a common format used in ROS for velocity data type. Twist contains two Vector3 message, linear and angular. All linear and angular velocity is in the same Twist message. The linear and angular proportional controller control the value of these linear and angular velocity message. Once the message has been created, it will be published to the robot as a command velocity.

Gazebo simulator will simulate the robot receiving the message. Thus, the message published by the program to the robot's command velocity will be read by Gazebo and Gazebo will simulate the message for example simulate robot's movement from the velocity command.

Linear and angular proportional controller are both tunable. The value of each Kp can be tuned such that it will not make the robot uncontrollable or overshoot/undershoot. For example, if the Kp is too large, the robot will likely suffer overshoot. If the Kp is too small, the robot will likely suffer undershoot or takes a long time to reach setpoint.

```
66      err_lin = sqrt( (pow(err_x,2) + pow(err_y, 2)) );
67      angle_to_goal = atan2(err_y,err_x);
68      err_ort = angle_to_goal - tht;
```

Fig.4 **Error Computation in Source Code**

```
71        spd.linear.x = Kp_lin * err_lin;
72        spd.angular.z = Kp_rot * err_ort;
```

Fig.5 **Proportional Controller Output in Source Code**

```
GX:1.000, GY:1.000, X:1.115, Y:0.716
[ INFO] [1653070210.607639748, 14.701000000]:
GX:1.000, GY:1.000, X:1.115, Y:0.716
[ INFO] [1653070210.626970213, 14.721000000]:
GX:1.000, GY:1.000, X:1.115, Y:0.720
```

Fig. 6 **Terminal Information**

When the program is running. The program will print the current state of the robot to the terminal. GX is goal of X position, GY is goal of Y position, X is current x position, and Y is y position. All the values are in meters.

The program was made using Ubuntu 20.04 LTS, ROS Noetic, and Gazebo 11. To run the program, make sure to use these versions. After compiling the program. The program can be run using this command.

```
$ roslaunch navrobot_gazebo
PConRobot.launch
```

```
$ rosrun navrobot_gazebo robot_3
_x_goal:=1.0 _y_goal:=1.0 _Kp_lin:=0.35
_Kp_rot:=0.45
```

The _x_goal, _y_goal, _Kp_lin, and _Kp_rot values can be changed. _Kp_lin is the Kp constant for linear velocity (linear velocity proportional controller) and _Kp_rot is the Kp constant for angular velocity (angular velocity proportional controller).

In this example, the Kp value is actually from trial and error tuning the robot. For the goal position of (1,1) the optimal _Kp_lin and _Kp_rot is 0.35 and 0.45 respectively. For other goal position the Kp value must be tuned accordingly. However, one should note that the Kp constant must not make the robot lost control or overshoot the setpoint.

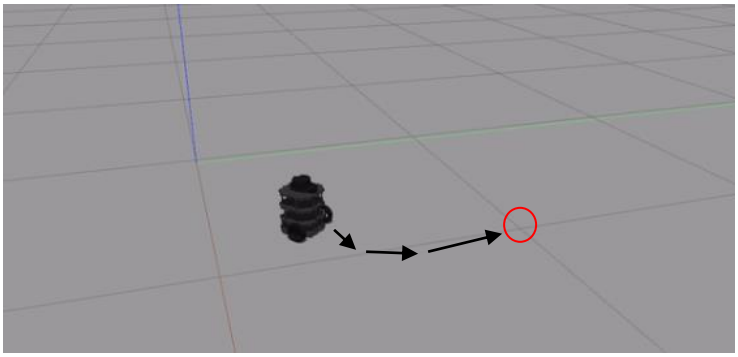


Fig.7 **The Robot Moving Towards Setpoint of (1,1)**

Once the robot reached the destination, the error will become zero and the output will also be zero. Tuning the correct Kp value is the common way of tuning the robot control system for finding the best performance of the robot. In this case, the performance metrics are minimum error position and maximum velocity

The test also was conducted on other setpoints, Kp_lin, Kp_rot, and its result whether able to reach is destination or not. If the robot is able to reach the setpoint without any overshoot then it is categorized as safe, otherwise unsafe. The start is always from x = 0 and y = 0

X goal	Y goal	Kp linear	Kp rotation	Result
0.5	0.5	0.2	0.3	Safe
0.5	0.5	0.5	0.5	Unsafe
1.0	1.0	0.5	0.6	Unsafe
1.0	1.0	0.35	0.45	Safe
1.5	1.5	0.35	0.45	Safe
1.5	1.5	0.4	0.5	Unsafe
2.0	2.0	0.4	0.5	Unsafe
2.0	2.0	0.3	0.4	Safe
2.5	2.5	0.3	0.4	Unsafe
2.5	2.5	0.1	0.2	Safe
3.0	3.0	0.1	0.2	Safe
3.0	3.0	0.3	0.4	Unsafe

Table 1 **Kp Test on Different Setpoints**

The safe Kp for each setpoints is getting smaller as the setpoint is getting larger. This is because when the setpoint is getting larger, the first error will be bigger giving the velocity command higher in the first place. The high velocity command during the start of movement will likely make the robot overshoot and will not be able to recover the intended path. Reducing the Kp value will reduce the first velocity command and also giving more chance for the rotational velocity to maneuver while also having a linear velocity giving the smooth movement of the robot.

The Kp rotation is also set to higher than Kp linear. This is because if Kp linear is higher then the forward motion of the robot is higher making the robot harder to rotate because of the smaller rotational velocity output.

There is also a problem with forward velocity of the robot. The wheel motion is simulated with differential drive plugin such that when the forward velocity command is relatively high the robot will move forward but suddenly rotate out of control. However, this seems to be the bug on the plugin because if another test was done with the same x goal, y goal, Kp linear, and Kp rotational the robot will move fine and smooth.

IV. Conclusion

The Proportional Controller has proven the ability to make a simple ability of moving the robot anywhere in the simulator. However, the lack of Integral and Derivative controller makes the robot easily overshoot its setpoint when the Kp value is not set correctly. Also, for different setpoint the Kp must also be tuned because the output in the beginning of iteration will be different since the starting error is also different.

The relatively high forward velocity makes the robot movements suddenly rotate out of control. This is caused by unknown bug from the built-in differential controller plugin of the robot. Thus, setting the Kp even lower is ideal to minimize the sudden rotation of the robot when the robot is moving forward.

Video Link at Youtube

Part 1 : https://youtu.be/Sfeo-x5UsP0

Part 2 : https://youtu.be/MttLGwroSjo

Github Repository

Link : https://github.com/farrel-a/robot-nav-simulator

References

[1] Munir, Rinaldi. "Algoritma Greedy", https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. Accessed on 21 May 2022.

[2] ROS. "ROS Documentation". http://wiki.ros.org/Documentation. Accessed on 21 May 2022.

[3] Open Robotics. "Gazebo Homepage". https://gazebosim.org/home. Acessed on 21 May 2022.

[4] Ahmad, Farrel. "Application of Dijkstra Algorithm for Robot's Obstacle Avoidance System in A Simulated Environtment Using ROS (Robot Operating System) and Gazebo Simulator." https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Makalah2021/Makalah-Matdis-2021%20(113).pdf. Accessed on 20 May 2022.

[5] Michigan University. "Introduction : PID Controller Design". https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID#22. Accessed on 20 May 2022.

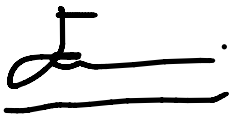
[6] Zhang, Shikin . "PID Pseudocode". https://shikinzhang.github.io/2016/07/25/PD-Controller/. Accessed on 20 May 2022.

[7] ROS.org."catkin/conecptual_overview". http://wiki.ros.org/catkin/conceptual_overview. Accessed on 21 May 2022.

Statement

I hereby declare that my paper is my own writing, not a summary, nor a translation from other's writing, and not a plagiarism.

Bandung, 21 May 2022



Farrel Ahmad - 13520110