Instruction for Assignment 1 for Term Project

Pure Pursuit Algorithm

Introduction

In this assignment, a mobile robot is required to follow or track a given path. So you have to implement a controller which can perform tracking a given path well. For this reason, we introduce a simple control algorithm called "pure pursuit". This algorithm is invented by CMU [1] and is widely used. This algorithm finds a control which makes robot move toward goal smoothly. We will provide skeleton code to you and you have to fill in the TODO part.

Pure Pursuit

Main Algorithm

The pure pursuit approach is a method of geometrically determining the curvature that will drive the vehicle to a chosen path point, termed the look-ahead point. An arc that joins the current robot position and the goal point is constructed. And chord length of this arc is the look-ahead distance. This length acts as the third constraint in determining a unique arc that joins the two points.

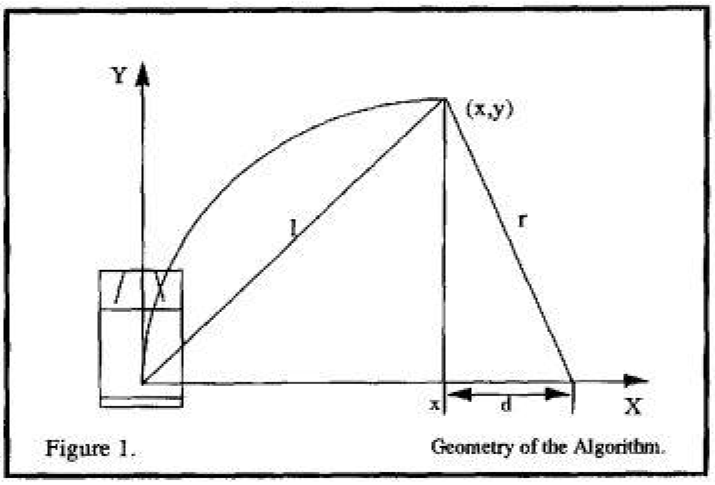

Figure 1

Consider Figure 1. The mobile robot is pictured, with the axes of its coordinate system drawn. The Y axis is mobile robot's heading. And point (x, y) which is one look-ahead distance l from the robot is one of points on the path. The following two equations hold. The first is from the geometry of the smaller right triangle in Figure 1. The second from the summing of line segments on the x axis. In the figure the length of x+d on the X axis and r on the hypotenuse of right triangle seems not be the same. But actually these are the same length r and constitute two sides of pie shape with arc. You can think point (x+d, 0) on the X axis as a pivot point of the arc.

$$x^2 + y^2 = l^2$$

$$x + d = r$$

Equation 1

And the next series of equations relate the curvature of the arc to the look-ahead distance. The algebra is straightforward.

$$d = r - x$$

$$(r - x)^2 + y^2 = r^2$$

$$r^2 - 2rx + x^2 + y^2 = r^2$$

$$2rx = l^2$$

$$r = \frac{l^2}{2x}$$

$$\gamma = \frac{2x}{l^2}$$

Equation 2

Final equation is just a relation between x offset of the goal point from the robot's coordinate and look-ahead distance l. Using final equation, we can calculate curvature of the arc which joins robot's position and the look-ahead point. And this curvature can be used to determine a proportional constant between linear velocity and angular velocity as following equation.

$$w = \gamma v$$

Equation 3

In implementation, the positions of robot and look-ahead point are set according to global coordinate. So, first, you have to transform them from global coordinate frame to relative coordinate frame in which the robot position becomes its origin. After the transformation, you can simply compute linear and angular velocity from the above equation.

Heuristics

Simply implementing pure pursuit algorithm is not sufficient to control mobile robot move smoothly. Sometimes, this algorithm shows poor performance generating large wave like trajectory as Figure 2. This phenomenon occurs when the heading of robot at current points deviates large from the look ahead direction. And also occurs at the point where the path bent abruptly. So we have to add some heuristics to handle this problem.

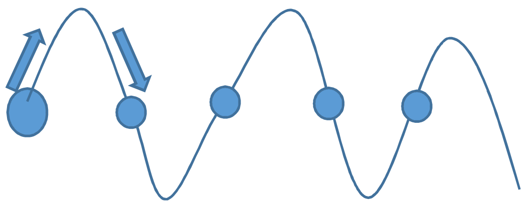

Figure 2

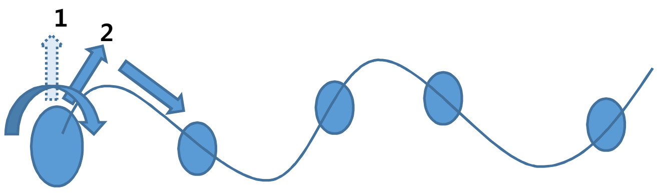

Figure 3

Pure pursuit works inefficiently when the deviation of heading of a robot from the look-ahead direction is too large. In that case, it would be better to stop and turn toward the look-ahead point at a current point. If the angle difference between robot's heading and the look-ahead direction is sufficiently small, then stop turning and start pure pursuit again. This procedure is illustrated in Figure 3. Parameters you have to design are threshold of angle difference above which the robot stops and turn to look-ahead direction and its angular velocity. And also you have to design robot's linear velocity when following the pure pursuit. And pioneer's maximum velocities are 1.2m/s and 300 degree/s. you have to consider these constraints when you design parameters. Whole procedure is summarized in algorithm 1.

Algorithm 1

```
1. Compute the angle difference between robot's heading and look-ahead point direction.
2. If the angle difference is larger than threshold1,
3.      Turn fast toward look-ahead point direction.
4. If the angle difference is larger than threshold2 (< threshold1),
5.      Turn slowly toward look-ahead point direction
6. Otherwise, do pure pursuit.
```

This is a simple example and you can modify the number of threshold level.

Skeleton Code

- ✓ Control.h
- ✓ Point.h
- ✓ purePursuit.h & purePursuit.cpp
- ✓ ppmain.cpp

You can download skeleton code from the course webpage. And extract it into "catkin_ws/src/" as previous assignment. This project files are composed of three header files and two cpp files. Two header files just indicate data type. One cpp file is pure pursuit class. And main cpp is controller for pioneer.

Control.h & Point.h

Control.h is a header file containing "control" structure or "control" data type. It contains two variables v and w. Variable v and w means linear and angular velocity respectively. You have to set v and w velocities computed from the algebra of pure pursuit. Point.h contains structure "point" or "point" data type. It contains three variables x, y and th. These variables represent robot's position and heading respectively. (x, y) is the position from the origin according to global frame. Variable th is angle of heading from x axis in radian.

```
struct control{
        double v;
        double w;
};

struct point{
        double x;
        double y;
        double th;
};
```

PurePursuit.h & PurePursuit.cpp

Purepursuit class has private member variable “ctrl” whose data type is “control” and public member function “get_control(point x_robot, point x_goal)”. The function ”get_control(point x_robot, point x_goal)” returns control “ctrl” as a result of pure pursuit algorithm. You have to fill in the TODO part in the Purepursuit.cpp file. When you are to use this library in the main loop (ppmain.cpp), make a purepursuit instance and call get_control member function to get control. Finally, publish them to GAZEBO simulator. Usage of this class is explained in next section.

Ppmain.cpp

It contains publishers visualizing path and get the robot position from GAZEBO topics. So you just need to implement publisher of robot’s velocity. This publisher is exactly the same as in the previous assignment. But this time, you need to add ROS spin function and sleep process for some duration. Whole structure you have to implement in TODO part in ppmain.cpp is as follows.

1. Make control input using pure pursuit class.
2. Publish control to pioneer. Use predefined publisher, "cmd_vel_pub" and use predefined variable, "cmd_vel".
3. Check whether the pioneer reach a currently following way point or not. Calculate distance between current position of robot and currently following way point (look-ahead point). If the distance is less than 0.2m (certain threshold, you can change this value), pursue next way point (look-ahead point).
4. Check whether the pioneer reach final way point (end of the path). If so, terminate controller.

All instances for publishing control to pioneer in GAZEBO is already defined and you can use it by calling it to control the pioneer. “cmd_vel_pub” is the publisher to publish velocity to GAZEBO. “cmd_vel” is a variable meaning velocity. Path will be given as vector type variable “path”. All the way points are stored in this variable “path”. Template type of the vector is “point” (std::vector<point>). The pioneer have to sequentially track the given path and the controller should be terminated when the robot arrives the final point of the path.

Assignment

You have to implement a code that a robot smoothly tracks a given path. You can check your code by simulating the controller using following commands.

```
roslaunch project1 project1.launch

rosrun project1 project1
```

If you type “roslaunch project1 project1.launch”, you will see GAZEBO simulator with pioneer. And if you run your rosnode, the blue balls appears in the air above each way point. Your pioneer will follow the blue balls.

The pure pursuit controller will be used in the next assignment. Therefore, it will be helpful to design a good and neat pure pursuit controller.

Submission Format

Compress your project folder which includes all your project files and upload it on eTL. The name of compressed file should be “**IS_Project_01_[TeamName].tar.gz**”.

Reference

[1] Coulter, R. Craig. *Implementation of the pure pursuit path tracking algorithm*. No. CMU-RI-TR-92-01. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1992.