Implementation of the Pure Pursuit Path Tracking Algorithm

Havish Netla

December 29, 2019

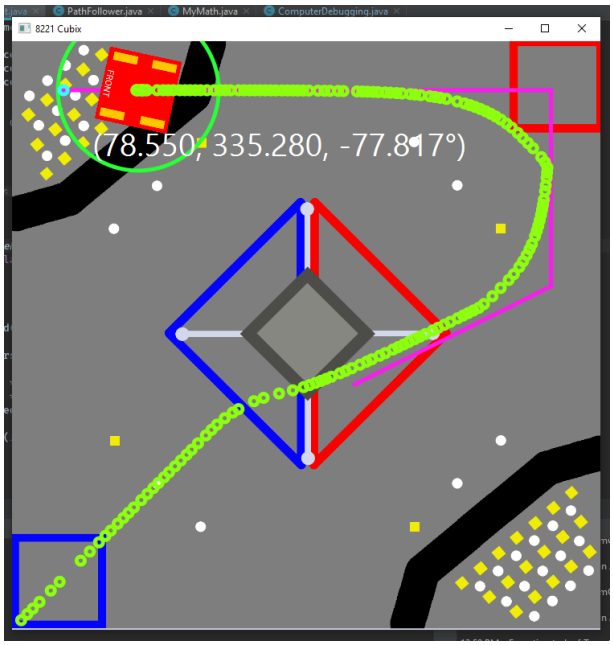
1 Introduction

Pure Pursuit is a path following algorithm used in many robotics applications, mainly used in non-holonomic machines such as cars, when implement properly can be benfecial to all robots due to most holonomic robots effeciency in certain positions.

In a regular path following algorithm, the robot moves to the specified point, and then the next creating a rigid robot-like path. With Pure Pursuit the robot moves towards a lookahead point which is found by finding the intersections of a circle centered around the robot with the robots path. The robot moves towards this point and every update the point is re-calculated, this cycle creates a smooth fluid path that turbs
In the field of robotics, most holonomic robots utilize mecanum wheels, a wheel with a 45°force vector that allows the robot to move in any direction. Mathematically a robot using mecanum wheels should be able to move in any direction with the same effeciency as a regular robot, but because of friction a robot utilizing mecanum wheels will be most effecient in 2 directions, for most robots this is the forward and backwards direction.

Pure Purusit, adpated for holonomic robots maintains the robots most effecient motion directions to optimize the time taken for a single path. Other algorithms that over use strafing are inferior because strafing is slower, less accurate and uses more energy compared to forward and backward.

In the scope of FIRST Tech Challenge, Pure Pursuit is usefull because of its adaptability, requiring only waypoints to be altered to change the path, and its versatility allowing for complex spline like paths and extreme customazability.



This picture depicts a robot simulator running our implementation of the pure pursuit algorithm, the pink path is the original path inputed by the programmer and the green path is the actual path the robot took, the robot automaticly "cuts the corners" to create a smooth fluid path.

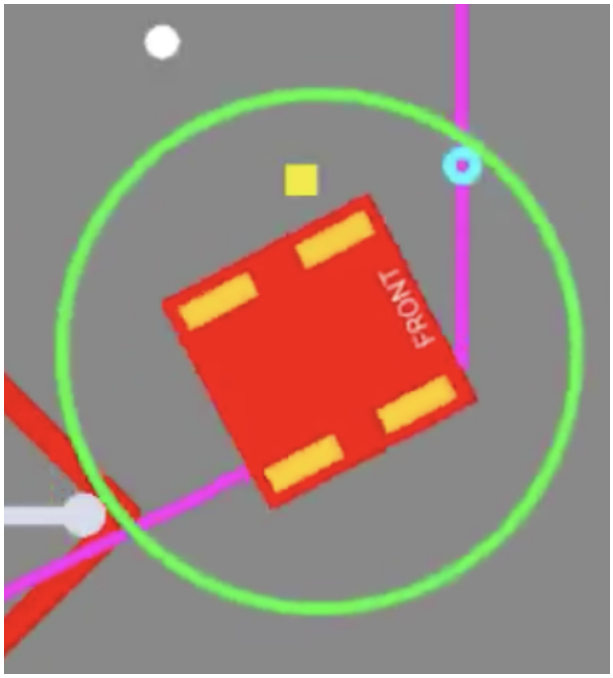
For Pure Pursuit to be implemented Three Wheel Odometry is required, for more information please view my paper titled "Three Wheel Localization of a Holonomic Robot"

2 Derivation

Derivation for Pure Pursuit has two steps, while there are only two steps, each step contains many substeps which contribute to the complexity of Pure Pursuit.

2.1 Look Ahead Point

The first step in Pure Pursuit is finding the lookahead point. To do this we create a circle around the robot with a radius specified by the programmer. then we find all intersections between the path and the circle, because of the nature of a circle and a line there can be 2 even multiple intersections between the look ahead circle and the path.



This image shows the lookahead point, the intersection of the circle with the path (the pink lines)

To find all the intersections, we loop through every path segment and find all intersections.

Defining two points (x_1, y_1) and (x_2, y_2) as the line segment we can further define

$$d_x = x_2 - x_1$$
$$d_y = y_2 - y_1$$

$$d_r = \sqrt{d_x^2 + d_y^2}$$

$$D = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1$$

gives the points of intersection as

$$x = \frac{D d_y \pm sgn(d_y) d_x \sqrt{r^2 d_r^2 - D^2}}{d_r^2}$$

$$y = \frac{-D d_x \pm |d_y| \sqrt{r^2 d_r^2 - D^2}}{d_r^2}$$

where the function $sgn(x)$ is defined as

$$sgn(x) = \begin{cases} -1 & \text{for } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

The discriminant

$$\Delta \equiv r^2 d_r^2 - D^2$$

applying this equation on every path segment with the lookahead circle gives us a list of all the possible lookahead points.

Now with all the possible lookahead points we need to find which point has the closest angle towards the point.

If we take the robot position and create a right triangle relative one of the lookeahead point we can find the angle to the point.

Defining the pose as (x_1, x_2) and the intersection as (y_1, y_2) we can apply

$$\theta = \tan^{-1}(y_2 - x_2, y_1 - x_1)$$

2.2 Holonomic move towards point

Our move to point function is special, it does not just find the Δx and Δy and apply a PID controller to it, we need to be able to make the robot move towards that point with an angle, that angle is stricly relative to the path, for example a path where the robots position is $(0, 0)$ and the goal is$(100, 100)$

with a goal angle of 0°would end up facing 45°in field coordinates. The 0°means relative to the path which direction we are going to face, so if we had put 90°as the we would end up facing 90°perpendicular to the path.

To acomplish this we need to first find the distance between the robots pose to the goal point which in this case is the lookahead point. If we define the robots pose as (x_1, x_2) and the goal point as (y_1, y_2) we can use the distance formula to find the

$$dist = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2}$$

Next we need to find the "absolute angle to target" which means the angle of the position to the goal. Defining the pose as (x_1, x_2) and the intersection as (y_1, y_2) we can apply

$$\theta = \tan^{-1}(y_2 - x_2, y_1 - x_1)$$

Now to find the "relative angle to point" or the amount the robot needs to turn to line up with the path, this is what allows us to "face towards the point" and allows us to be able to move in a path that feels like one fluid path when in reality it is the robot following a constantly changing path. Finding the relative angle is simple. If we define the robots heading as c we can apply

$$relAngle = c - absoluteAngleToTarget$$

Next we need to find the relativeX and relativeY so that the robot can move towards the point we can utilize sin and cos and we can apply

$$relX = sin(relAngle) \cdot dist$$

$$relY = cos(relAngle) \cdot dist$$

These powers are applied in a sinusoidal fashion which makes the powers scaled to the distance which creates a gracefull smooth motion to the point.

2.3 Implementation

There are a lot of little things that need to be added to implement Pure Pursuit in the real world. One of the things is that we need to implement a effecient way. What we need is to create a Path Builder class, where we can

create an ArrayList of an object that contains the goal, followAngle, speed, and turnSpeed.

To change those values for every segment of the path we need to create a check that checks if the lookAheadPoint on a certain segment of the path and then it will adjust the values accordingly.

One of the biggest problem with Pure Pursuit is ending, the robot will have trouble stopping, once the robot reaches the end of the path there is only one intersection with the path and therefore the lookahead point will be behing the robot. To combat this issue we can artificially extend the path foward so that there will be a lookahead point

We also need to check if the robot has finished its path and then move on to the next instruction. We can do this by returning once the power is less than 0.2.