Geometric Control

Alexander Koldy

Forest Hills Robotics League

alexanderkoldy.ak@gmail.com

September 24, 2024

Introduction

In this lecture, we will discuss geometric control in the context of the *FIRST Tech Challenge* (FTC).

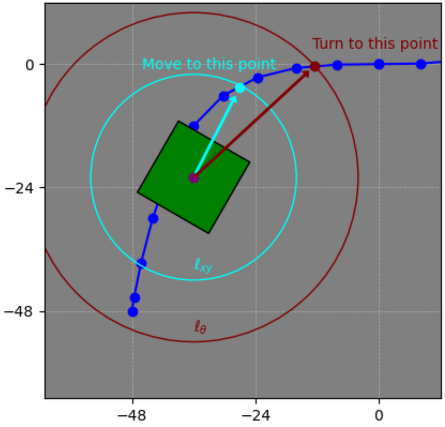
The original Pure Pursuit implementation can be found in this paper and is meant for non-holonomic drivetrains (e.g., differential drive). We will use a modified (slightly easier) version of this algorithm, as most FTC robots have holonomic drivetrains.

Holonomic Pure Pursuit

The Holonomic Pure Pursuit algorithm can be defined by a few simple steps:

1. Define line segments between each waypoint on the path.
2. Define a circle of radius ℓ around the center of the robot.
3. Find the furthest intersection between the circle and the path (made up of line segments).
4. Drive and turn the robot towards that point.

Steps 2-4 are repeated at each loop iteration. Moreover, we can use a second (larger) circle to determine the point to turn to.



Holonomic Pure Pursuit - intersection calculation

Since we have n waypoints in our path, we will have $n-1$ line segments. Let us write the algorithm for calculating the intersection between ith line segment and the lookahead circle. We define the waypoint p_i as the start-point of the line segment and waypoint p_{i+1} as the end-point of the line segment. The robot's position is defined as (x, y) from the state vector. The parameterization of the line segment between the two waypoints is

$$\begin{aligned} p_x(t) &= p_{x,i} + (p_{x,i+1} - p_{x,i})t \\ p_y(t) &= p_{y,i} + (p_{y,i+1} - p_{y,i})t \end{aligned}$$

where t is a parameterization variable (not time). The equation for the lookahead circle is

$$(c_x(t) - x))^2 + (c_y(t) - y)^2 = \ell^2$$

To check for intersections, we set $p(t) = c(t)$. We now have

$$(p_{x,i} + (p_{x,i+1} - p_{x,i})t - x))^2 + (p_{y,i} + (p_{y,i+1} - p_{y,i})t - y)^2 = \ell^2$$

Our goal now is to solve for t!

Holonomic Pure Pursuit - intersection calculation

The full walkthrough of the variable rearrangement can be found here. Essentially, we place the previous expression into quadratic form (in terms of t), where

$$\begin{aligned}
A &= (p_{x,i+1} - p_{x,i})^2 + (p_{y,i+1} - p_{y,i})^2 \\
B &= 2\big((p_{x,i} - x)(p_{x,i+1} - p_{x,i}) + (p_{y,i} - y)(p_{y,i+1} - p_{y,i})\big) \\
C &= p_{x,i}^2 - 2xp_{x,i} + x^2 + p_{y,i}^2 - 2yp_{y,i} + y^2 - \ell^2
\end{aligned}$$

Before solving for t, we check the discriminant value of $B^2 - 4AC$. If this value is negative, it means no intersection exists. Therefore, we should move onto the next line segment. We now use the quadratic formula to solve for t:

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

This yields two values for t. The parameterization range for t is $[0, 1]$ for the line segment. Therefore, any t values outside of this range are discarded.

Holonomic Pure Pursuit - intersection calculation

Assuming there are valid t values remaining, we simply use the line segment parameterization equations to calculate the intersection point(s) using these values.

We run this intersection algorithm on each line segment and collect a list of intersection points. In Java a `LinkedHashSet` can be used as it automatically removed duplicates (once again, see here).

Finally, we take the intersection furthest along the path (final point in the list). This is done at each iteration of the main robot loop to continuously calculate a new lookahead to move towards. Let's call the x and y positions of this point x_{des} and y_{des}, respectively.

To avoid doing this calculation for every line segment at each loop iteration, we can store the index of the last intersection and start from that value in the subsequent loop. This because the lookahead must move forward along the path.

Holonomic Pure Pursuit - movement calculations

Let's define our point-to-turn-to as $p_{des,\theta}$. If the heading and position lookahead circles are equivalent, then $p_{des,\theta} = p_{des}$. Our desired heading becomes the angle between the field frame's x-axis and vector from the robot's position and $p_{des,\theta}$:

$$\theta_{des} = \text{arctan2}\big((y_{des,\theta} - y), (x_{des,\theta} - x)\big)$$

In Java, we can use the `Math.atan2` function. Now, we have a full desired pose vector, i.e.,

$$q_{des} = \begin{bmatrix} x_{des} \\ y_{des} \\ \theta_{des} \end{bmatrix}$$

We can send this vector (as well as our robot's current pose) to the pose controller to calculate a wheel speed vector necessary to force the robot towards this pose!

However, we have a potential problem. If our lookahead distance is too small, the lookahead points might be very close, causing our pose controller to output very low wheel speeds.

Holonomic Pure Pursuit - scaling movement

To counteract this problem, we can set new gains for our pose controller (to yield larger velocities and therefore wheel speeds) or we can simply scale the output wheel speed vector. Let's try the latter solution: we define a maximum wheel speed $\omega_{w,max}$ that we will scale our wheel speed vector to.

We will scale our wheel speed vector based off of the highest wheel speed given by our pose controller:

$$\lambda_w = \frac{\omega_{w,max}}{\max(\mathbf{u}_{des})}$$

where $\mathbf{u}_{des}$ is our vector of wheel speeds calculated by the pose controller. We then scale our entire wheel speed vector to by this scalar:

$$\mathbf{u}_{des} = \lambda_w \mathbf{u}_{des}$$

Our maximum wheel speed can also by non-constant. For instance, we can choose to set up a motion profile for this value, allowing us to smoothly accelerate and decelerate.

Holonomic Pure Pursuit - stopping

In order to stop at the end-point of the path, we should eventually turn off the Pure Pursuit algorithm and simply use the pose controller. To do this, we can set a distance threshold d_{thresh} to the end-point. If the distance from the robot to the end-point is less than the threshold value, then we simply use the pose controller to come to a smooth stop.

We can also choose to set d_{thresh} equal to ℓ_{xy}. Doing this eliminates this tune-able variable, but will likely only work if we use motion profiles so that we are already slowing down, since the lookahead circle may be relatively small.

Finally, we should specify a final desired heading to stop at. This will prevent the robot from trying to turn towards the end-point of the path. We will turn towards this heading if our threshold condition is met.

Holonomic Pure Pursuit - static heading & reversing

In some cases, we may want to keep our heading static (i.e., constant) while we follow the path. This is possible since our drivetrain is holonomic.

To keep our heading constant, we can simply set θ_{des} to some value of our choice. Doing this may help our localization accuracy as our robot will not make any aggressive turns throwing off our heading estimation.

In some cases, we may want to follow a path backwards, i.e., turning the back of the robot towards the point-to-turn-to. For this, we would simply want to offset our desired heading by 180°:

$$\theta_{des} = \begin{cases} \theta_{des} + \pi & \text{sign}(\theta_{des}) = -1 \\ \theta_{des} - \pi & \text{sign}(\theta_{des}) = 1 \end{cases}$$