The pure pursuit path following algorithm

Juan Francisco Rascón Crespo

juan.rascon@eurecat.org

1 Introduction

This document is just a summary of some documents I have worked with when developing the pure pursuit algorithm for a project in the company I work for, Eurecat. Given the general success of this algorithm in literature and its simplicity, it is likely that it will be used again in forthcoming land-based-navigation projects, so I decided to compile here my notes for future reference.

This document aims to expose the geometric derivation for the path following algorithm called pure pursuit for differential drives.

This report also includes a geometric derivation of the method, and presents some insights into the performance of the algorithm as a function of its parameters.

The pure pursuit algorithm is a tracking algorithm that works by calculating the **curvature**, γ, that will move a vehicle **from its current position to some chosen path position**. The whole point of the algorithm is to choose a goal position that is **some distance ahead of the vehicle on the path** (a waypoint). The name pure pursuit comes from the analogy that it is used to describe the method. *We tend to think of the vehicle as chasing a point on the path some distance ahead of it, i.e., it is pursuing that moving point.* That analogy is often used to compare this method to *the way humans drive.* We tend to look some distance in front of the car and head toward that spot. This lookahead distance, denoted as L_{ah}, changes as we drive to reflect the twist of the road and vision occlusions.

2 Geometric derivation

The pure pursuit algorithm is a method of geometrically determining the curvature that will drive the vehicle to a goal point. This goal point is a point on the path that is one lookahead distance, L_{ah}, from the current vehicle's position. An arc that joins the current vehicle's position and the goal point is found out. The chord of this arc is the lookahead distance, and this is a constraint in determining the unique arc that joins the two points. Consider the lookahead distance to be analogous to the distance to a spot in front of a car that a human driver might look toward to track the roadway. The result of the algorithm is the curvature, γ of the arc that joins the vehicle's position to the chosen goal point and whose chord length is L_{ah}

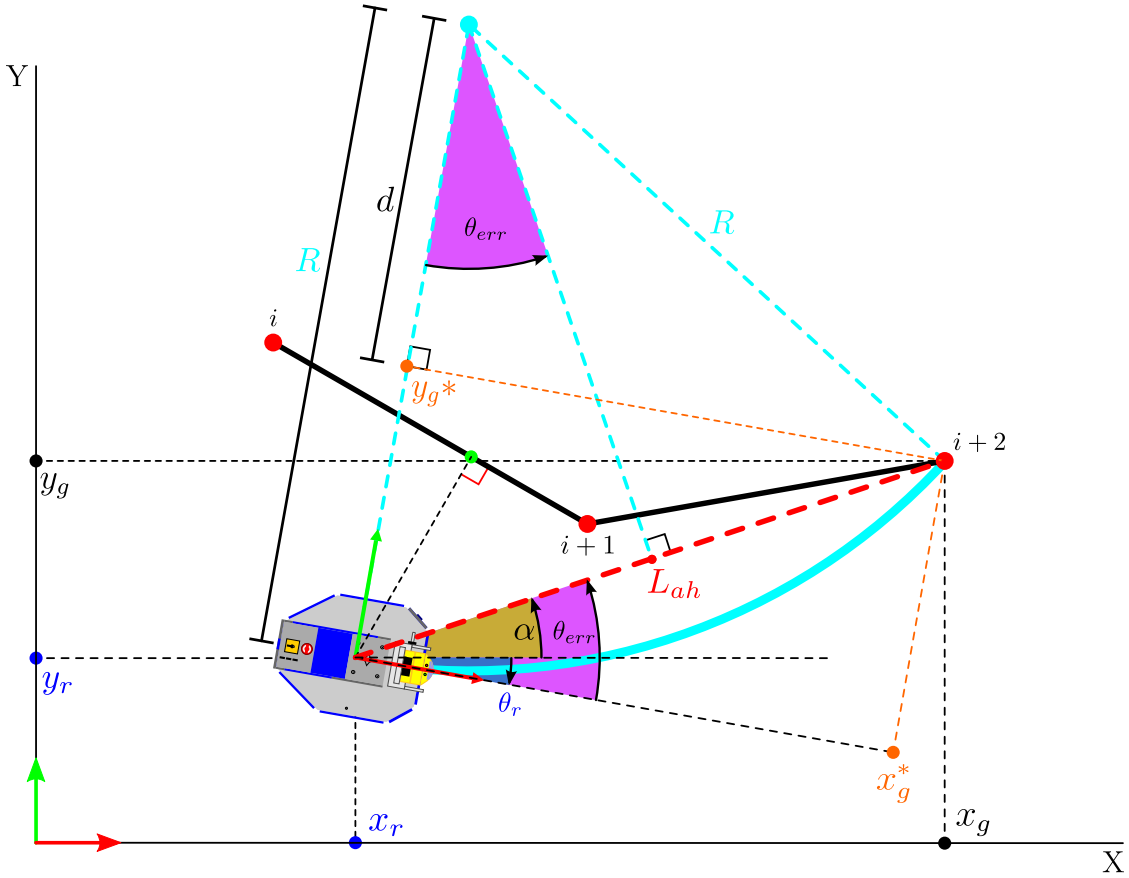

Figure 1: Geometric analysis for the pure pursuit algorithm.

$$L_{ah} = \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2} \tag{1}$$
$$= \sqrt{x_g^{*2} + y_g^{*2}} \tag{2}$$

$$\theta_{err} = \alpha - \theta_r \tag{3}$$
$$= \arctan\left(\frac{y_g - y_r}{x_g - x_r}\right) - \theta_r \tag{4}$$

$$R = d + y_g^* \tag{5}$$
$$R^2 = d^2 + x_g^{*2} \tag{6}$$
$$= \left(R - y_g^*\right)^2 + x_g^{*2} \tag{7}$$

$$R^2 = R^2 - 2\,R\,y_g^* + y_g^{*2} + x_g^{*2} \tag{8}$$

Since

$$L_{ah} = \sqrt{x_g^{*2} + y_g^{*2}} \tag{9}$$
$$y_g^* = L_{ah}\,\sin\left(\theta_{err}\right) \tag{10}$$
$$\tag{11}$$

equation 8 becomes:

$$L_{ah}^2 = 2\,R\,y_g^* \tag{12}$$
$$= 2\,R\,L_{ah}\,\sin\left(\theta_{err}\right) \tag{13}$$
$$L_{ah} = 2\,R\,\sin\left(\theta_{err}\right) \tag{14}$$

Finally:

$$\gamma = \frac{1}{R} = \frac{2\,\sin\left(\theta_{err}\right)}{L_{ah}} \tag{15}$$

3 Implementation

The implementation of the pure pursuit algorithm is fairly straightforward. The pure pursuit algorithm can be outlined as follows:

- Determine the robot's current location.
- Find the path's goal waypoint to navigate to. This action can be done in different ways. The way I have implemented this now is that in the software I store the path's index for the last goal waypoint the robot navigated to because this waypoint was one lookahead distance from it. Next time the algorithm has to find a goal point that is one lookahead distance from the robot's position, it starts looking from this index onwards, selecting the first waypoint that satisfies this distance constraint.
- Calculate the curvature using the previous equations and request the vehicle to set the steering to that curvature.
- Back to the first step.

4 Effects of changing the lookahead distance

There is one parameter in the pure pursuit algorithm, the lookahead distance. The effects of changing the parameter L_{ah} are easy to imagine using the analogy to human driving.

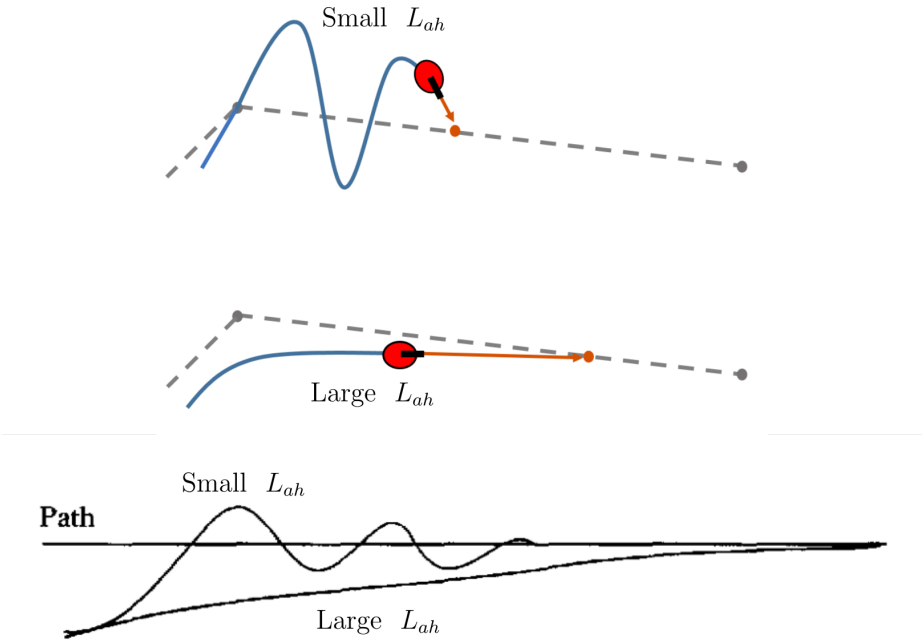

Figure 2: Effect of L_{ah} in the tracking behaviour.

Longer lookahead distances tend to converge to the path more gradually and with less oscillation. The response of the pure pursuit tracker looks similar to the step response of a second-order dynamic system and the value of L_{ah} tends to act as a damping factor.

The longer the lookahead distance, the less *curvy* of a path that can be followed. The algorithm is calculating a curvature so that the vehicle can drive an arc. If the path between the vehicle and the goal point is sufficiently *curvy* then there might not be an arc that uses the current value of L_{ah} to join the two points; any driven arc will induce error.

Just a final word about the lookahead distance. Try to make it at least equal to the differential base's track width (distance between wheels). That way, in the unusual case when $\theta_{err} = \frac{\pi}{2}$, the radius R is half the distance between wheels and in that case the robot will turn around one of the wheels, while the other will remain stopped. If the lookahead distance is less than the track width, then in the weird case when $\theta_{err} = \frac{\pi}{2}$ the turning radius, R, will be less than half the distance between wheels and then the robot will turn around a point that is located somewhere in between one of the wheels and the robot's center, and even when this is not harmful it is not very useful.

A video showing this: https://tinyurl.com/y6ox5zee

5 How to command the differential drive to follow the curvature

Since we are commanding a differential drive, we have to tell the low-level controller the linear speed, v, and the angular speed, ω, for the robot to achieve the desired curvature. The pure pursuit algorithm does not tell you what values are for these speeds; therefore, here you have to be creative and design a strategy that fits your application and your platform. I'll show what I've done in my particular case, but for yours, this might be inadequate. So feel free to experiment. The only relationship we know that relates the curvature given by the pure pursuit controller and the speeds we have to compute is:

$$v = R\,\omega \tag{16}$$

$$= \frac{1}{\gamma}\,\omega \tag{17}$$

My strategy for the linear speed is this one:

- If the angular error, θ_{err}, is within the interval defined by a small enough angle, $[-\theta_{min}, \theta_{min}]$, then the robot will move in straigh line with speed V_{max}. Let's say that $\theta_{min} = 5°$. So if $|\theta_{err}| \leq \theta_{min}$ then, the heading error can be considered small enough to assume that the robot can move in a straight line at maximum linear speed. Besides, the angular speed, ω, is forced to be 0 rad/s.

$$\begin{aligned} v &= v_{max} \quad |\theta_{err}| \leq \theta_{min} \\ \omega &= 0 \qquad |\theta_{err}| \leq \theta_{min} \end{aligned}$$

- If the angular error is outside the interval defined by a large enough angle, $[-\theta_{max}, \theta_{max}]$, then the robot will turn-in-place to correct its heading and to point to the proper goal waypoint. Let's say, for instructional purposes, $\theta_{max} = 70°$. If the heading error is so huge to be outside the interval $[-\theta_{max}, \theta_{max}]$, then there is no point in following the path. Stop it and turn in place to recover the proper orientation. In this situation, $v = 0$ m/s and ω is going to increase linearly with the heading error up to its maximum value. The

maximum angular speed, in the module, is achieved when the heading error, in absolute value, exceeds the predefined angle $\theta_{rotmax} = \frac{\pi}{2}$ rad, for example.

$$
\begin{aligned}
v &= 0 && |\theta_{err}| \geq \theta_{max} \\
\omega &= \left(\tfrac{\omega_{maxrot}-\omega_{minrot}}{\theta_{rotmax}-\theta_{rot}}\right)(\theta_{err}-\theta_{max})+\omega_{minrot} && \theta_{err} \geq \theta_{max} \\
\omega &= \left(\tfrac{\omega_{maxrot}-\omega_{minrot}}{\theta_{rotmax}-\theta_{rot}}\right)(\theta_{err}+\theta_{max})-\omega_{minrot} && \theta_{err} \leq -\theta_{max}
\end{aligned}
$$

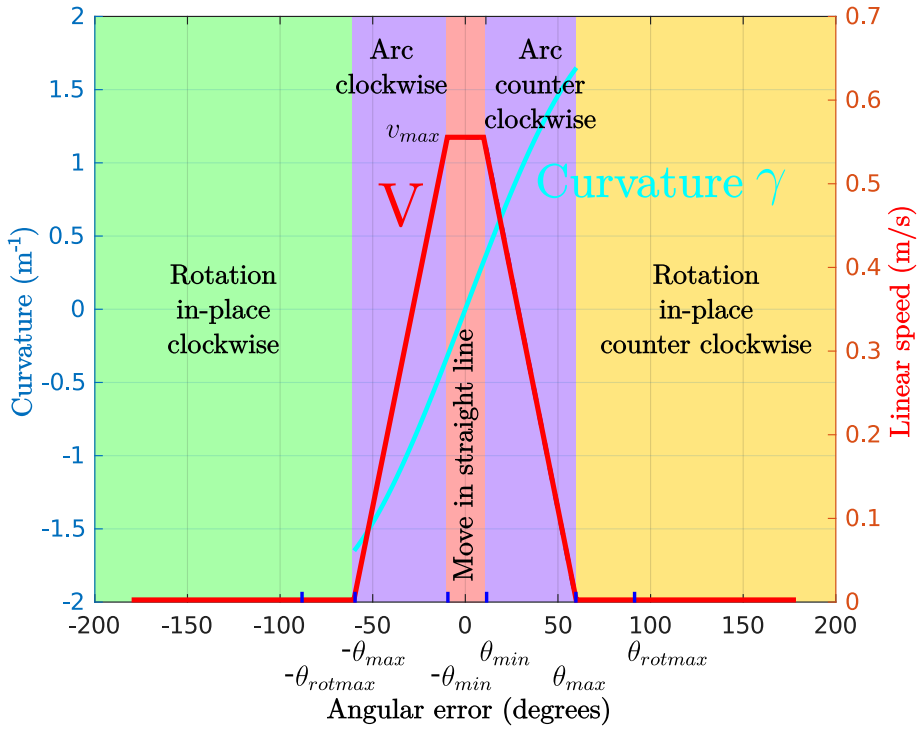

Figure 3: Linear speed and curvature (in cyan).

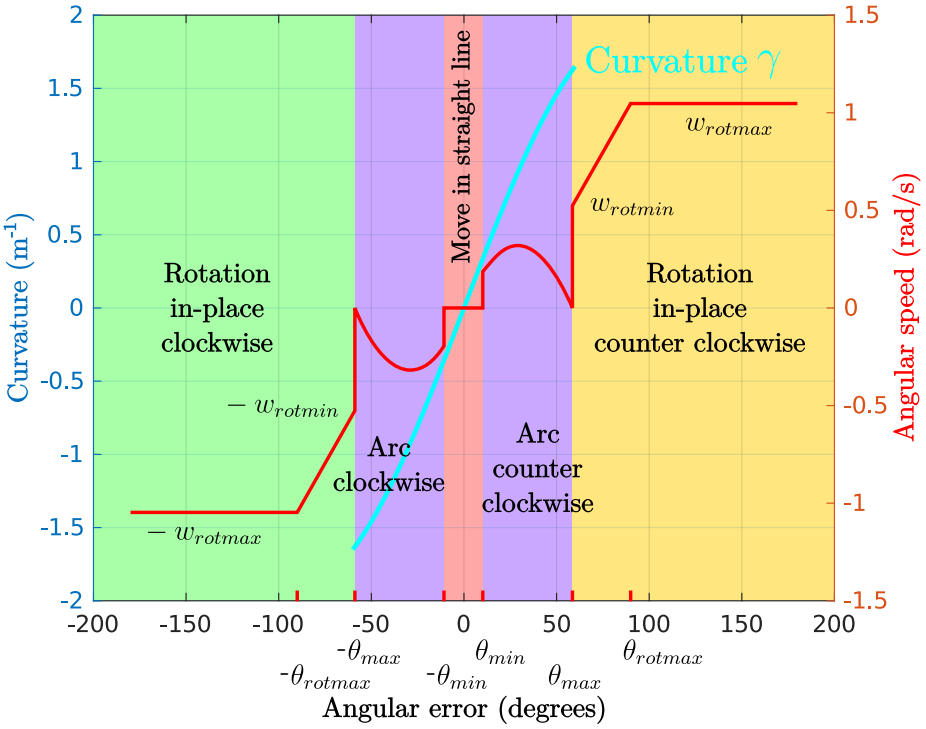

Figure 4: Angular speed and curvature (in cyan).

- If the angular error, in absolute value, is within the interval $[\theta_{min}, \theta_{max}]$ then the robot moves describing an arc with curvature γ. In this situation the linear speed and the angular speed are describe by these expressions:

$$v = \left(\frac{-V_{max}}{\theta_{max}-\theta_{min}}\right)(|\theta_{err}| - \theta_{max}) \quad \theta_{min} \leq |\theta_{err}| \leq \theta_{max}$$
$$\omega = \frac{v}{R} = v\,\gamma \qquad\qquad \theta_{min} \leq |\theta_{err}| \leq \theta_{max}$$

  Finally, the algorithm should check if the computed angular speed is below its maximum level all the time the vehicle is in this interval, ω_{max}. To be clear, I'm considering a maximum angular speed when the robot rotates in place, ω_{rotmax}, also a minimum angular speed when it rotates in place, ω_{rotmin}, and also a different maximum angular speed when the robot is traversing an arc, ω_{max}. I consider all these speeds to be as flexible as possible. Then you can merge those values as you want.

  Two cases here. If the angular speed, ω is below its maximum level in this interval all the time, then the algorithm has finished.

$$\text{When } (\theta_{min} \leq |\theta_{err}| \leq \theta_{max}) \longrightarrow |\omega| \leq w_{max}$$

It might happen that the computed angular speed at any moment in this interval were, in absolute value, $|\omega|$, greater than its maximum level, ω_{max}, $|\omega| > \omega_{max}$. In that case, you should truncate the angular speed's value to be below its maximum level. Now, as you want the vehicle to follow the computed curvature, to traverse the found arc, then the algorithm has to re-compute the linear speed, now with the truncated angular speed and the curvature:

$$v = R\omega = \frac{\omega}{\gamma}$$

Be careful to check if the curvature is zero before dividing.

Some images to show what I've just explained.

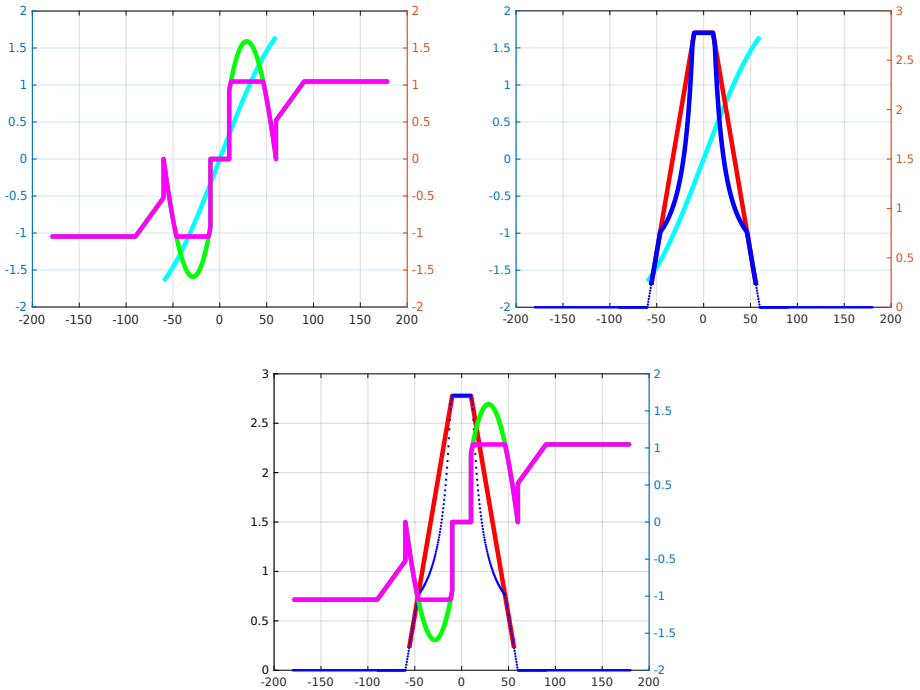

Figure 5: Truncated angular and linear speed.

6 References

Implementation of the Pure Pursuit Path Tracking Algorithm, R. Craig Coulter, CMU-RI-TR-92-01.