ECE 6562 PROJECT FINAL REPORT

# CURVE TRACKING USING PURE PURSUIT ALGORITHM

Jingyu Shi

## Project Summary

In this project, I studied the pure pursuit for curve tracking task of a mobile robot. The theoretical derivation is given in the following sections. An simulation of this algorithm is performed and experiments on system parameters, namely look-ahead-distance and desired velocity, are carried out. The results of the experiments align with the theoretical deduction that either too small or too large look-ahead-distance will impair the system performance and so will velocity value. A dynamic setting of look-ahead-distance based on velocity is given.

## Background

For unmanned vehicles, after planning the path which is usually called the global path.A global path consists of a series of path points. These path points merely contain spatial position information. Pose or ego information is not necessary for a global path, neither is time steps. These path points are called Global Way-points. The difference between a path and a trajectory is that a trajectory also contains time information. Time constraints are added to the Global Way-points. We call these trajectory points as Local Way-points.

## Objective

The objective of this project is to learn and implement a methodology for curve tracking task, which is, given a reference path, designing a control law for a mobile robot to follow this path.

In the process of such path tracking, the reference path curve can be independent of time parameters. During tracking control, it can be assumed that the mobile robot is moving at a constant speed at the current speed, and the driving path is approached to the reference path at a certain cost rule, while in trajectory tracking, the reference path curve is related to time and space, and thus it requires the mobile robot to reach a preset reference path point within a specified time.

Path tracking is different from trajectory tracking. It is not subject to time constraints and only needs to track the reference path within a certain error range. Motion control in path tracking is to find a bounded control input sequence to make the unmanned vehicle go from an initial configuration to a set desired configuration.

# Methodology

There are two major methodologies for curve tracking of unmanned vehicles, geometry-based tracking and model-based tracking. This project focuses on one of the geometry-based methods, namely Pure Pursuit. The outline of Pure Pursuit algorithm is:

**1** Determine the current location of the robot

**2** Find the closest way point to the robot

**3** Find the goal

**4** Calculate the curvature and control the robot to set its steering to the curvature

**5** Update the location of the vehicle

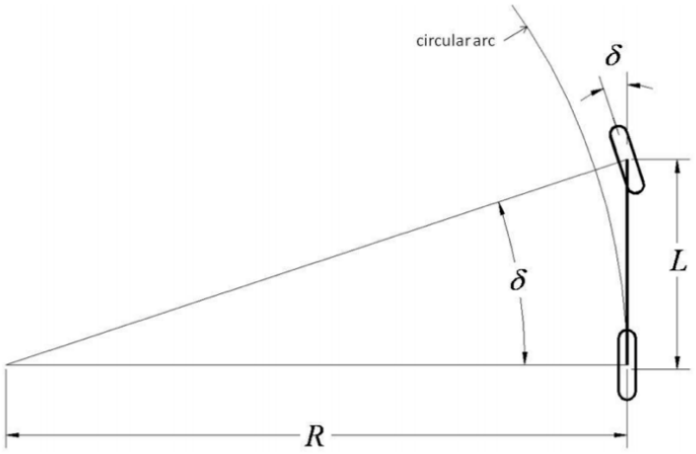


**Figure 1:** Bicycle Model for Mobile Robot

To study Pure Pursuit, we can start with the introduction of bicycle model for a mobile robot. As shown in Figure 1, the bicycle model is indeed a simplified version of Ackermann Steering Geometry, where the four-wheel vehicle is simplified to a two-wheel vehicle that only moves on a plane. The steering angle of the front wheel and the rear axle follow the

geometric equation:

$$\tan(\delta) = \frac{L}{R} \tag{1}$$

where $\delta$ is the steering angle of the front wheel, $L$ the wheelbase and $R$ the radius of the circular arc that the mobile will be following at this time step.

Pure Pursuit algorithm, as shown in Figure 2, takes the rear axle of the vehicle as the tangent point and the longitudinal body of the vehicle as the tangent. By controlling the front wheel angle, the vehicle can drive along a circular arc passing through the goal point. Point $(g_x, g_y)$ is the next way-point on the path we are tracking, which is obtained by searching the closest point to the current position of the robot within $l_d$, the "look ahead distance". To control the

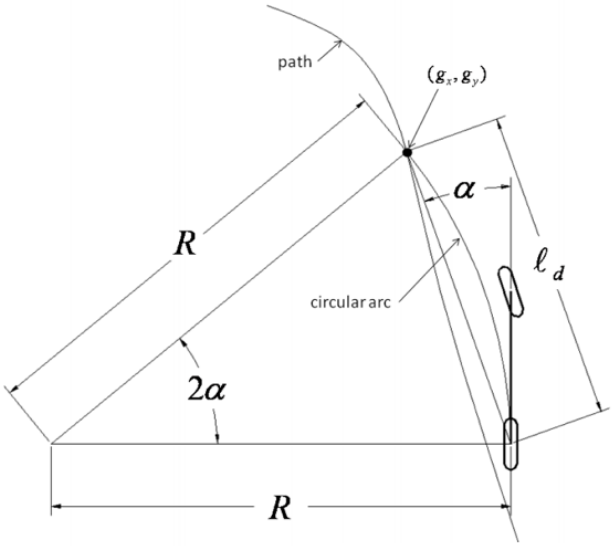


**Figure 2:** Pure Pursuit

robot to move to point $(g_x, g_y)$, we need to design a control for choosing the steering angle $\delta$ of the front wheel. According to the law of sines we have:

$$\frac{l_d}{\sin 2\alpha} = \frac{R}{\sin\frac{\pi}{2} - \alpha} \tag{2}$$

$$\frac{l_d}{2\sin\alpha\cos\alpha} = \frac{R}{\cos\alpha} \tag{3}$$

$$\frac{1}{R} = \frac{2\sin\alpha}{l_d} \tag{4}$$

Since

$$\delta = \tan^{-1}\frac{L}{R} \tag{5}$$

We eventually obtain a controller over steering angle $\delta$:

$$\delta = \tan^{-1}\frac{2L\sin\alpha(t)}{l_d} \tag{6}$$

We can see that the pure pursuit algorithm is tracking the curve by controlling the steering angle $\delta$ of the bicycle model, given $\alpha$, the angle between mobile robot orientation and the way point being tracked.

## Simulation

The simulation system is given in Figure 3. A built-in block for Pure Pursuit algorithm is used. The output linear velocity is passed to a saturation unit to set a limit. The $x$ and $y$ coordinates and the orientation are calculated by integration of linear velocity and angular velocity respectively. Additionally for each time step, the distance between the current position of the robot and the goal is computed. If the distance is within a threshold, the robot is considered to have reached its destination.

**Listing 1:** code for random path generation

```
function path = randomcurve(n,m)
    step = (2.5.*rand(n,2)-1).*m./n;
    path = cumsum(step);
    path(:,1) = smooth(path(:,1));
    path(:,2) = smooth(path(:,2));
end
```

The code for generating random 2-dimensional path is given in Listing 1. This script takes in the number of points to sample $n$ and an approximate travel distance $m$ and generates steps using a uniform distribution. The path is obtained by accumulate these steps.

At the experiment stage, the parameters settings are given in Listing 2.

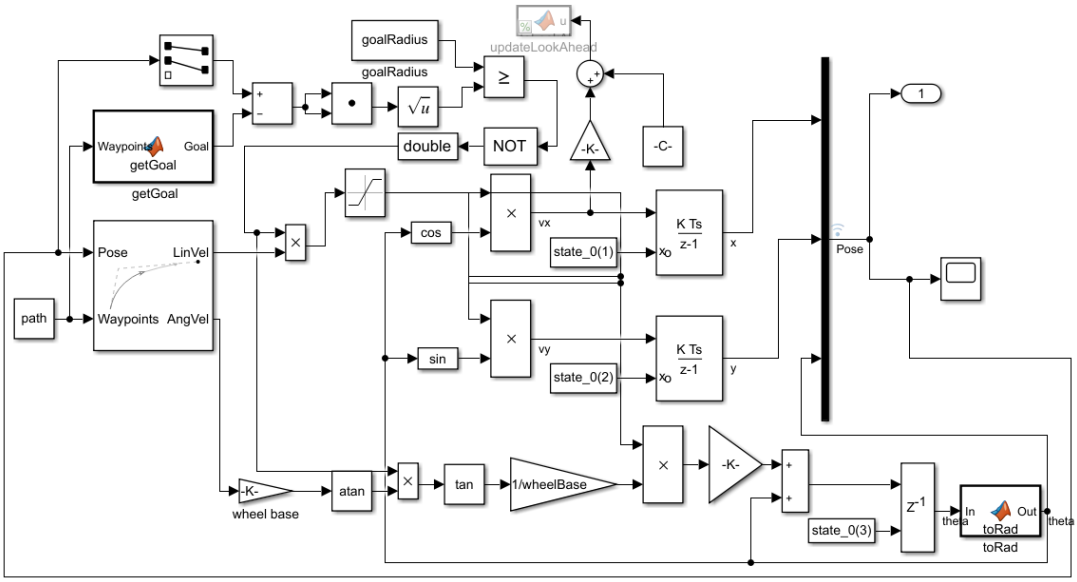


**Figure 3:** The simulation system

**Listing 2:** parameter settings

```
desiredLinearVelocity = 5; % the desired linear velocity for pure pursuit
maxAngularVelocity  = 10; % the limit for angular velocity
lookaheadDist  = 0.1; %the look ahead distance for pure pursuit
goalRadius = 0.1;  %the radius of the destination, the robot stops within the circle
maxLinearVelocity = 80; %the limit for the saturation unit
wheelBase = 1; %the distance between front wheel and rear wheel
sampleTime = 0.01; %the sample time for simulation
state_0 = [0 0 0]; %the initial state of the robot
```

# Results

The system are experimented on different look-ahead-distances and desired linear velocity. I firstly set the $lookahaeadDist = 0.001, 0.1$ and 1. The resulted trajectories are given in Figure 4, Figure 5 and Figure 6 respectively. The red curves are the output trajectories and the blue curves are the given paths.

According to Equation 6, $lookaheadDist$ $l_d$ contributes vastly to the steering angle $\delta$ of each time step. Theoretically, a small $l_d$ can make the tracking more accurate, since $\delta$ can be large enough to keep track with the curve, while a large $l_d$ can make the tracking smoother

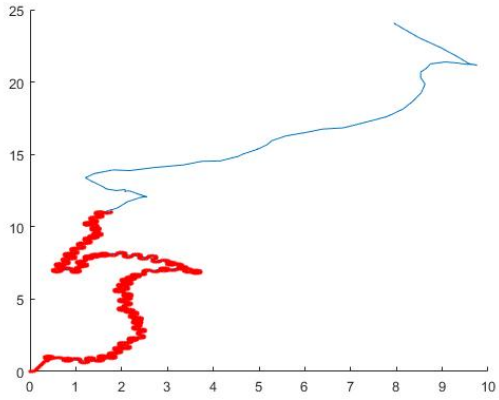


**Figure 4:** $lookahaeadDist = 0.001$

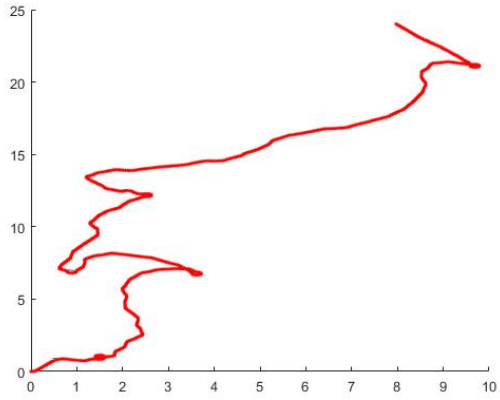


**Figure 5:** $lookahaeadDist = 0.1$

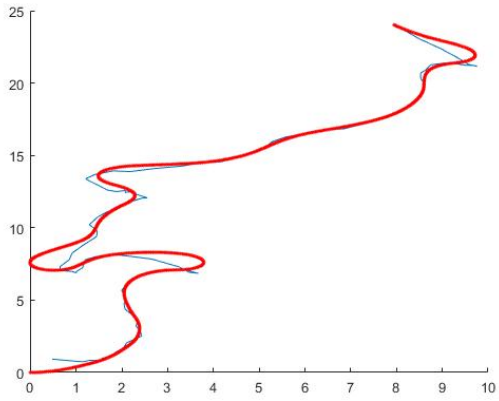


**Figure 6:** $lookahaeadDist = 1$

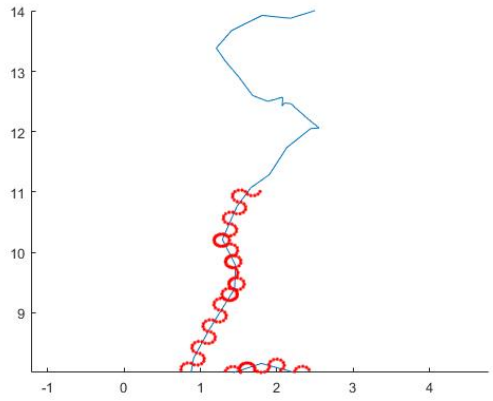


**Figure 7:** $lookahaeadDist = 0.001$, system Oscillating

due to a small $\delta$ value. However, if $l_d$ is too small the system will be unstable and if $l_d$ is too large the system will perform terribly.

The simulation results align with the theory. In Figure 4, since $l_d$ is too small resulting in a large steering angle, the system keeps oscillating around the given path as shown in Figure 7. In Figure 6, since $l_d$ is too big, the trajectory is not as close to the given curvature as it is in Figure 5, where an appropriate value of $l_d$ is chosen.

Additionally, since the built-in Pure Pursuit block in Simulink does not provide a dynamic setting of look-ahead-distance, if the velocity of the robot is too large, the system will be unstable as shown in Figure 8, Figure 9, Figure 10 and Figure 11. As the value of velocity increases, the system oscillates more. In Figure 11, when the value of velocity is too large, the system is unstable and does not work.

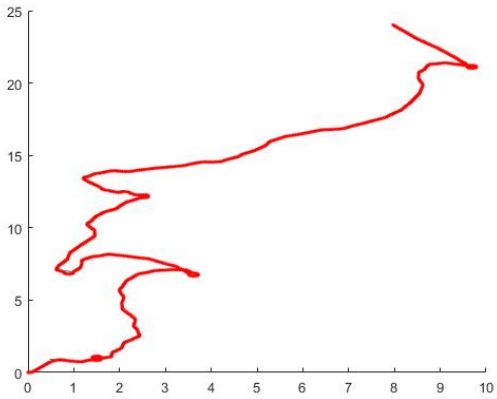


**Figure 8:** $desiredvelocity = 5$

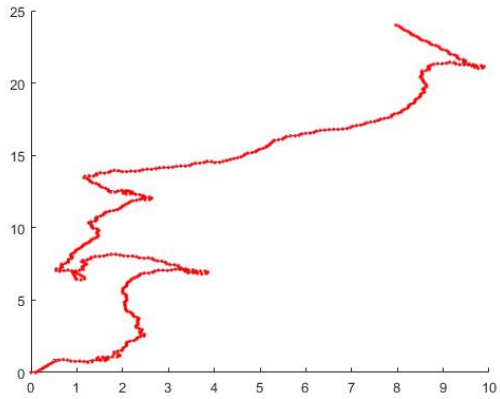


**Figure 9:** $desiredvelocity = 10$

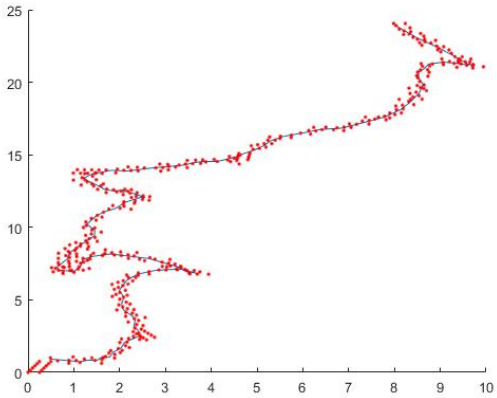


**Figure 10:** $desiredvelocity = 25$

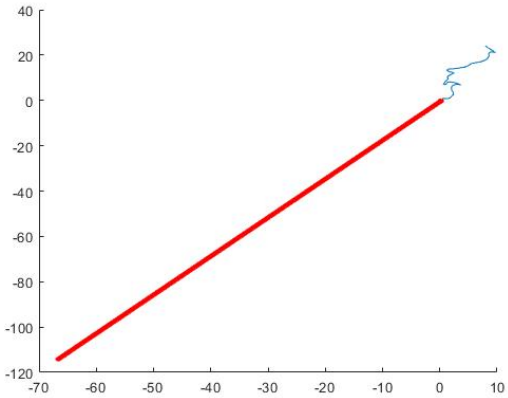


**Figure 11:** $desiredvelocity = 40$

A potential solution to this problem is setting the look-ahead-distance as a function of the velocity of the robot:

$$l_d = kv(t) + l_{d0} \tag{7}$$

Then Equation 6 becomes:

$$\delta = \tan^{-1} \frac{2L \sin \alpha(t)}{kv(t) + l_{d0}} \tag{8}$$

The system will be stable again even if the robot is moving relatively fast.