Vehicle Dynamics Simulation
Entry Programming Assignment

Ronen Rojas
AV control Team, Mobileye

May 2024

Contents

List of Figures

1 Abstract

The goal is to build a small library that simulates a $2D$ vehicle that tracks a simple geometric $2D$ path, along with a small command line utility that uses that library. With the programming assignment you'll be asked some theoretical questions.

This assignment is not all for didactic reasoning per se, it's a soft introduction to what we do in the team, a peek to the realm of vehicle dynamics and control.

The coding tasks must be implemented in `python3` or in `C++`. Make sure to pay close attention to the design of your code, names, constants, comments and it must be in the *OOP* paradigm. For `python3` we kindly recommend to follow PEP8 style guide, for `C++` code we kindly recommend using camelCase convention.

The simulation should explicitly implement the dynamic models and algorithms as explained in the following sections. There are some optional implementation assignments that will be more intricate and show a deeper insight. We highly recommend that you do these as well.

2 Models and algorithms

In these next sections we'll elaborate on each component that should be simulated and implemented.

2.1 Coordinate systems

2.1.1 Global coordinate system

This *Cartesian* coordinate systems is the main coordinate frame in which the output of the simulation will be displayed:

- **Origin** - O_G: The beginning of the desired path.
- **X axis** - X_G: Aligned with the initial heading of desired path.
- **Y axis** - Y_G: Creates a right hand coordinate system with Z_G.
- **Z axis** - Z_G: Facing upwards.

2.1.2 Ego coordinate system

This *Cartesian* coordinate systems is a coordinate frame of the vehicle:

- **Origin** - O_E: Vehicle center rear axle.
- **X axis** - X_E: Aligned with road wheel angle $\delta = 0$ axis.
- **Y axis** - Y_E: Creates a right hand coordinate system with Z_E.
- **Z axis** - Z_E: aligned with Z_G.

Denote ψ as the angle between X_G and X_E *i.e.*, the heading angle . ψ is positive with positive Z_E rotation direction.
See illustration for a better understanding:

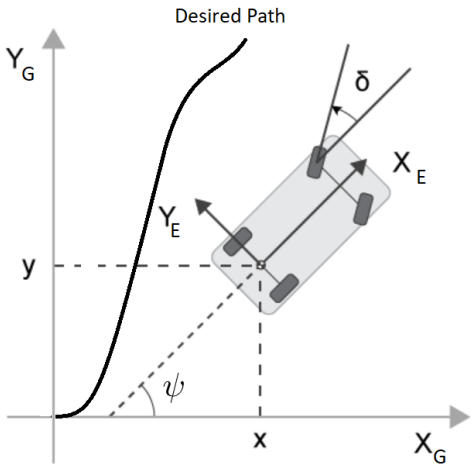

Figure 1: Path coordinate system

2.1.3 Path coordinate system

This *curvilinear* coordinate system is along the path, better described as (s, t) tuple:

- **Origin** - (s,t)= (0,0) coincides with O_G.
- **s coordinate** - The longitudinal coordinate along the path
- **t coordinate** - The normal coordinate along the path, *i.e.* at point s the distance perpendicular to the path.

See illustration for a better understanding:

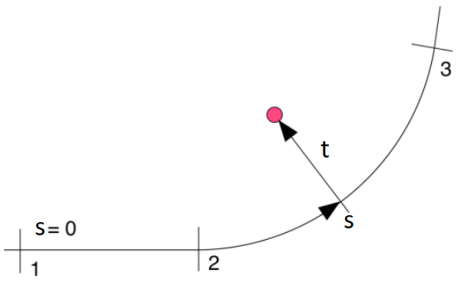

Figure 2: Path coordinate system

2.2 Path object

The path object should encapsulate the coordinates of the desired path in the *global coordinate system* and the transformation between this system and the *path coordinate system*. A path in this context should be either a straight line or a circle. Analytic or numerical representation of the path coordinates are acceptable.

The path object can contain any other relevant functions.

(optional):
Build a general path which is numerically described as a $2D$ array and will be given as an input to the simulation as further explained.

2.3 Vehicle object

The vehicle object should encapsulate the state of the kinematics of the vehicle frame: position, velocity and angular velocity. Also it should encapsulate the transformation between *global coordinate system* and *ego coordinate system*.

The lateral actuator will be road wheel angle denoted δ. A positive road wheel angle will make positive rotation in the Z_E axis. For simplicity assume that the velocity is always aligned with X_E.

The Vehicle object can contain any other relevant functions.

(optional):
Consider the option where the localization is not perfect, *i.e.* the vehicle object only has an estimation of the heading angle and its global position. Add a random white noise to these states.

2.4 Longitudinal dynamics

The vehicle longitudinal dynamics is constant speed positive denoted v.

2.5 Lateral dynamics

The lateral dynamics should adhere to the kinematic bicycle model. For vehicle parameters use Zeeker 01 model.

2.5.1 Servo dynamics

For simplicity assume that the servo actuator controls the road wheel angle.

consider a time constant delay between the command and the actual road wheel angle $\Delta t = 200[msec]$.

Assume that the maximum road wheel angle is between $|\delta| \leqslant 45^\circ$ and that the servo has a rate limiter of $|\dot{\delta}_{command}| \leqslant 20[\frac{^\circ}{sec}]$

(optional):
Consider a closed loop servo-mechanical actuator between the command and the actual road wheel angle, model it as a 2nd order closed loop system with $\omega_{delta}, \zeta_{delta}$ as bandwidth and damping respectively.

2.6 Tracking algorithm

The path tracking algorithm for the vehicle should be a simple *Pure Pursuit*. As shown in the illustration and explained further in this section:

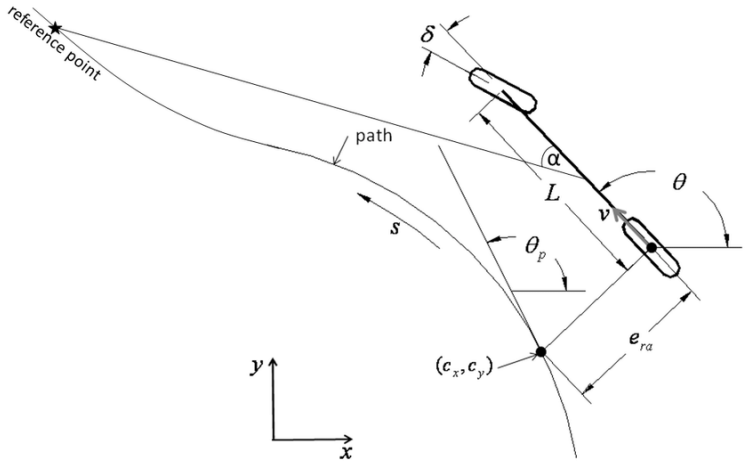

Figure 3: Pure pursuit tracking

Given a desired path at a point in time we want to calculate the desired turn radius in the following steps:

1. The starting point of the path coordinate $(s_{start}, 0)$ (or (c_x, c_y) in the illustration) is the intersection of Y_E and the path.
2. Calculate the reference point $(s_{start} + d_{look\ ahead}, 0)$ in path coordinates.
3. Calculate the radius of the circle with 2 points: O_E and the reference point. Assume also X_E is tangent to this circle.

Food for thought, should $d_{look\ ahead}$ be a constant at all speeds?

2.7 Stanley Controller (optional)

There a different approach to the lateral control algorithm a.k.a. Stanley Controller.

Implement this controller and compare it to the performance of the pure pursuit tracking algorithm

3 Requirements

3.1 Implementation Requirements

1. The simulation should be implemented in `python3` or `C++`.
2. The simulation should run until the end of the path or 300 seconds, whichever happens first.
3. Use only standard packages in `python`: pandas, numpy, matplotlib, etc.
4. The `python3` usage should be:

   ```
   $ python3 <py_file> -x0 0.1 -y0 0.1 -psi 0.0 -v 1.0
   ```

5. Use only standard libraries in `C++`: std, math, etc.
6. The `C++` usage should be

   ```
   $ make all
   $ <exec_file> -x0 0.1 -y0 0.1 -psi 0.0 -v 1.0
   ```

7. $x0$, $y0$ is the initial position; ψ is the initial heading; v is the vehicle speed.
8. For the optional assignment of a 2D array for the desired path the program should get a csv file of the array as an input.
9. The program has to plot the real and the desired path of the vehicle in global coordinates system.
10. The program has to plot the lateral kinematic state of vehicle vs. time e.g. yaw rate.
11. Choose your own step size of the simulation and integration method.

3.2 Submission Requirements

1. Write a Readme.md file that explains the main classes in your programs and their functionality.
2. The theoretical questions should be submitted in English in a pdf file.
3. The source code should be submitted in a remote **private** git repository (meaning not open to everybody in the internet community). We highly recommend using `www.gitlab.com`
4. If you do use gitlab please grant maintainer permission to the users: *ronen.rojas@mobileye.com*, *nirel.kakon@mobileye.com* and *rotem.itzharkatz@mobileye.com*.
5. The commits should reflect the progress in the assignment and commit messages should be informative.

Important note:
You cannot save this project assignment into a your private platform or personal project space other then the aforementioned private repository

4 Theoretical Questions

1. What will be the steady state effect of a bias in mechanical road wheel servo for the pure-pursuit tracking?
   (a) Show this effect in your simulation.
   (b) Develop an expression between the aforementioned bias and the steady state effect (optional)
2. It is known that kinematic bicycle model is only valid in low speed $\sim 5\frac{m}{sec}$, suggest a model for higher speeds and state what cannot be neglected anymore.

5 References

We highly recommend using the **Vehicle Dynamics and Control** book for a deeper understanding and for implementation.

A good online lecture for vehicle dynamics is `https://www.youtube.com/watch?v=LZ82iANWBL0`.

Thank you and good luck.