Project: Part 1

24-677 Special Topics: Modern Control - Theory and Design

Prof. D. Zhao

Due: Nov 2, 2021, 11:59 pm. Submit within the deadline.

- Your online version and its timestamp will be used for assessment.
- We will use Gradescope to grade. The link is on the panel of CANVAS. If you are confused about the tool, post your questions on Campuswire.
- Submit **your_controller.py** to Gradescope under **Programming-P1** and your solutions in **.pdf** format to **Project-P1**. Insert the performance plot image in the **.pdf**. We will test **your_controller.py** and manually check all answers.
- We will make extensive use of Webots, an open-source robotics simulation software, for this project. Webots is available here for Windows, Mac, and Linux.
- For Python usage with Webots, please see the Webots page on Python. Note that you may have to reinstall libraries like `numpy`, `matplotlib`, `scipy`, etc. for the environment you use Webots in.
- Please familiarize yourself with Webots documentation, specifically their User Guide and their Webots for Automobiles section, if you encounter difficulties in setup or use. It will help to have a good understanding of the underlying tools that will be used in this assignment. To that end, completing at least Tutorial 1 in the user guide is highly recommended.
- If you have issues with Webots that are beyond the scope of the documentation (e.g. the software runs too slow, crashes, or has other odd behavior), please let the TAs know via Campuswire. We will do our best to help.
- We advise you to start with the assignment early. All the submissions are to be done before the respective deadlines of each assignment. For information about the late days and scale of your Final Grade, refer to the Syllabus in Canvas.

1 Introduction

This project will entail using Webots, an open-source robotics simulation software, to learn more about control methods for autonomous vehicles. Webots was chosen because of its ease of use, flexibility, and impressive rendering capability. A brief guide for Windows, Mac OS, and Linux follows. Please see the links on the previous page for a link to documentation and more information.

For Windows:

1. Download and install Webots R2021a.
2. Download and install Python natively from Python's official webpage. 3.9 is recommended, but any version from 3.7 - 3.9 will work.
3. Open a Command Prompt window (right-click and select "Run as Administrator"). On the command line, install `numpy`, `scipy`, and `matplotlib` with the following command:
   `python -m pip install <package-name>`.
4. If you encounter issues installing `matplotlib`, please try upgrading `pip` and `setuptools`::
   `python -m pip install --upgrade pip`
   `python -m pip install --upgrade setuptools`

For Mac:

1. Download and install Webots R2020b.
2. Download and install Python natively from Python's official webpage. 3.8 is recommended, but 3.7 will also work.
3. In a Terminal window, install `numpy`, `scipy`, and `matplotlib` with the following command:
   `pip3.x install <package-name>`
   where `x` is the version number you installed.
4. Check where your Python is installed with:
   `which python3.x`
   Copy the path that appears and open Webots. In the Webots menu at the top, open Tools → Preferences, and paste the copied path under "Python command".

For Linux:

1. Download and install Webots R2021a.
2. In a terminal window, make sure you have `numpy`, `scipy`, and `matplotlib` with the following command:
   `pip install <package-name>`.

As you might already know, Buggy is a time-honored CMU tradition (you can learn more about it here and here). Unfortunately, it was not able to be held in-person this year due to the pandemic. To help make up for this, you will be asked to create a controller that helps a vehicle to complete the same course in simulation. However, instead of controlling an actual Buggy, you'll be controlling a Tesla Model 3. It's a bit more fun and stylish (we hope)!

We will follow the steps listed below to learn more about the system and synthesize several different kinds of controllers:

1. Examine the provided nonlinear control model
2. Linearize the state space system equations [P1]
3. Develop a PID controller for the system [P1]
4. Check the controllability and stabilizability of the system [P2]
5. Design a full-state feedback controller using pole placement [P2]
6. Design an optimal controller [P3]
7. Implement A* algorithm [P3]
8. Implement an extended Kalman filter (EKF) for simultaneous localization and mapping (SLAM) [P4]

The project has been divided into 4 parts to reflect this:

- P1 (a) Linearize the state space model
  (b) Design a PID lateral and PID longitudinal controller
- P2 (a) Check the controllability and stabilizability of the linearized system
  (b) Design a lateral full-state feedback controller
- P3 (a) Design an lateral optimal controller
  (b) Implement A* path planning algorithm
- P4 (a) Implement EKF SLAM to control the vehicle without default sensor input
  (b) Race with other Buggy competitors in the class

2 Model

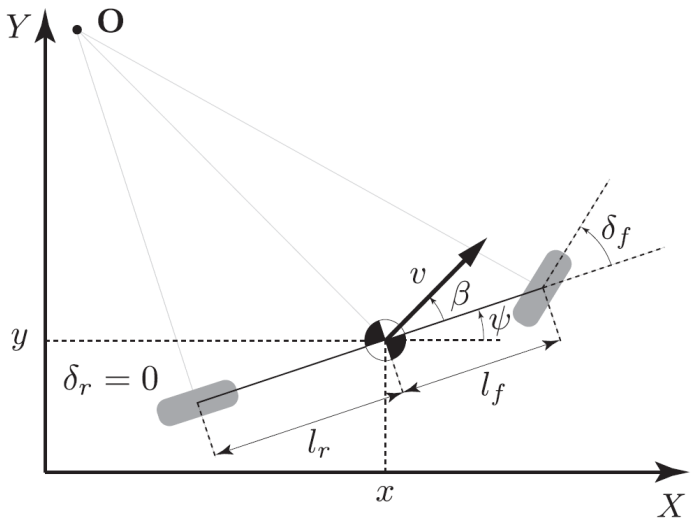

Figure 1: Bicycle model [2]

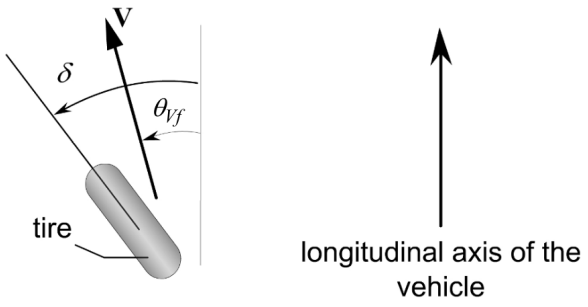

Figure 2: Tire slip-angle [2]

We will make use of a bicycle model for the vehicle, which is a popular model in the study of vehicle dynamics. Shown in Figure 1, the car is modeled as a two-wheel vehicle with two degrees of freedom, described separately in longitudinal and lateral dynamics. The model parameters are defined in Table 2.

2.1 Lateral dynamics

Ignoring road bank angle and applying Newton's second law of motion along the y-axis:

$$ma_y = F_{yf} \cos \delta_f + F_{yr}$$

where $a_y = \left(\frac{d^2 y}{dt^2} \right)_{inertial}$ is the inertial acceleration of the vehicle at the center of geometry in the direction of the y axis, F_{yf} and F_{yr} are the lateral tire forces of the front and rear

wheels, respectively, and δ_f is the front wheel angle, which will be denoted as δ later. Two terms contribute to a_y: the acceleration $\ddot{y}$, which is due to motion along the y-axis, and the centripetal acceleration. Hence:

$$a_y = \ddot{y} + \dot{\psi}\dot{x}$$

Combining the two equations, the equation for the lateral translational motion of the vehicle is obtained as:

$$\ddot{y} = -\dot{\psi}\dot{x} + \frac{1}{m}(F_{yf}\cos\delta + F_{yr})$$

Moment balance about the axis yields the equation for the yaw dynamics as

$$\ddot{\psi} I_z = l_f F_{yf} - l_r F_{yr}$$

The next step is to model the lateral tire forces F_{yf} and F_{yr}. Experimental results show that the lateral tire force of a tire is proportional to the "slip-angle" for small slip-angles when vehicle's speed is large enough - i.e. when $\dot{x} \geq 0.5$ m/s. The slip angle of a tire is defined as the angle between the orientation of the tire and the orientation of the velocity vector of the vehicle. The slip angle of the front and rear wheel is

$$\alpha_f = \delta - \theta_{Vf}$$
$$\alpha_r = -\theta_{Vr}$$

where θ_{Vp} is the angle between the velocity vector and the longitudinal axis of the vehicle, for $p \in \{f, r\}$. A linear approximation of the tire forces are given by

$$F_{yf} = 2C_\alpha \left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{\dot{x}}\right)$$
$$F_{yr} = 2C_\alpha \left(-\frac{\dot{y} - l_r\dot{\psi}}{\dot{x}}\right)$$

where C_α is called the cornering stiffness of the tires. If $\dot{x} < 0.5$ m/s, we just set F_{yf} and F_{yr} both to zeros.

2.2 Longitudinal dynamics

Similarly, a force balance along the vehicle longitudinal axis yields:

$$\ddot{x} = \dot{\psi}\dot{y} + a_x$$
$$ma_x = F - F_f$$
$$F_f = fmg$$

where F is the total tire force along the x-axis, and F_f is the force due to rolling resistance at the tires, and f is the friction coefficient.

2.3 Global coordinates

In the global frame we have:

$$\dot{X} = \dot{x}\cos\psi - \dot{y}\sin\psi$$
$$\dot{Y} = \dot{x}\sin\psi + \dot{y}\cos\psi$$

2.4 System equation

Gathering all of the equations, if $\dot{x} \geq 0.5$ m/s, we have:

$$\ddot{y} = -\dot{\psi}\dot{x} + \frac{2C_\alpha}{m}(\cos\delta\left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{\dot{x}}\right) - \frac{\dot{y} - l_r\dot{\psi}}{\dot{x}})$$
$$\ddot{x} = \dot{\psi}\dot{y} + \frac{1}{m}(F - fmg)$$
$$\ddot{\psi} = \frac{2l_fC_\alpha}{I_z}\left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{\dot{x}}\right) - \frac{2l_rC_\alpha}{I_z}\left(-\frac{\dot{y} - l_r\dot{\psi}}{\dot{x}}\right)$$
$$\dot{X} = \dot{x}\cos\psi - \dot{y}\sin\psi$$
$$\dot{Y} = \dot{x}\sin\psi + \dot{y}\cos\psi$$

otherwise, since the lateral tire forces are zeros, we only consider the longitudinal model.

2.5 Measurements

The observable states are:

$$y = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ X \\ Y \\ \psi \end{bmatrix}$$

2.6 Physical constraints

The system satisfies the constraints that:

$$|\delta| \leqslant \tfrac{\pi}{6}\ rad$$
$$F \geqslant 0 \text{ and } F \leqslant 15736\ N$$
$$\dot{x} \geqslant 10^{-5}\ m/s$$

Table 1: Model parameters.

Name	Description	Unit	Value
$(\dot{x}, \dot{y})$	Vehicle's velocity along the direction of vehicle frame	m/s	State
(X, Y)	Vehicle's coordinates in the world frame	m	State
ψ, $\dot{\psi}$	Body yaw angle, angular speed	rad, rad/s	State
δ or δ_f	Front wheel angle	rad	State
F	Total input force	N	Input
m	Vehicle mass	kg	1888.6
l_r	Length from rear tire to the center of mass	m	1.39
l_f	Length from front tire to the center of mass	m	1.55
C_α	Cornering stiffness of each tire	N	20000
I_z	Yaw intertia	kg mˆ2	25854
F_{pq}	Tire force, $p \in \{x, y\}$,$q \in \{f, r\}$	N	Depends on input force
f	Rolling resistance coefficient	N/A	0.019
`delT`	Simulation timestep	sec	0.032

3 Resources

3.1 Simulation

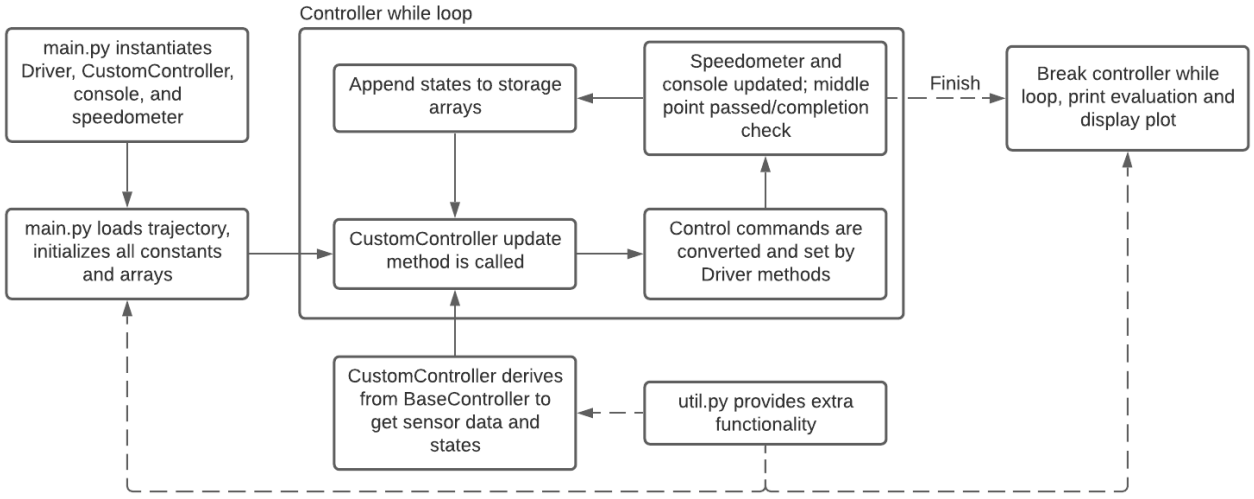

Figure 3: Simulation code flow

Several files are provided to you within the `controllers/main` folder. The `main.py` script initializes and instantiates necessary objects, and also contains the controller loop. This loop runs once each simulation timestep. `main.py` calls `your_controller.py`'s `update` method on each loop to get new control commands (the desired steering angle, δ, and longitudinal force, F). The longitudinal force is converted to a throttle input, and then both control commands are set by Webots internal functions. The additional script `util.py` contains functions to help you design and execute the controller. The full codeflow is pictured in Figure 3.

Please design your controller in the `your_controller.py` file provided for the project part you're working on. Specifically, you should be writing code in the `update` method. Please **do not** attempt to change code in other functions or files, as we will only grade the relevant `your_controller.py` for the programming portion. However, you are free to add to the `CustomController` class's `__init__` method (which is executed once when the `CustomController` object is instantiated).

3.2 BaseController Background

The `CustomController` class within each `your_controller.py` file derives from the `BaseController` class in the `base_controller.py` file. The vehicle itself is equipped with a Webots-generated GPS, gyroscope, and compass that have no noise or error. These sensors are started in the `BaseController` class, and are used to derive the various states of the vehicle. An explanation on the derivation of each can be found in the table below.

Table 2: State Derivation.

Name	Explanation
(X, Y)	From GPS readings
$(\dot{x}, \dot{y})$	From the derivative of GPS readings
ψ	From the compass readings
$\dot{\psi}$	From the gyroscope readings

3.3 Trajectory Data

The trajectory is given in `buggyTrace.csv`. It contains the coordinates of the trajectory as (x, y). The satellite map of the track is shown in Figure 4.

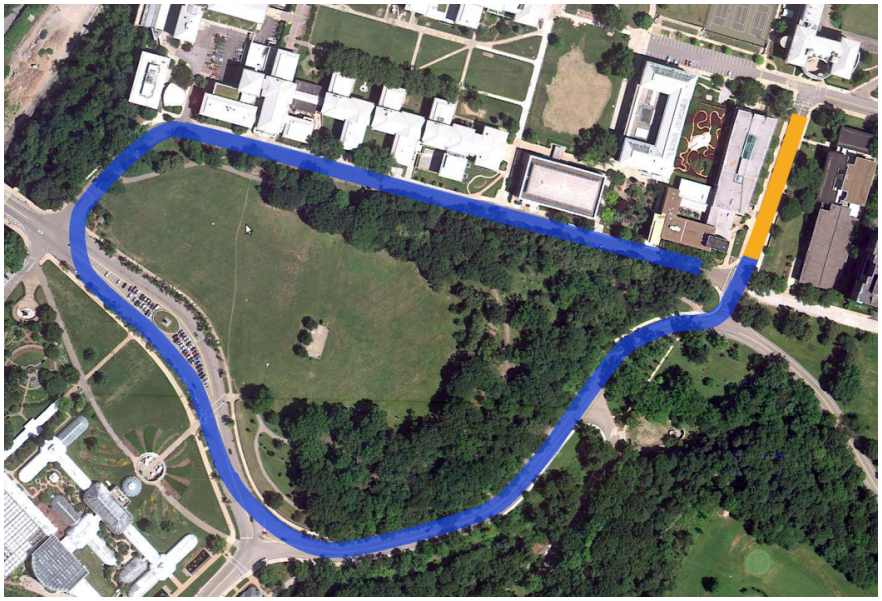

Figure 4: Buggy track[3]

4 P1: Problems

Exercise 1. Model Linearization. As mentioned in class, model linearization is always the first step for non-linear control. During this assignment, you will approximate the given model with a linear model.

Since the longitudinal term $\dot{x}$ is non-linear in the lateral dynamics, we can simplify the controller by controlling the lateral and longitudinal states separately. You are required to write the system dynamics in linear forms as $\dot{s_1} = A_1 s_1 + B_1 u$ and $\dot{s_2} = A_2 s_2 + B_2 u$ in terms of the following given input and states:

$$u = \begin{bmatrix} \delta \\ F \end{bmatrix}, s_1 = \begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix}, s_2 = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

Assume that for values of $\dot{x} < 0.5$ m/s that the system is identical, i.e. there is no need to linearize two separate systems.

Exercise 2. Controller Synthesis in Simulation. The driver functions that control the car take the desired steering angle δ and a throttle input - ranging from 0 to 1 - which is derived from the desired longitudinal force F.

For this question, you have to design a PID longitudinal controller and a PID lateral controller for the vehicle. A PID is an error-based controller that requires tuning proportional, integral, and derivative gains. As a PID allows us to minimize the error between a set point and process variable without deeper knowledge of the system model, we will not need our result from Exercise 1 (though it will be useful in future project parts).

Design the two controllers in `your_controller.py`. You can make use of Webots' built-in code editor, or use your own.

Check the performance of your controller by running the Webots simulation. You can press the play button in the top menu to start the simulation in real-time, the fast-forward button to run the simulation as quickly as possible, and the triple fast-forward to run the simulation without rendering (any of these options is acceptable, and the faster options may be better for quick tests). If you complete the track, the scripts will generate a performance plot via `matplotlib`. This plot contains a visualization of the car's trajectory, and also shows the variation of states with respect to time.

Submit `your_controller.py` and the final completion plot as described on the title page. Your controller is **required** to achieve the following performance criteria to receive full points:

1. Time to complete the loop = 400 s
2. Maximum deviation from the reference trajectory = 10.0 m
3. Average deviation from the reference trajectory = 5 m

Some hints that may be useful:

- Using a PID controller requires storing variables between method calls. Python allows this through use of the `self` preface (in other words, making them class member variables). Think about which variables you use in your controller that you will need to store, and be sure to add them to `CustomController`'s `__init__` method so they are initialized.
- If you are tuning your lateral controller for a point on the trajectory that is the closest to the vehicle, the controller may struggle on sudden turns. Think about how to add some mechanism that "looks ahead" of your current position before calculating a control command.
- Functions in `util.py` might be useful. For example, `closestNode` returns the absolute value of the cross-track error, which is the distance from the vehicle's center of gravity to the nearest waypoint on the trajectory. In addition, the function also returns the index of the closest waypoint to the vehicle.

- If your car does not perform well, it may just be that it requires tuning. You may already have some experience tuning PID controllers, but if not, viewing online resources is recommended. See this Wiki link [4] for more background in the process of manual tuning.

5 Reference

1. Rajamani Rajesh. Vehicle Dynamics and Control. Springer Science & Business Media, 2011.

2. Kong Jason, et al. "Kinematic and dynamic vehicle models for autonomous driving control design." Intelligent Vehicles Symposium, 2015.

3. cmubuggy.org, https://cmubuggy.org/reference/File:Course_hill1.png

4. "PID Controller - Manual Tuning." *Wikipedia*, Wikimedia Foundation, August 30th, 2020. https://en.wikipedia.org/wiki/PID_controller#Manual_tuning