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 R2021b.
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 R2021b.
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 R2021b.
2. In a terminal window, make sure you have `numpy`, `scipy`, and `matplotlib` with the following command:
   `pip install <package-name>`.

Buggy is a time-honored CMU tradition (you can learn more about it here and here). 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

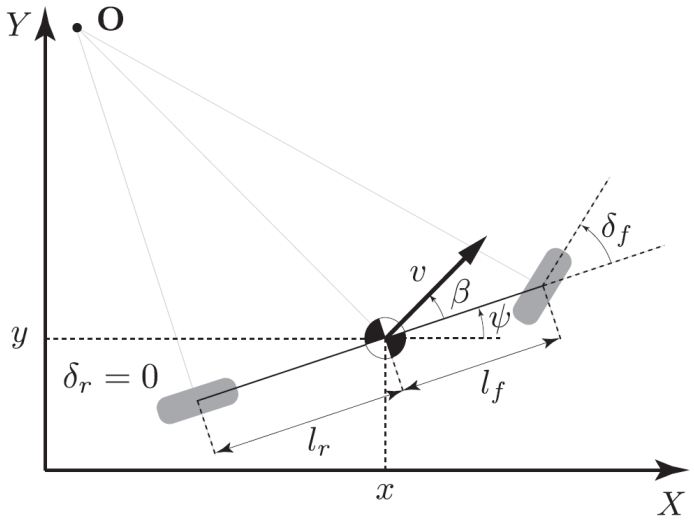

Figure 1: Bicycle model [2]

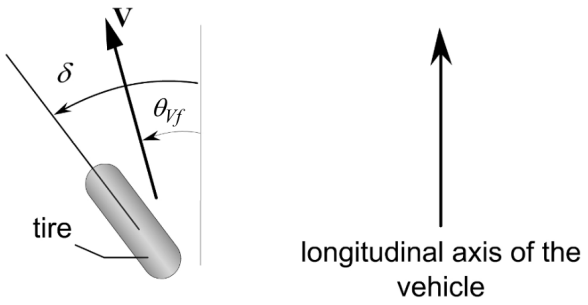

Figure 2: Tire slip-angle [2]

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	Input
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

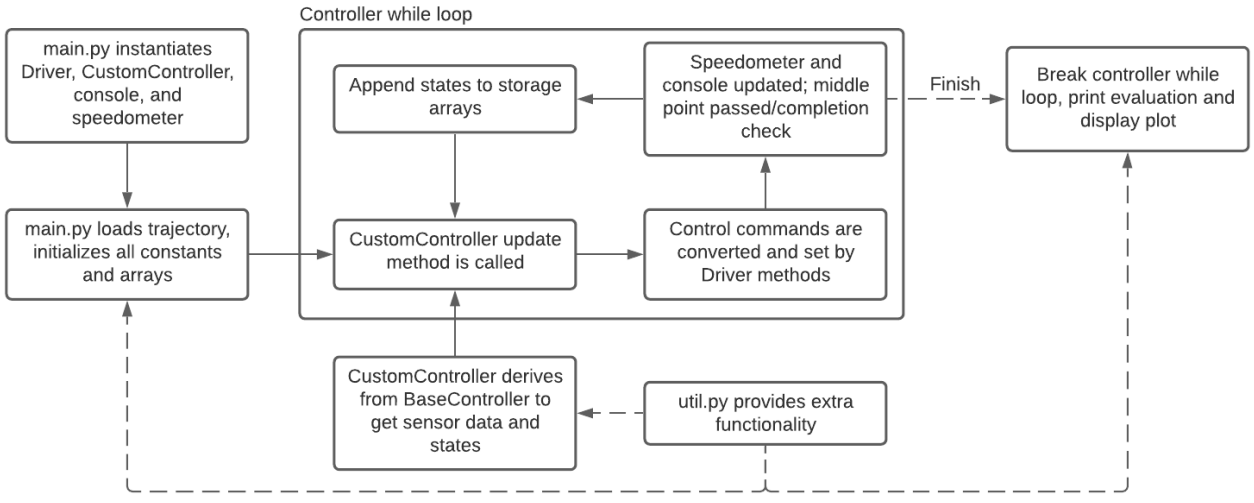

Figure 3: Simulation code flow

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.