ROB 599 Controls Project: Racing on a Pre-Defined Map with Unknown Obstacles

December 7, 2017

1 Introduction

In this project, you will work in groups and be asked to control a bicycle model. The goal is to complete a pair of tasks: first, you will design a controller for the system to get from the beginning to the end of a pre-defined track as rapidly as possible; second, you will develop a control design algorithm (which may or may not modify the controller constructed in the first task) to avoid obstacles that are known only at run-time.

The project is due at 6:00PM on December 20th.

2 Deliverables and Evaluation

2.1 Deliverables

You will submit the controls portion of the project on Canvas. Your deliverables are a `.mat` file and a `.m` file called

```
ROB599_ControlsProject_part1_Team⟨your team number⟩.mat
ROB599_ControlsProject_part2_Team⟨your team number⟩.m
```

where you should replace ⟨your team number⟩ with the team number we give you. Do not include the brackets in your file name. In this file, you will submit a vector for the first part of the project, and a function for the second part of the project. These variable for part 1 should be named:

```
ROB599_ControlsProject_part1_input
```

See section 6 for more details about these deliverables. **NOTE: We will not load any variables from your `.mat` file except for this one.**

2.2 Qualifying and Final Code Evaluation

We will evaluate every team's code on the same computer for fairness. This process will be automated, so it is critical that you name your `.mat` file correctly. **Your function for part 2 will have 5 minutes to complete solving.**

You can submit your code to us any time before the deadline. We will have two **optional** "qualifying sessions"

- Q1: December 6th at 5:00 PM EST
- Q2: December 13th at 5:00 PM EST

On each of these dates, we will run all code that has been submitted so far, and post a leaderboard/results on Canvas.

In each of the qualifying sessions and in the final evaluation, **each submission will be run 5 times** for each of the project's two parts, and the mean track completion percentage and vehicle speed (for parts 1 and 2) and mean solve time (for part 2) will be reported. For part 2 of the project (with obstacles), every team will encounter the exact same randomly-generated obstacles. The final results will be posted after the final deadline.

3 Track

We will use a portion of the Circuit of the Americas in Austin, TX, which has hosted the Formula One United States Grand Prix since 2012. The GPS coordinates of the track were collected from Google Earth and converted to Cartesian coordinates. Altogether, 246 pairs of left and right boundary points were collected. The centerline was calculated by averaging the corresponding left and right boundary points. The track is plotted in Figure 1.

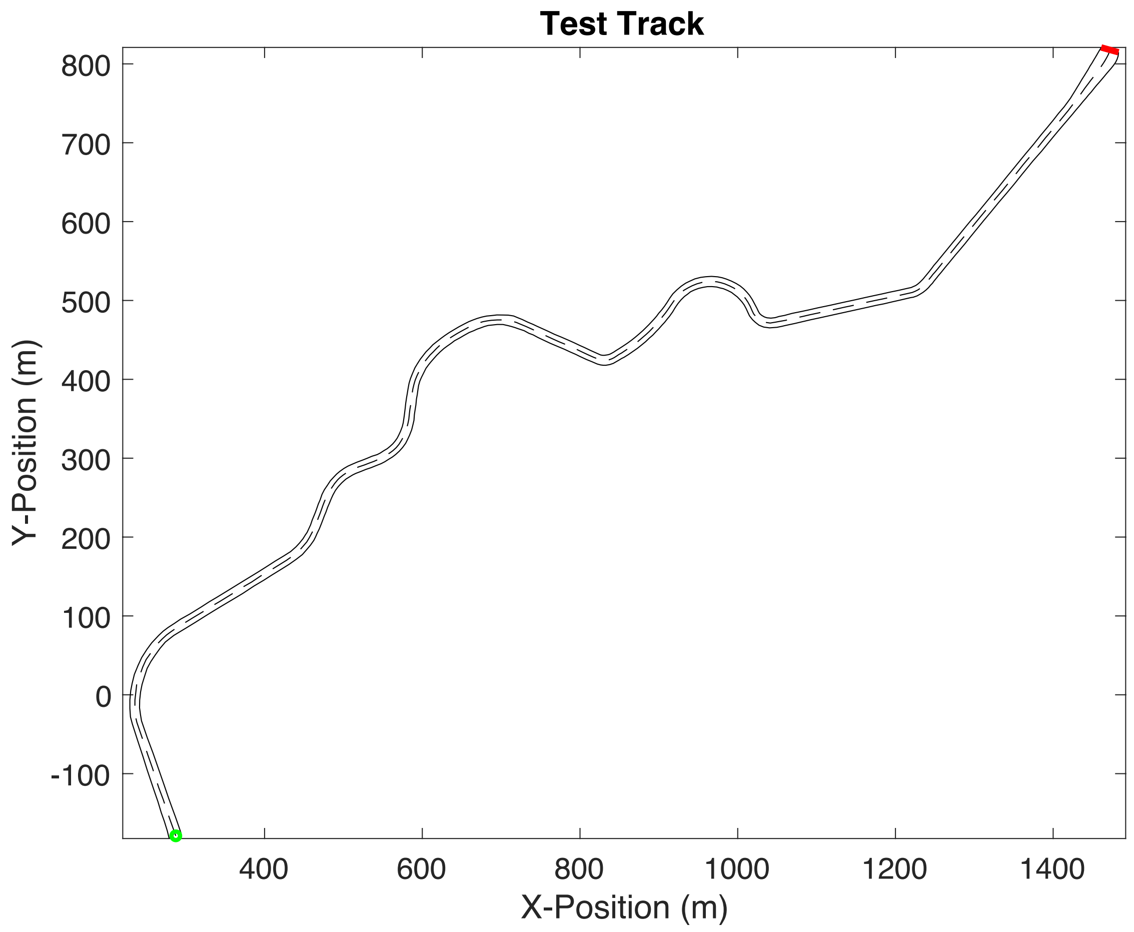

Figure 1: An illustration of the test track that you will design an autonomous vehicle to drive on. The initial position of the vehicle is shown by the green circle at $(287, -176)$ and your first task will be to drive the vehicle through the red line on the track as quickly as possible while staying within the track. Your second task will be to drive as quickly as possible through the track while safely avoiding obstacles that are only given at run-time.

4 Model

We will model the car with a non-linear bicycle model. As depicted in Figure 2, we choose the generalized coordinates of the center of mass (x,y) and the yaw angle ψ to define the vehicles dynamics:

$$\begin{bmatrix}\dot{x}\\ \dot{u}\\ \dot{y}\\ \dot{v}\\ \dot{\psi}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} u\cos\psi - v\sin\psi \\ \frac{1}{m}(-fW + N_w F_x - F_{yf}\sin(\delta_f)) + vr \\ u\sin\psi + v\cos\psi \\ \frac{1}{m}(F_{yf}\cos(\delta_f) + F_{yr}) - ur \\ r \\ \frac{1}{I_z}(aF_{yf}\cos(\delta_f) - bF_{yr}) \end{bmatrix}. \tag{1}$$

The lateral forces F_{yf} and F_{yr} are described using the Pacejka “Magic Formula:”

$$F_{yf} = D_y \sin(C_y \tan^{-1}(B_y \phi_{yf})) + S_{vy} \tag{2}$$

$$F_{yr} = D_y \sin(C_y \tan^{-1}(B_y \phi_{yr})) + S_{vy} \tag{3}$$

where

$$\phi_{yf} = (1 - E_y)(\alpha_f + S_{hy}) + \frac{E_y}{B_y}\tan^{-1}(B_y(\alpha_f + S_{hy})) \tag{4}$$

$$\phi_{yr} = (1 - E_y)(\alpha_r + S_{hy}) + \frac{E_y}{B_y}\tan^{-1}(B_y(\alpha_r + S_{hy})) \tag{5}$$

where α_f and α_r are the front and rear lateral slip angles which are given in **degrees** in the previous formulas. The front and rear lateral slip angles which is described in **radians** is given by:

$$\alpha_f = \delta_f - \tan^{-1}(\frac{v + ar}{u}) \tag{6}$$

$$\alpha_r = -\tan^{-1}(\frac{v - br}{u}) \tag{7}$$

The inputs into this model are δ_f, which is the front wheel steering angle; and F_x, which is the traction force generated at each tire by the vehicle’s motor. The vehicle begins from the following initial condition:

$$\begin{bmatrix}x\\ u\\ y\\ v\\ \psi\\ r\end{bmatrix} = \begin{bmatrix}287\ [\text{m}]\\ 5\ [\text{m/s}]\\ -176\ [\text{m}]\\ 0\ [\text{m/s}]\\ 2\ [\text{rad}]\\ 0\ [\text{rad/s}]\end{bmatrix} \tag{8}$$

All of the parameters of this model and limits on the input are listed in Table 1. **NOTE: though the car is assumed to have a non-zero wheel base while describing the dynamics, you are only required to check that the center of mass of the vehicle does not leave the track or run into any obstacles.**

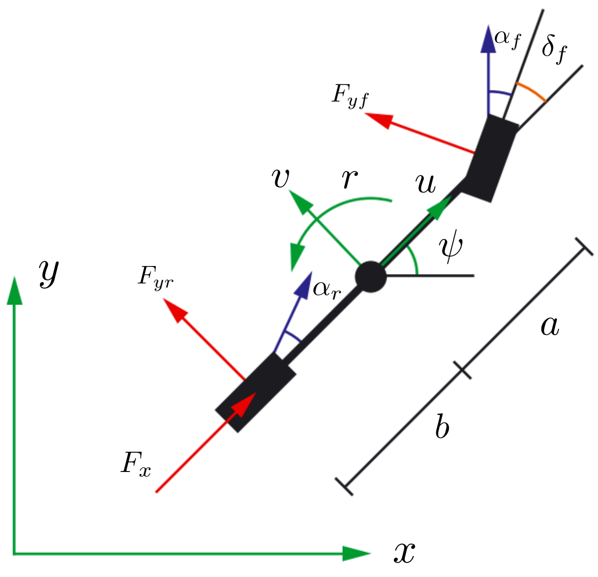

Figure 2: An illustration of the bicycle model used to define the vehicle's dynamics.

Vehicle Parameter	**Value**
δ	$[-0.5, 0.5]$
F_x	$[-10000, 5000]$
m	1400
W	13720
N_w	2
f	0.01
I_z	2667
a	1.35
b	1.45
B_y	0.27
C_y	1.2
D_y	2921
E_y	-1.6
S_{hy}	0
S_{vy}	0

Table 1: Datasheet for the Vehicle. Note: The Pacejka parameters are for the slip angle in **degrees**

5 Simulation Package

You will be given the following MATLAB files:

1. `TestTrack.mat`
2. `forwardIntegrateControlInput.m`
3. `generateRandomObstacles.m`
4. `checkTrajectory.p`

5.1 `TestTrack.mat`

The `TestTrack.mat` file contains the following fields that describe the track:

1. `TestTrack.bl` is a sequence of points describing the left boundary of the track.
2. `TestTrack.br` is a sequence of points describing the right boundary of the track.
3. `TestTrack.cline` is a sequence of points describing the centerline of the track.
4. `TestTrack.theta` is a sequence of points describing the centerline's orientation.

5.2 `forwardIntegrateControlInput.m`

The function `forwardIntegrateControlInput` is exactly the same forward integration method that we will use to check the validity of your control input and resulting trajectory. It is based on `ode45`, and assumes a zero order hold on your control inputs. It also assumes that your control inputs are defined every 0.01s. This function takes in a control input and initial condition and returns a trajectory in state space.

5.3 `generateRandomObstacles.m`

The function `generateRandomObstacles` is the same function that we will use to generate obstacles when checking your code for part 6.2. This function takes in an integer `Nobs`, the number of obstacles along the track, and returns `Xobs`, a cell array of size 1×`Nobs`, where each cell contains a 4×2 matrix describing each obstacle (the first column is each obstacle's x-coordinates and the second column is its y-coordinates). This function spaces the obstacles roughly evenly along the track. When we test your code, we will use between 10 and 25 obstacles.

5.4 `checkTrajectory.p`

The function `checkTrajectory` will tell you if your car leaves the track, crashes into any obstacles, or exceeds any input limits. It takes in your trajectory (as an $N \times 2$ vector, where the first column is the x coordinates of your trajectory and the second column is the y coordinates), your input (as an $N \times 2$ vector where the first column is δ and the second column is F_x), and a cell array of obstacles as an optional argument. The cell array of obstacles should be in the same format as the output of `generateRandomObstacles`. This function returns `false` if your car did not crash and did not exceed any input limits.

NOTE: Your trajectory counts as a crash if it is inside an obstacle or on the boundary of an obstacle. Your trajectory counts as leaving the track if it is *outside* the track boundaries, so the car is allowed to touch the boundary of the track.

You may want to write your own trajectory checking function that tells you more detailed information, such as where/when you crashed or exceeded input limits.

6 Challenge Rules

Recall that your deliverables for this project are a vector for part 1 and a function for part 2. The vector for part 1 will be used as follows:

```
sol_1 = forwardIntegrateControlInput(ROB599_ControlsProject_part1_input)
```

where `sol_1` is the vehicle's trajectory. The function will be called as follows:

```
sol_2 = ROB599_ControlsProject_part2_Team⟨your team number⟩.m(TestTrack,Xobs)
```

where `sol_2` is a vector of *control inputs* that will be passed to `forwardIntegrateControlInput`. The input `TestTrack` is the structure loaded from within `TestTrack.mat`, and the input `Xobs` is of the same format as the output of the `generateRandomObstacles` function.

Make sure that, if you are using any system commands (`cd`, `ls`, etc.) or plotting commands for testing purposes, they are *not* inside your function for part 2. If you call any such commands, or return anything besides the specified output, you will receive 0 points.

6.1 Control Design on Pre-Defined Map [60 points]

You are tasked to design a control input to make the car complete the track as quickly as possible. You will be scored according to whether you complete the track. Completion is defined as the center of mass passing the red line and not leaving the track at any point. If you complete the track you will receive 60 points towards your final grade. If the track is not completed then you will receive a grade proportional to the distance traveled.

In order for us to test your controller, you will provide your control input as an $n \times 2$ double in the following format:

$$\begin{bmatrix} \delta_1 & F_{x,1} \\ \delta_2 & F_{x,2} \\ \vdots & \vdots \\ \delta_n & F_{x,n} \end{bmatrix}$$

where n is the number of timesteps. The length of each timestep should be 0.01 seconds.

To evaluate your solution, we will take your input vector and use ode45 with a zero-order hold between timesteps, i.e. the control input between timesteps k and $k+1$ is the input from timestep k held constant over the entire 0.01 s between time steps. We are providing you with an integration function so that you can check your control inputs (see 5.2).

6.2 Real-Time Control Design with Unknown Obstacles [40 points]

You are again tasked to design a control input to get you across the track as quickly as possible. However, now obstacles will be randomly distributed on the track.

You will be scored according to the time it takes you to complete the track. Completion is defined the same as before: the center of mass passes the red line and does not leave the track or hit obstacles at any point. If you complete the track you will receive 40 points towards your project grade. If the track is not completed then you will receive a grade proportional to the distance traveled.

You will be handed a random number of randomly-generated obstacles in a 1×`Nobs` cell array, where `Nobs` is the number of obstacles. Each cell will contain a 4×2 matrix of doubles. Each row is an (x, y) coordinate of a vertex of the obstacle. From top to bottom, the rows give the vertices in counter clockwise order. The function we will use to generate obstacles is given to you (see 5.3).

You will have to provide a function that takes in the obstacle cell array and produces a vector of inputs. The inputs should be stored in the same format as 6.1. The solution will be evaluated with ode45 and a zero order hold as in 6.1.

7 Acknowledgements

Thank you to Gabor Orosz and Chaozhe He for creating the track.