ROB 535 Control Project: Racing on a Pre-Defined Map with Unknown Obstacles

December 5, 2021

1 Introduction

In this project, you will work in groups and be asked to control a bicycle model. Each group can have 4 to 6 members, and you should have already registered your groups on Canvas. The goal the control project 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 **4:00PM EST on December 16th**.

2 Deliverables and Evaluation

2.1 Deliverables

You will submit the control portion of the project on Canvas. Your deliverables include Matlab files and a documentation described below.

(a) Desired Matlab files include a `.mat` file and a `.m` file called

`ROB535_ControlProject_part1_Team⟨your team number⟩.mat`
`ROB535_ControlProject_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. For example, if you are Team 56, your should name your files:

`ROB535_ControlProject_part1_Team56.mat`
`ROB535_ControlProject_part2_Team56.m`

The variable for part 1, inside of the `.mat` file, should be named:

`ROB535_ControlProject_part1_input`

See section 6 for more details about these deliverables. **NOTE: We will ONLY load/run the two files we specified above. If you want us to load any other variables from your `.mat` file, or want to include any backup `.m` files, or need any other accomodation to run your code, you must send us an email. We will not load/run any of your backup files, but you can load/call them inside the `.m` file we specified above if needed.**

(b) Each team member needs to individually submit a short documentation briefly describing the work you did on this project and the work each of your teammates did. Your file can be named as

`ROB535_ControlProject_Team⟨your team number⟩_⟨your name⟩.pdf`

For example Jinsun Liu in Team 1 should name file as

`ROB535_ControlProject_Team1_JinsunLiu.pdf`

2.2 Qualifying and Final Code Evaluation

We will evaluate every team's code on the same computer for fairness using Matlab 2021a. **You are only allowed to use exactly the same toolboxes as you are allowed in Matlab Grader**, and permitted toolboxes are listed on the homepage of our course in Matlab Grader. This process will be automated, so it is **critical** that you name your `.mat` file and `.m` file correctly. You can submit your code to us any time before the deadline.

We will have one **mandatory** "qualifying session" on **December 2nd at 5:00 PM EST**. This qualifying session is for us to make sure your files have the right format. You are excepted to submit your Matlab deliverables only in this session, and they don't need to be the same as your final submission. **To encourage you to submit your code before the date above, we will award 5 points to your total grade. Therefore to achieve full credit, you must submit your code for the qualifying session.** On the date of qualifying session, we will run all code that has been submitted so far, and post a leaderboard/results on Canvas.

In the qualifying session and the final evaluation, each submission will be run **once** for part 1 and **five times** (with different obstacles) for part 2. 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 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.

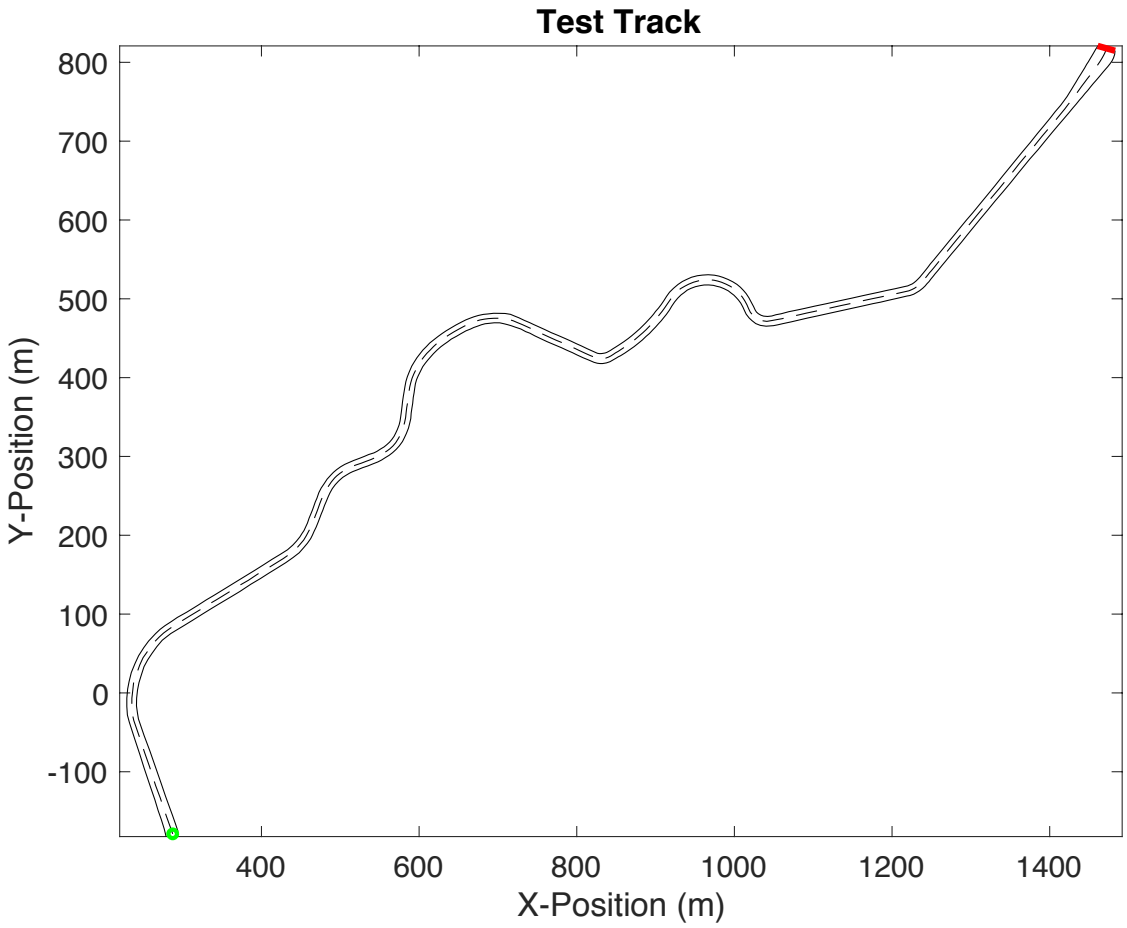

Figure 1: An illustration of the test track that you will design an autonomous vehicle to drive on. The initial position of the vehicle is shown by the green circle at $(287, -176)$ and your first task will be to drive the vehicle through the red line on the track as quickly as possible while staying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

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

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

$$F_{zf} = \frac{b}{a+b}mg \tag{2}$$

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

$$F_{zr} = \frac{a}{a+b}mg \tag{4}$$

$$F_{yr} = F_{zr} D_y \sin(C_y \tan^{-1}(B_y \phi_{yr})) + S_{vy} \tag{5}$$

where

$$\phi_{yf} = (1 - E_y)(\alpha_f + S_{hy}) + \frac{E_y}{B_y}\tan^{-1}(B_y(\alpha_f + S_{hy})) \tag{6}$$

$$\phi_{yr} = (1 - E_y)(\alpha_r + S_{hy}) + \frac{E_y}{B_y}\tan^{-1}(B_y(\alpha_r + S_{hy})) \tag{7}$$

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:

$$\alpha_f = \delta_f - \tan^{-1}(\frac{v + ar}{u}) \tag{8}$$

$$\alpha_r = -\tan^{-1}(\frac{v - br}{u}) \tag{9}$$

Additionally, combined longitudinal and lateral loading of tires will be limited to F_x^* and F_{yr}^* in the following manner:

$$F_{\text{total}} = \sqrt{(N_w F_x)^2 + (F_{yr})^2} \tag{10}$$

$$F_{\max} = 0.7mg \tag{11}$$

$$\tag{12}$$

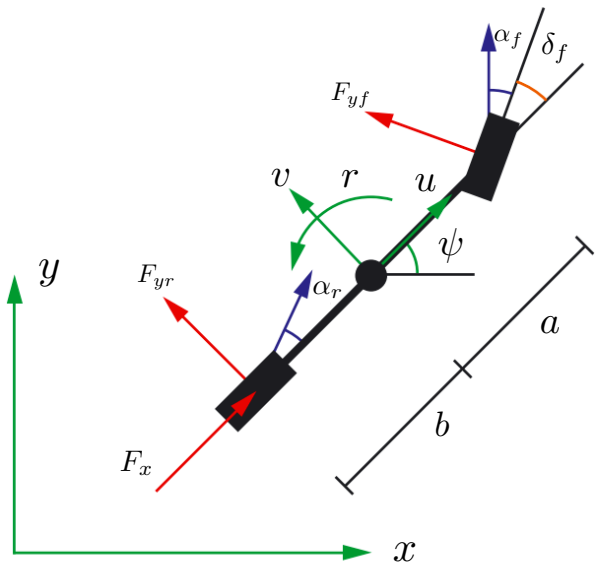

Figure 2: An illustration of the bicycle model used to define the vehicle's dynamics.

If: $F_{\text{total}} > F_{\max}$:

$$F_x^* = \frac{F_{\max}}{F_{\text{total}}} F_x \tag{13}$$

$$F_{yr}^* = \frac{F_{\max}}{F_{\text{total}}} F_{yr} \tag{14}$$

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:

$$\begin{bmatrix} x \\ u \\ y \\ v \\ \psi \\ r \end{bmatrix} = \begin{bmatrix} 287 \text{ [m]} \\ 5 \text{ [m/s]} \\ -176 \text{ [m]} \\ 0 \text{ [m/s]} \\ 2 \text{ [rad]} \\ 0 \text{ [rad/s]} \end{bmatrix} \tag{15}$$

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.**

Vehicle Parameter	Value
δ	$[-0.5, 0.5]$
F_x	$[-5000, 5000]$
m	1400
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	0.7
E_y	-1.6
S_{hy}	0
S_{vy}	0
g	9.806

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

5 Simulation Package

You are given the following MATLAB files:

1. `TestTrack.mat`
2. `forwardIntegrateControlInput.m`
3. `generateRandomObstacles.m`
4. `senseObstacles.m`
5. `forwardIntegrate.m`
6. `getTrajectoryinfo.p`

NOTE: while you are free to modify these files (eg. add plotting functions, etc) none of these changes will be reflected in the instructor's files, so make sure your deliverables take that into account.

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`. It assumes that your control inputs are defined **every 0.01 second**. This function takes in the 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 senseObstacles.m

The function `senseObstacles` is exactly the same sensing method we will use to test your controller for the second task (6.2). It takes in the current position of the car `curr_pos` and the cell array `Xobs` of all the obstacles generated from `generateRandomObstacles` and returns a subset of the cell array of obstacles, `Xobs_seen`, which represents all obstacles detected within a 150 meters radius centered at the vehicle center of mass. `Xobs_seen` has the same format as `Xobs`.

5.5 forwardIntegrate.m

The function `forwardIntegrate` is the script we will run to test your controller for the second task (6.2) by providing it information of obstacles it encounters within a sensing radius as it traverses the test track. Vehicle trajectory, control inputs which are generated using your `.m` file, and computational time will be returned.

5.6 getTrajectoryInfo.p

The function `getTrajectoryInfo` 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 a struct of information including when and where the first crash occurs, when and where the vehicle first leaves the track, the distance traveled along the track, and the time to complete the track.

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(ROB535_ControlProject_part1_input)
```

where `sol_1` is the vehicle's trajectory.
The function for part 2 will be called as follows:

```
[sol_2, FLAG_terminate] = ROB535_ControlProject_part2_Team⟨your team number⟩
                          (TestTrack,Xobs_seen,curr_state)
```

Outputs:

- `sol_2` is a vector of *control inputs* that will be passed to `forwardIntegrateControlInput`. As mentioned later in 6.2, `sol_2` must have enough control inputs with time step 0.01 seconds to forward integrate vehicle dynamics for the next 0.5 second.
- `FLAG_terminate` is a binary flag set by the function to indicate when to stop the simulation. Setting this flag to 1 at a planning iteration indicates that, after integrating forward vehicle dynamics using the control inputs from `sol_2` for 0.5 second, the vehicle would have reached the goal. If this flag is set to 0, then it implies that the car would not reach the goal after forward integrating for 0.5 second. Note that, in the event that `FLAG_terminate` is never set to 1, the simulation would stop after 20 minutes so that we can process all the teams files in a timely manner.

Inputs:

- `TestTrack` is the structure loaded from `TestTrack.mat`.
- `Xobs_seen` is the output of the `senseObstacles` function.
- `curr_state` is taken from the last state in `Y`, where `Y` is the actual trajectory updated after forward integrating the vehicle dynamics using control inputs at every planning iteration.

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 [55 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 55 points towards your final grade. If the track is not completed then you will receive a grade proportional to the distance traveled. We suggest you try to make your control input smooth, since sharp, instantaneous changes in control input may result in large overshoot in the system response and affect the vehicle performance.

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

$$\begin{bmatrix} \delta_1 & F_{x,1} \\ \delta_2 & F_{x,2} \\ \vdots & \vdots \\ \delta_n & F_{x,n} \end{bmatrix} \tag{16}$$

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

To evaluate your solution, we will be forward integrating the car with the control inputs provided in `ROB535_ControlProject_part1_input` and initial condition (15). An integration function is provided so that you can check your control inputs (see 5.2).

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

We recommend you refer to the `forwardIntegrate.m` script in the `SimPkg` zip file provided on Canvas while reading through this task.

You are again tasked to design control inputs to get you across the track as quickly as possible. However, now obstacles will be randomly distributed on the track. To emulate a real scenario, positions of all obstacles are not given beforehand. Instead, obstacles will only be sensed by the car within a 150 meters sensing radius as it traverses the track at run-time. Therefore receding horizon control will be required where your function will consider a planning horizon of 0.5 seconds, i.e. your function will be called repeatedly and need to provide control inputs for the next 0.5 seconds of the simulation. Again, we suggest you try to make your control input smooth, since sharp, instantaneous changes in control input may result in large overshoot in the system response and affect the vehicle performance.

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.

Aside from the project points, teams will be ranked according to the following criteria:

- Completion of the track.

- Total time taken to complete the task.
- Solving for a set of control inputs at each planning iteration in less than 0.5 seconds.

The last criteria comes from the motivation of running this on a real autonomous vehicle, which requires that you can solve for control inputs within a pre-defined time limit (0.5 seconds for this project). Failure to do so could lead to fatal crashes in the real world, thus we will penalize solutions which take longer than 0.5 seconds to compute. An example of a metric we may use to find team rankings is as follows:

$$t\_score = t\_total + M \cdot max(num\_exceed\_time\_limit, 0) \tag{17}$$

where *t_total* is the total time taken to complete the test track, *num_exceed_time_limit* is the number of times your function took longer than 0.5 seconds to produce a solution at a planning step, M is some penalty value for taking longer than 0.5 seconds to produce a solution. The smaller *t_score* is, the better your rank.

NOTE: The penalty applies only to the team rankings for the final scoreboard and will NOT affect your points for the final project. Completing the track will secure full credit for this part of the project. For example, say Team 1 finishes the track in 4 minutes, but takes longer than 0.5 seconds to plan a path 3 times during the simulation; Team 2 finishes in 4 minutes 30 seconds, but plans in less than 0.5 seconds at every planning step; Team 3 gets 98% of the way in 2 minutes, but terminates the simulation early (i.e. setting `FLAG_terminate` to 1 too early). Then Teams 1 and 2 will get full credit, while Team 3 will get 98% of full credit. However on the leaderboard, Team 2 will be ranked 1st, followed by Team 1 and then Team 3.

At each planning iteration, you will be handed all the observed obstacles `Xobs_seen` in a $1 \times n$ cell array, where $n \in [0, \cdots, \texttt{Nobs}]$ and `Nobs` is the total number of obstacles. The functions we use to generate obstacles are given to you (see 5.3 and 5.4).

You will have to provide a function that takes in the test track, cell array of observed obstacles, and current state of the car at that time instant and produce a vector of control inputs. The control inputs should be stored in the same format as in (16).

7 Acknowledgements

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