ROB 535 Control Project Team26 Documentation

Shuoqi Wang
shuoqi@umich.edu

1 Task Definition

The control project of controlling a bicycle model mainly contains two parts as follows.

- **Task 1** We are required to design a controller for the system to get from the beginning to the end of a pre-defined track as rapidly as possible;
- **Task 2** We need to 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.

2 My Contribution

I was mainly working on solving the control tasks using Model Predictive Control (MPC) method. For task1, I developed the jacobian matrices A,B of the linearized system in discrete time for MPC, for task2, I was focusing on how to decrease the impact of object avoidance on MPC tracking. Since other group members got a better result in PID control, this method is abandoned.

2.1 Task 1

Due to the complexity of Pacejka "Magic Formula", it was hard to calculate the jacobian matrix of the non-linear bicycle model by hand. Therefore, I used the MATLAB to obtain the matrix A,B with the jacobian() function, and convert the symbolic result to function handle using matlabFunction(). In this case, Ai, Bi are obtained, where different input and state values can be taken into the matrix at each iteration. The example code is shown in figure2. With Ai, Bi, the MPC can have a relative good result on task 1 after tuning as shown in figure1.

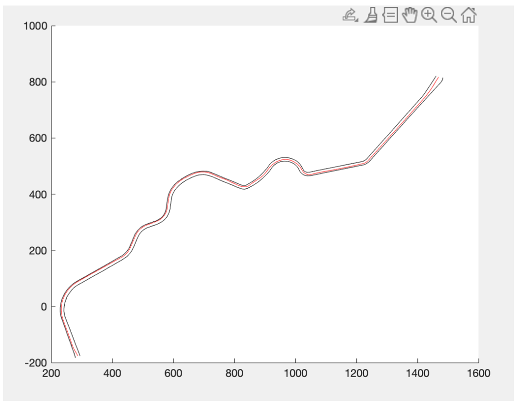
2.2 Task 2

The initial strategy for task 2 is that the car would follow either the right or the left boundary of the track, and whenever the car detects there is an obstacle in front, it would change the line to follow. Since the distance between two obstacles are unknown and the input to change the track is missing, the MPC didn't have a very good performance on task 2. After tests, it was found that the object avoidance has a great impact on MPC tracking. With less time and distance to change the route, the MPC tracking could perform better but the car couldn't effectively avoid all obstacles. Conversely, given more time and distance to change the route, the car had a nice performance on obstacles avoidance but would be out of boundary of the track. An example is shown in figure3. One guess for this issue is that the MPC heavily depend on the reference trajectory and input. Since we didn't have a fine reference input for the part where the line changes, the MPC couldn't perform as we expected.

3 Teammates' Work

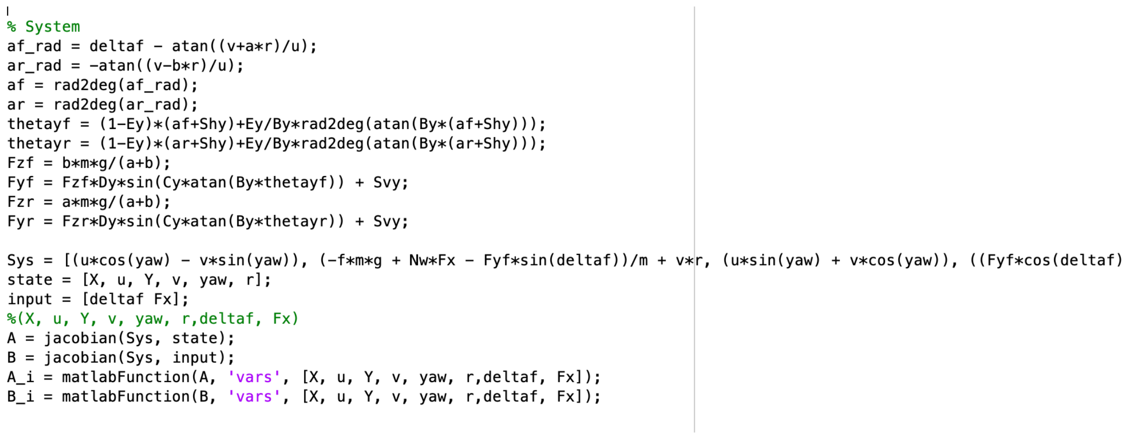
- **Zimin Lu** Control Task: mainly focus on task1. Generate the control input matrix using PID methods successfully.
- **Chengtian Zhang** Control Task: Explore MPC method for task1 and calculate the Jacobin matrix by hand. Generate the reference lane for task2.

- **Jiachen Jiang** Control Task: mainly focus on task2, successfully completed task2 using PID.
- **Lingfei Huo** Perception Task.



```
                                  Y: [11818×6 double]
                                  U: [11818×2 double]
                         t_finished: []
                              t_end: 118.1700
                left_track_position: []
                    left_track_time: []
   left_percent_of_track_completed: []
                     crash_position: []
                         crash_time: []
  crash_percent_of_track_completed: []
                     input_exceeded: [0 0]
        percent_of_track_completed: 1
                            t_score: []
```

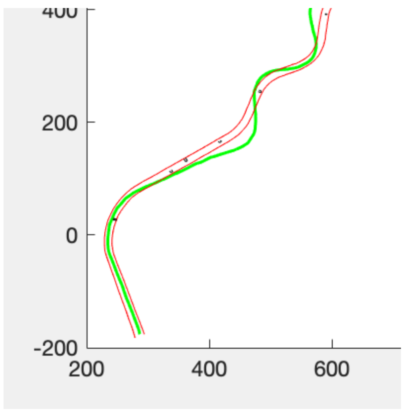
Figure 1: Task1 result with MPC



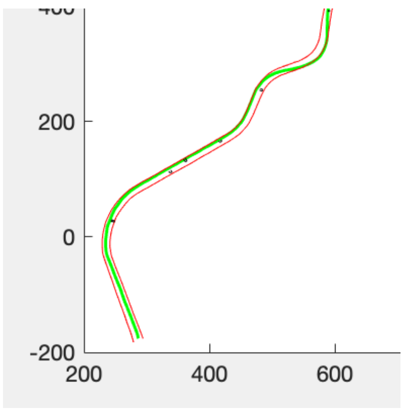
```
% System
af_rad = deltaf - atan((v+a*r)/u);
ar_rad = -atan((v-b*r)/u);
af = rad2deg(af_rad);
ar = rad2deg(ar_rad);
thetayf = (1-Ey)*(af+Shy)+Ey/By*rad2deg(atan(By*(af+Shy)));
thetayr = (1-Ey)*(ar+Shy)+Ey/By*rad2deg(atan(By*(ar+Shy)));
Fzf = b*m*g/(a+b);
Fyf = Fzf*Dy*sin(Cy*atan(By*thetayf)) + Svy;
Fzr = a*m*g/(a+b);
Fyr = Fzr*Dy*sin(Cy*atan(By*thetayr)) + Svy;

Sys = [(u*cos(yaw) - v*sin(yaw)), (-f*m*g + Nw*Fx - Fyf*sin(deltaf))/m + v*r, (u*sin(yaw) + v*cos(yaw)), ((Fyf*cos(deltaf)
state = [X, u, Y, v, yaw, r];
input = [deltaf Fx];
%(X, u, Y, v, yaw, r,deltaf, Fx)
A = jacobian(Sys, state);
B = jacobian(Sys, input);
A_i = matlabFunction(A, 'vars', [X, u, Y, v, yaw, r,deltaf, Fx]);
B_i = matlabFunction(B, 'vars', [X, u, Y, v, yaw, r,deltaf, Fx]);
```

Figure 2: Solving Jacobian Matrix using MATLAB



(a) Long time to change route with MPC



(b) Short time to change route with MPC

Figure 3: Comparison of tracking performance of car given a long vs short time to change route with MPC