ROB 535 Control Project Team26 Documentation

Zimin Lu
ziminlu@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 participated both Task1 and Task2, but I mainly focused on Task1.

2.1 Task1

I mainly tried 3 kinds of algorithms to complete Task1. Finally, I succeed to complete the Task1 using PID control.

2.1.1 Trajectory Synthesis

I first tried trajectory synthesis algorithm because Task1 is a tracking problem with track boundary constraints, and no reference input and reference trajectory are given.

The trajectory synthesis algorithm mainly includes four parts. The first is the upper and lower bounds correspond to the left and right boundaries of the track and the limits of other states together with the inputs. The second is the nonlinear constraint function, which can be used to avoid obstacles, which is not used in Task1. The third is the cost function, which mainly applies a quadratic penalty based on the distance to the goal in Task1. Fourth, solve the input and the trajectory using fmincon.

However, great difficulties are encountered when using this algorithm. The first is the construction of the upper and lower boundaries. The track is curved and slender, not a whole rectangle like the homework problem, so I have to divide the track into little rectangles. However, these rectangles cannot make full use of the track area. One method is to interpolate the track data for more detailed segmentation, but there is still a large gap in the density of the track points, resulting in a sharp change in the speed of the car which cannot be realized using a limited input. Moreover, interpolation makes the fmincon even slower. The low speed of the fmincon makes the debugging inefficient (it may be related to the nonlinearity of the system). In fact, in order to verify the practicability of this algorithm, I replaced the upper and lower boundaries with one rectangle, hoping to get a straight line but the solution is still slow. And I noticed that if you change the value of dt a little, you may not reach the end point. Overall, because the trajectory synthesis algorithm is pure local and has no feedback, it needs more fine tuning for complex tracks.

2.1.2 MPC Control

Since the trajectory synthesis algorithm does not work well, I tried MPC algorithm. However, MPC algorithm needs reference input and reference trajectory. In order to verify the feasibility of MPC in tracking long reference trajectory, we manually obtained a reference. After adjusting

Q and R, the bike can move along the reference trajectory. However, the manual reference has actually completed Task1, so the MPC verification is mainly to prepare for Task2. A difficult problem in MPC algorithm is linearization. We calculated the Jacobian matrix by hand. In order to check whether it is correct, I wrote two functions (Fig. 1 and Fig. 2), wrote the system as composite functions, and solved it by using the Jacobian function of MATLAB, which corrected the error of the matrix by hand (The reason why this function file is not called directly in MPC is that it takes longer time).

2.1.3 PID Control

Since neither trajectory synthesis nor MPC achieved the desired results, I turned to other methods. My new idea is to make the direction of the front wheel be the same as the current direction ψ of the center line. I used the nearest point's direction on the center line from the current position of the bike as a reference, but found that the bike often turned too late, so I used the next point from the nearest point as the reference point, but then it turned too early, so I took the average value of the nearest point and the next point as a reference, which perfectly achieved the goal. However, the bike moves slowly with the input of a fixed $F_x = 100$. My goal is that the car can turn at a lower speed and move forward quickly on the straight road, just like a racing car. Therefore, I have improved this algorithm many times. First, I found that the track data points are sparse on the straight track and dense at the corner (Fig. 3), so I make F_x roughly inversely proportional to the distance between two points of the track near the current position. In this way, at the corner, because the track is dense, F_x is small, it is not easy to rush out of the track. The average speed is improved by this method. However, the brake has still not been introduced, that is, $F_x > 0$ all the time. And this method depends on the density of the track, which may not be applicable in the trajectory generated by Task2 for obstacle avoidance.

Therefore, I linked the speed of the bike with the curvature of the track. To judge the curvature of the track in front, I used the standard deviation (STD) of the direction of the center line at several points in front of the bike (1), and then took the exponential of e, and then took the reciprocal to get the ratio(2), which can ensure that the ratio is greater than zero and less than or equal to 1. In this way, when the track is almost straight, STD is close to zero, the ratio is close to 1, $v_{desired}(k+1) \approx v_{max}$(3). When the track bends, STD is large, and then the ratio becomes smaller and $v_{desired}(k+1)$ becomes smaller. Then, using the relationship between F_x and velocity(4), F_x (Fig. 4) is obtained(5).

K_p (Fig. 5) is designed to control δ_f in order to enhance the tracking performance after the bike deviates from the center line. Experiments show that even if the bike deviates far from the center line, it can still return to it quickly.

In order that the bike does not rush out of the track at a high speed, a series of advance quantities are added for turning and speed change by changing the weight of different reference points. Finally, the time for the car to finish the track was reduced from 326.7 seconds to 85.2 seconds(Fig. 6). Acceleration and deceleration can be carried out as expected under limited conditions (Fig. 7).

$$STD(\psi) = \sqrt{\frac{1}{5-1}\sum_{i=0,1,3,5,7}(\psi(idx+i)-\mu)^2}$$
$$\mu = \frac{\sum_{i=0,1,3,5,7}\psi(idx+i)}{5} \tag{1}$$

Where idx is the index of the nearest point on the center line from the current position of the bike.

$$ratio = \sqrt{e^{(-STD(\psi))^{\frac{1}{1.7}}}} \tag{2}$$

So, $0 < ratio \leq 1$.

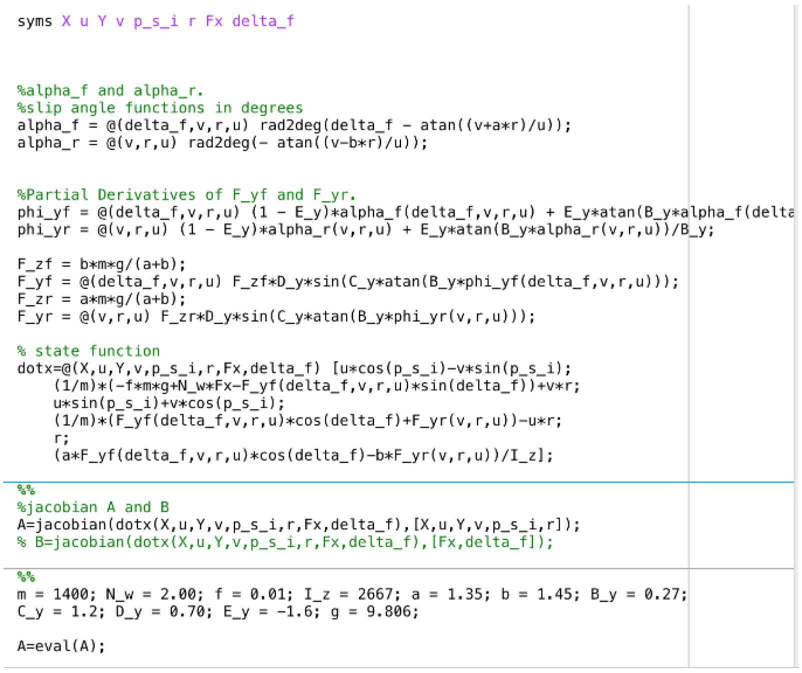
Set the desired the velocity,

$$v_{desired}(k+1) = v_{max} * ratio \tag{3}$$

According to the relationship between force and velocity,

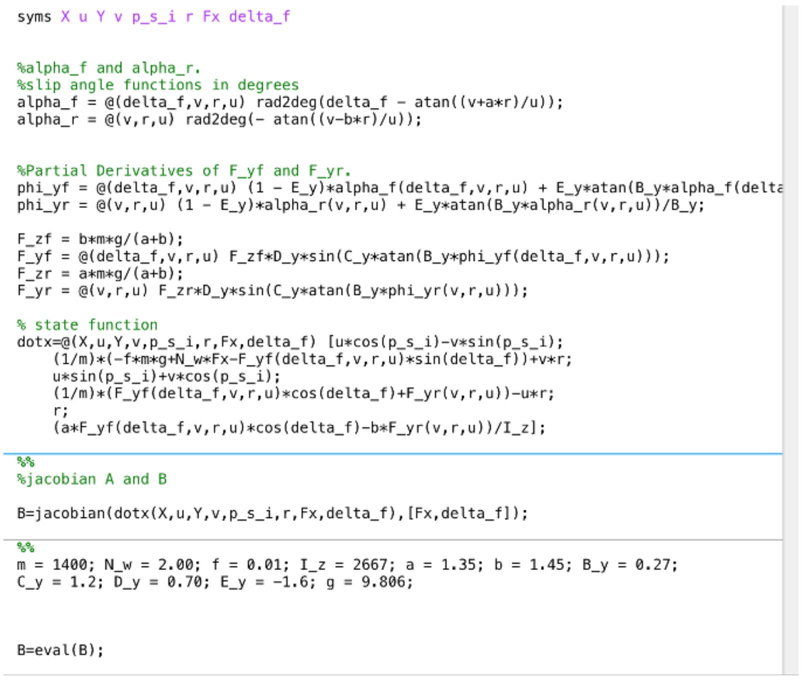
$$v(k+1) = v(k) + \frac{F_x}{m} * dt \tag{4}$$

$$F_x = \frac{m(v_{desired}(k+1) - v(k))}{dt} \tag{5}$$



```
syms X u Y v p_s_i r Fx delta_f



%alpha_f and alpha_r.
%slip angle functions in degrees
alpha_f = @(delta_f,v,r,u) rad2deg(delta_f - atan((v+a*r)/u));
alpha_r = @(v,r,u) rad2deg(- atan((v-b*r)/u));


%Partial Derivatives of F_yf and F_yr.
phi_yf = @(delta_f,v,r,u) (1 - E_y)*alpha_f(delta_f,v,r,u) + E_y*atan(B_y*alpha_f(delta
phi_yr = @(v,r,u) (1 - E_y)*alpha_r(v,r,u) + E_y*atan(B_y*alpha_r(v,r,u))/B_y;

F_zf = b*m*g/(a+b);
F_yf = @(delta_f,v,r,u) F_zf*D_y*sin(C_y*atan(B_y*phi_yf(delta_f,v,r,u)));
F_zr = a*m*g/(a+b);
F_yr = @(v,r,u) F_zr*D_y*sin(C_y*atan(B_y*phi_yr(v,r,u)));

% state function
dotx=@(X,u,Y,v,p_s_i,r,Fx,delta_f) [u*cos(p_s_i)-v*sin(p_s_i);
    (1/m)*(-f*m*g+N_w*Fx-F_yf(delta_f,v,r,u)*sin(delta_f))+v*r;
    u*sin(p_s_i)+v*cos(p_s_i);
    (1/m)*(F_yf(delta_f,v,r,u)*cos(delta_f)+F_yr(v,r,u))-u*r;
    r;
    (a*F_yf(delta_f,v,r,u)*cos(delta_f)-b*F_yr(v,r,u))/I_z];

%%
%jacobian A and B
A=jacobian(dotx(X,u,Y,v,p_s_i,r,Fx,delta_f),[X,u,Y,v,p_s_i,r]);
% B=jacobian(dotx(X,u,Y,v,p_s_i,r,Fx,delta_f),[Fx,delta_f]);

%%
m = 1400; N_w = 2.00; f = 0.01; I_z = 2667; a = 1.35; b = 1.45; B_y = 0.27;
C_y = 1.2; D_y = 0.70; E_y = -1.6; g = 9.806;

A=eval(A);
```

Figure 1: Jacobian A Using MATLAB



```
syms X u Y v p_s_i r Fx delta_f


%alpha_f and alpha_r.
%slip angle functions in degrees
alpha_f = @(delta_f,v,r,u) rad2deg(delta_f - atan((v+a*r)/u));
alpha_r = @(v,r,u) rad2deg(- atan((v-b*r)/u));


%Partial Derivatives of F_yf and F_yr.
phi_yf = @(delta_f,v,r,u) (1 - E_y)*alpha_f(delta_f,v,r,u) + E_y*atan(B_y*alpha_f(delta
phi_yr = @(v,r,u) (1 - E_y)*alpha_r(v,r,u) + E_y*atan(B_y*alpha_r(v,r,u))/B_y;

F_zf = b*m*g/(a+b);
F_yf = @(delta_f,v,r,u) F_zf*D_y*sin(C_y*atan(B_y*phi_yf(delta_f,v,r,u)));
F_zr = a*m*g/(a+b);
F_yr = @(v,r,u) F_zr*D_y*sin(C_y*atan(B_y*phi_yr(v,r,u)));

% state function
dotx=@(X,u,Y,v,p_s_i,r,Fx,delta_f) [u*cos(p_s_i)-v*sin(p_s_i);
    (1/m)*(-f*m*g+N_w*Fx-F_yf(delta_f,v,r,u)*sin(delta_f))+v*r;
    u*sin(p_s_i)+v*cos(p_s_i);
    (1/m)*(F_yf(delta_f,v,r,u)*cos(delta_f)+F_yr(v,r,u))-u*r;
    r;
    (a*F_yf(delta_f,v,r,u)*cos(delta_f)-b*F_yr(v,r,u))/I_z];

%%
%jacobian A and B

B=jacobian(dotx(X,u,Y,v,p_s_i,r,Fx,delta_f),[Fx,delta_f]);

%%
m = 1400; N_w = 2.00; f = 0.01; I_z = 2667; a = 1.35; b = 1.45; B_y = 0.27;
C_y = 1.2; D_y = 0.70; E_y = -1.6; g = 9.806;



B=eval(B);
```

Figure 2: Jacobian B Using MATLAB

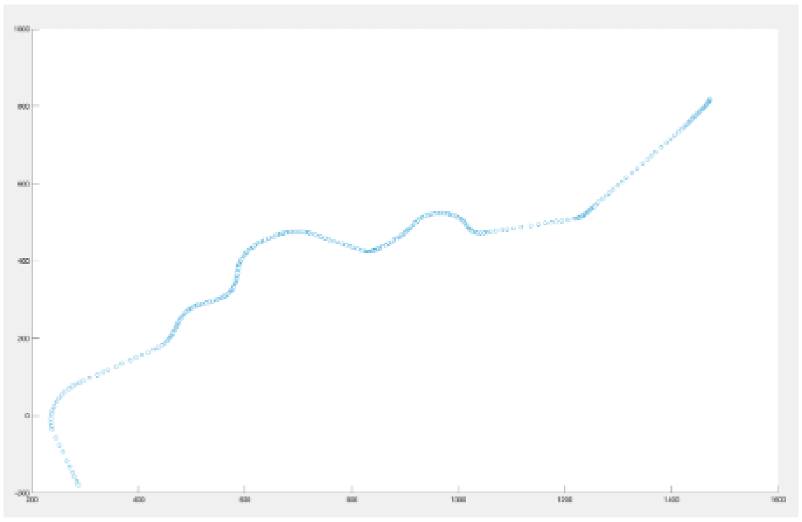
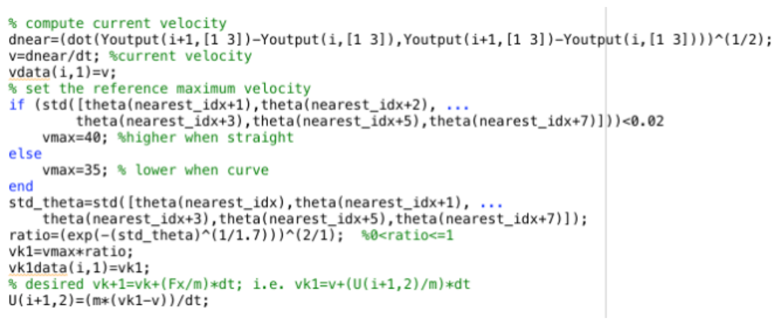
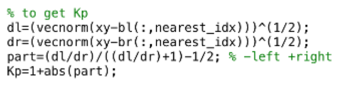


Figure 3: Cline of the Track Using Scatter



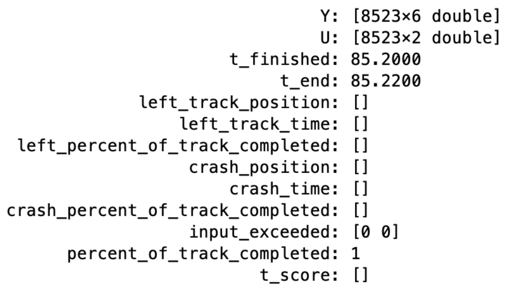
```
% compute current velocity
dnear=(dot(Youtput(i+1,[1 3])-Youtput(i,[1 3]),Youtput(i+1,[1 3])-Youtput(i,[1 3])))^(1/2);
v=dnear/dt; %current velocity
vdata(i,1)=v;
% set the reference maximum velocity
if (std([theta(nearest_idx+1),theta(nearest_idx+2), ...
        theta(nearest_idx+3),theta(nearest_idx+5),theta(nearest_idx+7)]))<0.02
    vmax=40; %higher when straight
else
    vmax=35; % lower when curve
end
std_theta=std([theta(nearest_idx),theta(nearest_idx+1), ...
    theta(nearest_idx+3),theta(nearest_idx+5),theta(nearest_idx+7)]);
ratio=(exp(-(std_theta)^(1/1.7)))^(2/1);  %0<ratio<=1
vk1=vmax*ratio;
vk1data(i,1)=vk1;
% desired vk+1=vk+(Fx/m)*dt; i.e. vk1=v+(U(i+1,2)/m)*dt
U(i+1,2)=(m*(vk1-v))/dt;
```

Figure 4: Get Fx



```
% to get Kp
dl=(vecnorm(xy-bl(:,nearest_idx)))^(1/2);
dr=(vecnorm(xy-br(:,nearest_idx)))^(1/2);
part=(dl/dr)/((dl/dr)+1)-1/2; % -left +right
Kp=1+abs(part);
```

Figure 5: Get Kp



```
                                Y: [8523×6 double]
                                U: [8523×2 double]
                       t_finished: 85.2000
                            t_end: 85.2200
              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 6: Result

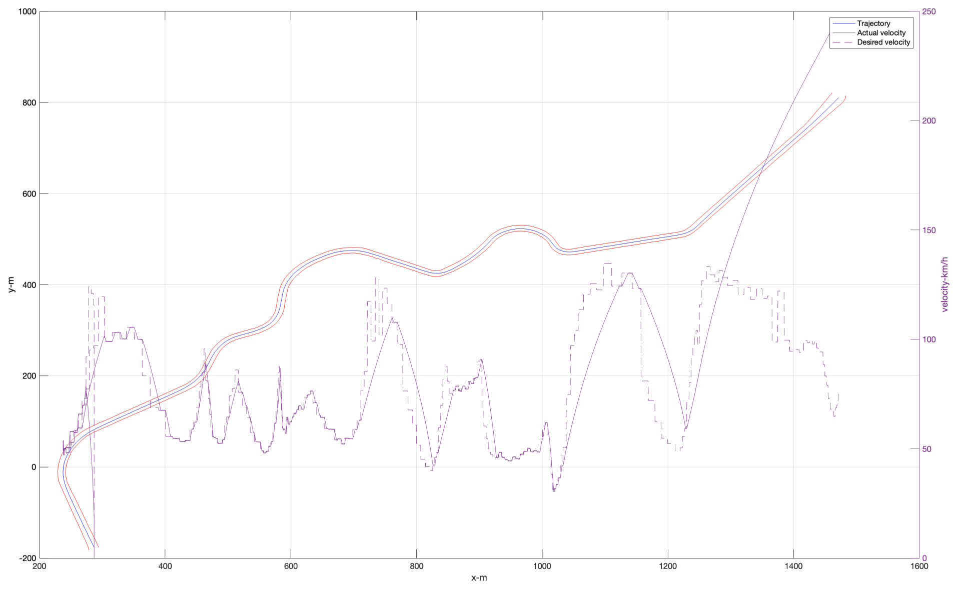


Figure 7: Actual Trajectory together with Actual Velocity and Desired Velocity

2.2 Task2

My initial idea for Task2 is to generate obstacle avoidance reference through trajectory synthesis, or to use MPC algorithm. The two algorithms did not work very well in the experiment. After my first success run of Task1 using PID control, Jiachen Jiang began to use PID algorithm to complete Task 2. After I improved Task1, I tried to use my algorithm in Task2, but soon Jiachen's algorithm could complete Task2, so I continued to focus on improving the speed of Task1.

3 Teammates' Work

- **Jiachen Jiang** Control Task, mainly focused on task2, successfully completed task2.
- **Chengtian Zhang** Control Task, Explored MPC control for task1 and calculated the Jacobin matrix by hand. Explored to generate a reference lane for task2.
- **Shuoqi Wang** Control Task, Calculated Jacobin matrix using jacobian() function in MATLAB. Tested MPC control for task2.
- **Lingfei Huo** Perception Task.