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).

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

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, $0 < ratio \leq 1$.

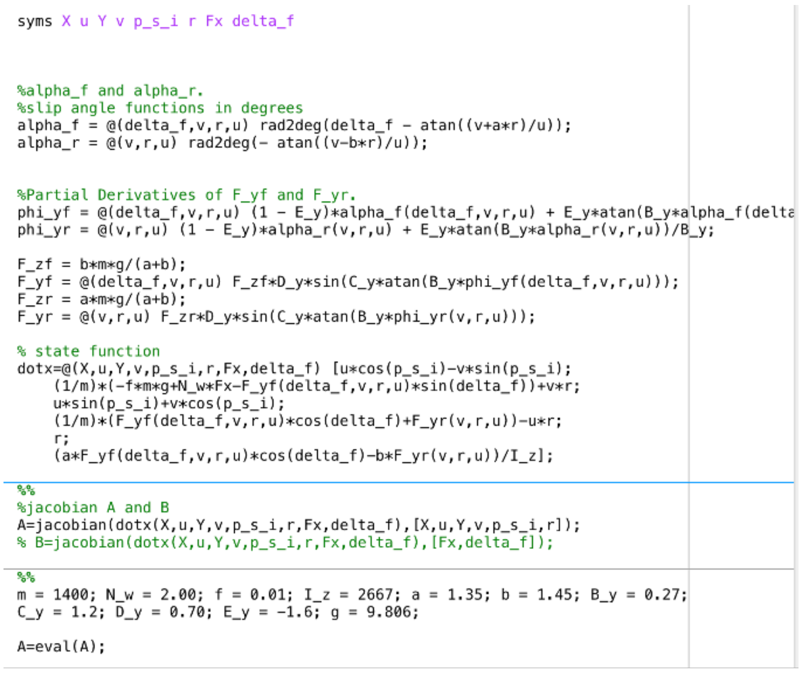
Set the desired the velocity,

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

According to the relationship between force and velocity,

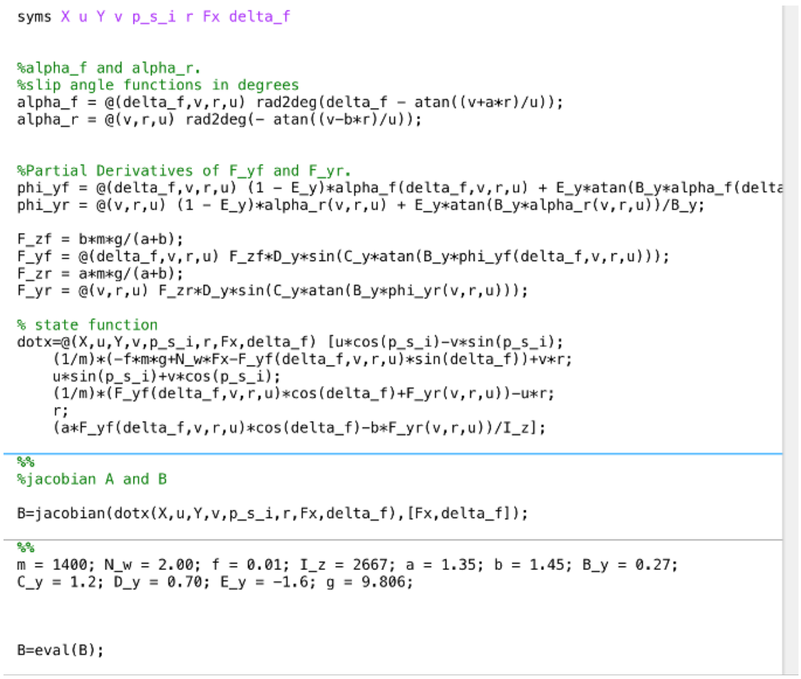
$$v(k+1) = v(k) + \frac{F_x}{m} * dt \tag{4}$$

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



```
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

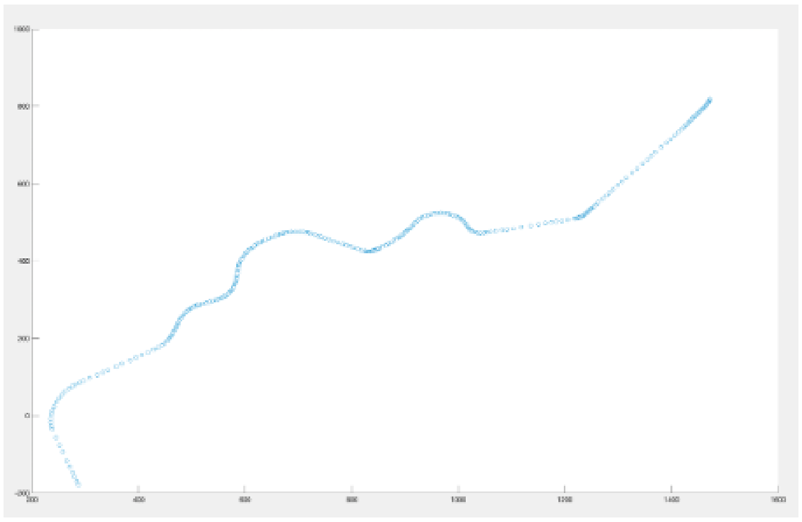
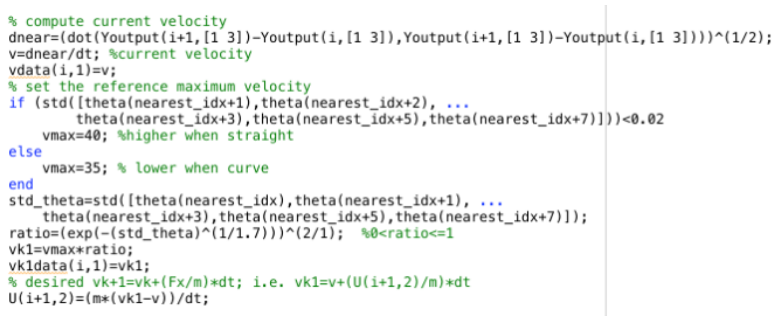
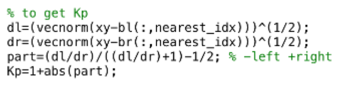


Figure 3: Cline of the Track Using Scatter



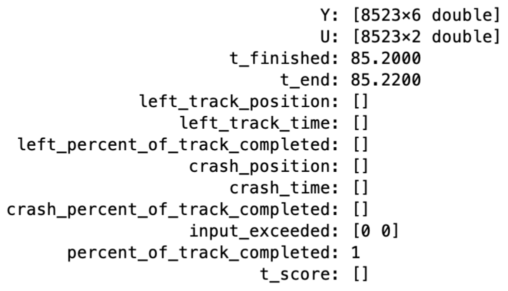
```
% compute current velocity
dnear=(dot(Youtput(i+1,[1 3])-Youtput(i,[1 3]),Youtput(i+1,[1 3])-Youtput(i,[1 3])))^(1/2);
v=dnear/dt; %current velocity
vdata(i,1)=v;
% set the reference maximum velocity
if (std([theta(nearest_idx+1),theta(nearest_idx+2), ...
        theta(nearest_idx+3),theta(nearest_idx+5),theta(nearest_idx+7)]))<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

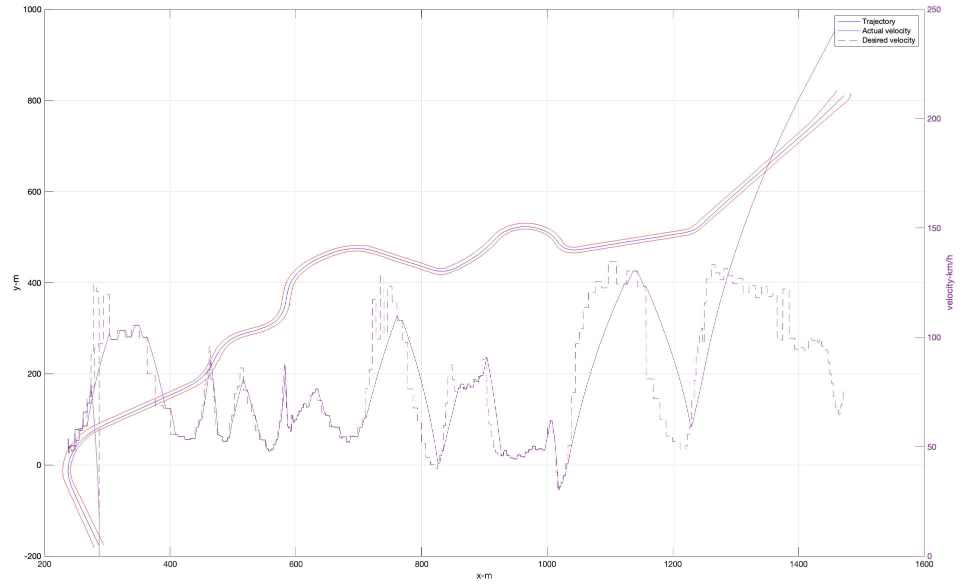


Figure 7: Actual Trajectory together with Actual Velocity and Desired Velocity

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.

ROB 535 Control Project Team26 Documentation

Jiachen Jiang
jiachenj@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 mainly focus on task2. There are three parts of my work, including Avoid obstacles module, generate reference speed and angle module and PI controller module.

2.1 Avoid Obstacles

The function of this module is getting target path according to obstacles and test track. Specifically, the target path is a series of positions and angles, e.g. (x, y, θ), which can avoid all obstacles seen within 150m. I choose the center lane as the target path initially. If I detect an obstacle on my target path, I would calculate the minimum distance between the center of the obstacle and the left/right lane to determine whether the obstacle on the left or right. Then we would choose the left lane or right lane as our target path according to position of the obstacle.

To detect the obstacle, I draw a circle around the polygon of the obstacle with a little bit large radius. Then I calculate the distance between the centre of circle and our target line and compare it with the radius. If the distance is smaller than the radius of obstacle, then the obstacle is detected.

To avoid touching the boundaries of left/right lane, I set the target path only at 2/3 between center lane and left/right lane.

2.2 Generate Reference

The function of this module is generating reference velocity(u) and angle(ϕ) according to the first module's target path. Since the points in original path is sparse, I interpolated it for 20 times. Then for each simulation time step, e.g. 0.01, I get the reference velocity(u) and angle(ϕ).

For velocity(u) in time t,

$$u(t) = \frac{\sqrt{(x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2}}{t_{perpoint}} \tag{1}$$

where $x_1(t)$ and $y_1(t)$ are positions of start point in target path in time t, $x_2(t)$ and $y_2(t)$ are positions of end point in target path in time t. And $t_{perpoint}$ is constant time used from start point to end point.

For angle(ϕ) in time t, I use the angle(θ) in start point in target path in time t directly.

2.3 PID Controller

The PID controller would output control $[\delta, F_x]$ to meet reference in the second module. It only needs to simulate 50 points to control the car for 0.5s. I use Proportional controller for F_x according to reference velocity and Proportional+Integral controller for δ according to reference angle.

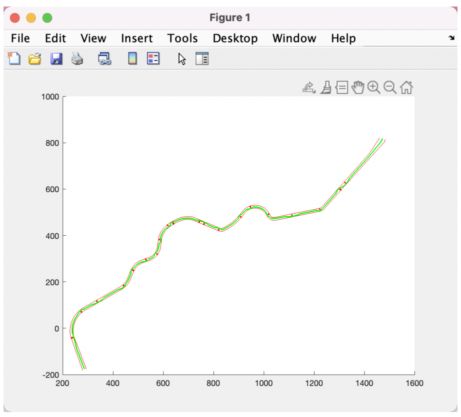
To generate appropriate $F_x(t)$,

$$F_x(t) = Kp_u * (u_{ref}(t) - u(t)) \tag{2}$$

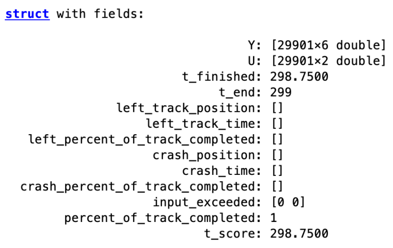
where Kp_u is velocity gain of proportional part and $u_{ref}(t)$ is reference velocity and $u(t)$ is our current velocity.
To generate appropriate $\delta(t)$,

$$\begin{aligned} \delta(t) &= Kp_\phi * (\phi_{ref}(t) - \phi(t)) + Ki_\phi * I_{part}(t) \\ I_{part}(t) &= \sum_{i=1}^{t*0.01} 0.01 * Sign(l/r) * d_{ref}[i] \end{aligned} \tag{3}$$

Where Kp_ϕ and Ki_ϕ are velocity gain of proportional part and integral part. $\phi_{ref}(t)$ is reference angle and $\phi(t)$ is our current angle. $I_{part}(t)$ is the integral part. The $sign(l/r)$ indicates that whether the car is on the left or right of reference path. The $d_{ref}[i]$ is the distance from current point to reference point.



(a) Trajectory of avoiding obstacles in real time.



```
struct with fields:

                           Y: [29901×6 double]
                           U: [29901×2 double]
                  t_finished: 298.7500
                       t_end: 299
         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: 298.7500
```

(b) Information of avoiding obstacles in real time.

Figure 1: Task 2 Result

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.
- **Shuoqi Wang** Control Task: Calculate Jacobin matrix using matlab jacobian() function. Test MPC method for task2.
- **Lingfei Huo** Perception Task.

ROB 535 Control Project Team26 Documentation

Chengtian Zhang
zctchn@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

MPC adopts a compromise strategy, not only considering the current state, but considering the future finite time domain, sacrificing optimality to a certain extent. I mainly focused on MPC method, which was not used because my system was unable to tackle with task 2 perfectly.

2.1 Task 1

In task 1, the MPC system will periodically accept the state quantity state, inputs the MPC model, and calculates the optimal control solution based on model, cost and constraints.
General working steps of MPC can be summarized as follows:

- Build prediction model based on state information, design loss function and constraint function based on model constraints and output requirements;
- Combined with previous information, current state and prediction model, calculate the optimal control solution and predict the n-step system output;
- Output execution;
- Enter next iteration, obtain the detection information, and repeat steps 1, 2 and 3.

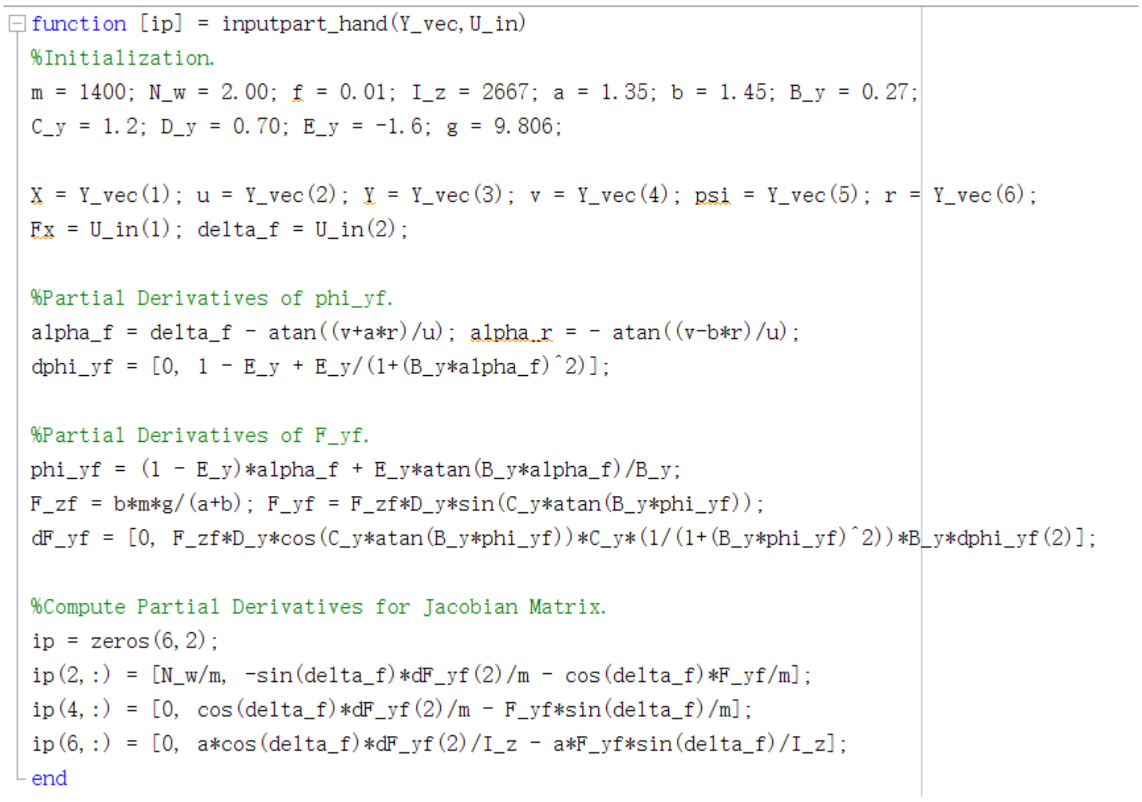
Figure 1, 2, 3 shows the calculation and implement for MPC input and state (A_i and B_i) parts.
Figure 4 shows MPC worked well on task 1.

2.2 Task 2

In task 2, I proposed to first run on the left lane and detect obstacle, if the current running lane exists an obstacle, I will switch to another lane. For the left and right reference, I used Zimin Lu's results. Before I could build up a correct system, my teammates investigating PID methods finished their prototype so we just devote together to test and verify the PI method.

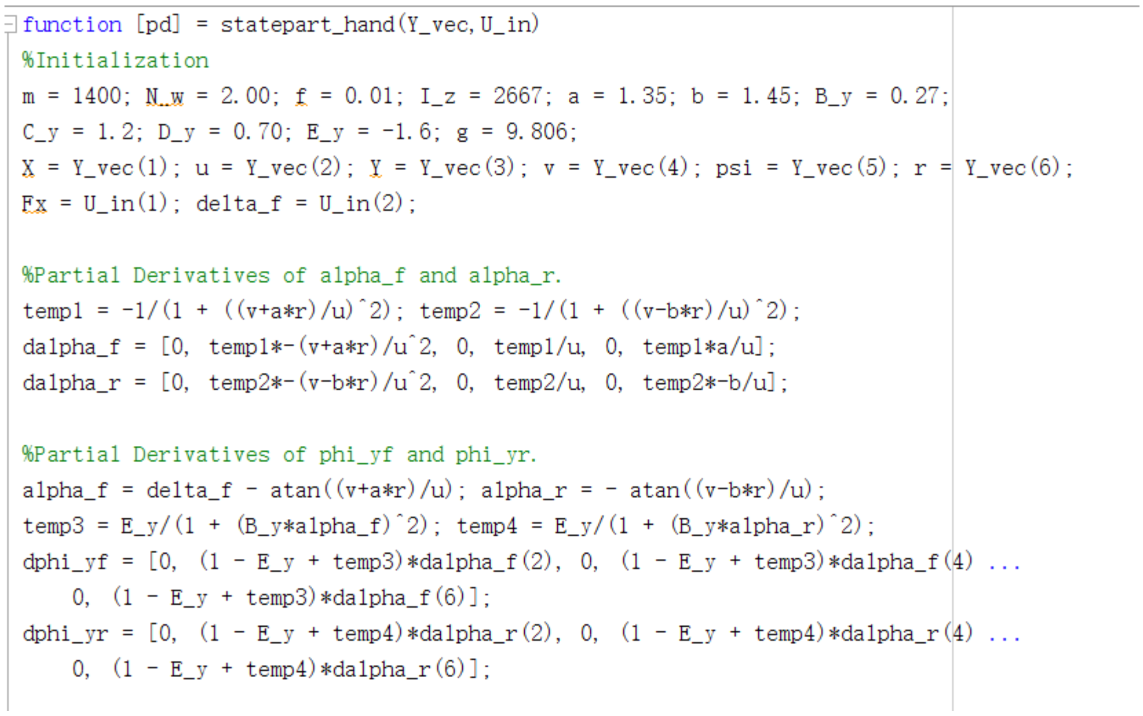
3 Teammates' Work

- **Jiachen Jiang** Control project part2
- **Zimin Lu** Control project part1
- **Shuoqi Wang** MPC method
- **Lingfei Huo** Preception task



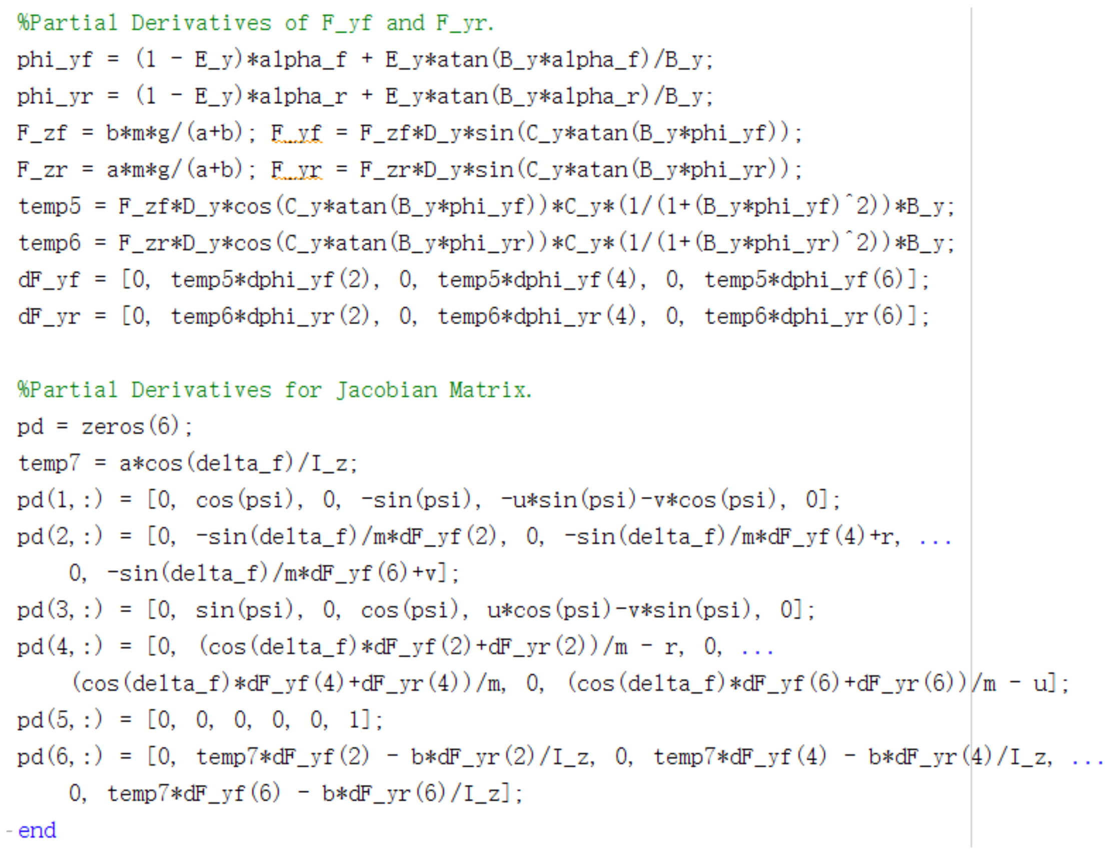
```
function [ip] = inputpart_hand(Y_vec,U_in)
%Initialization.
m = 1400; N_w = 2.00; f = 0.01; I_z = 2667; a = 1.35; b = 1.45; B_y = 0.27;
C_y = 1.2; D_y = 0.70; E_y = -1.6; g = 9.806;

X = Y_vec(1); u = Y_vec(2); Y = Y_vec(3); v = Y_vec(4); psi = Y_vec(5); r = Y_vec(6);
Fx = U_in(1); delta_f = U_in(2);

%Partial Derivatives of phi_yf.
alpha_f = delta_f - atan((v+a*r)/u); alpha_r = - atan((v-b*r)/u);
dphi_yf = [0, 1 - E_y + E_y/(1+(B_y*alpha_f)^2)];

%Partial Derivatives of F_yf.
phi_yf = (1 - E_y)*alpha_f + E_y*atan(B_y*alpha_f)/B_y;
F_zf = b*m*g/(a+b); F_yf = F_zf*D_y*sin(C_y*atan(B_y*phi_yf));
dF_yf = [0, F_zf*D_y*cos(C_y*atan(B_y*phi_yf))*C_y*(1/(1+(B_y*phi_yf)^2))*B_y*dphi_yf(2)];

%Compute Partial Derivatives for Jacobian Matrix.
ip = zeros(6,2);
ip(2,:) = [N_w/m, -sin(delta_f)*dF_yf(2)/m - cos(delta_f)*F_yf/m];
ip(4,:) = [0, cos(delta_f)*dF_yf(2)/m - F_yf*sin(delta_f)/m];
ip(6,:) = [0, a*cos(delta_f)*dF_yf(2)/I_z - a*F_yf*sin(delta_f)/I_z];
end
```

Figure 1: input part by hand



```
function [pd] = statepart_hand(Y_vec,U_in)
%Initialization
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;
X = Y_vec(1); u = Y_vec(2); Y = Y_vec(3); v = Y_vec(4); psi = Y_vec(5); r = Y_vec(6);
Fx = U_in(1); delta_f = U_in(2);

%Partial Derivatives of alpha_f and alpha_r.
temp1 = -1/(1 + ((v+a*r)/u)^2); temp2 = -1/(1 + ((v-b*r)/u)^2);
dalpha_f = [0, temp1*-(v+a*r)/u^2, 0, temp1/u, 0, temp1*a/u];
dalpha_r = [0, temp2*-(v-b*r)/u^2, 0, temp2/u, 0, temp2*-b/u];

%Partial Derivatives of phi_yf and phi_yr.
alpha_f = delta_f - atan((v+a*r)/u); alpha_r = - atan((v-b*r)/u);
temp3 = E_y/(1 + (B_y*alpha_f)^2); temp4 = E_y/(1 + (B_y*alpha_r)^2);
dphi_yf = [0, (1 - E_y + temp3)*dalpha_f(2), 0, (1 - E_y + temp3)*dalpha_f(4) ...
    0, (1 - E_y + temp3)*dalpha_f(6)];
dphi_yr = [0, (1 - E_y + temp4)*dalpha_r(2), 0, (1 - E_y + temp4)*dalpha_r(4) ...
    0, (1 - E_y + temp4)*dalpha_r(6)];
```

Figure 2: state part by hand 1



```
%Partial Derivatives of F_yf and F_yr.
phi_yf = (1 - E_y)*alpha_f + E_y*atan(B_y*alpha_f)/B_y;
phi_yr = (1 - E_y)*alpha_r + E_y*atan(B_y*alpha_r)/B_y;
F_zf = b*m*g/(a+b); F_yf = F_zf*D_y*sin(C_y*atan(B_y*phi_yf));
F_zr = a*m*g/(a+b); F_yr = F_zr*D_y*sin(C_y*atan(B_y*phi_yr));
temp5 = F_zf*D_y*cos(C_y*atan(B_y*phi_yf))*C_y*(1/(1+(B_y*phi_yf)^2))*B_y;
temp6 = F_zr*D_y*cos(C_y*atan(B_y*phi_yr))*C_y*(1/(1+(B_y*phi_yr)^2))*B_y;
dF_yf = [0, temp5*dphi_yf(2), 0, temp5*dphi_yf(4), 0, temp5*dphi_yf(6)];
dF_yr = [0, temp6*dphi_yr(2), 0, temp6*dphi_yr(4), 0, temp6*dphi_yr(6)];

%Partial Derivatives for Jacobian Matrix.
pd = zeros(6);
temp7 = a*cos(delta_f)/I_z;
pd(1,:) = [0, cos(psi), 0, -sin(psi), -u*sin(psi)-v*cos(psi), 0];
pd(2,:) = [0, -sin(delta_f)/m*dF_yf(2), 0, -sin(delta_f)/m*dF_yf(4)+r, ...
    0, -sin(delta_f)/m*dF_yf(6)+v];
pd(3,:) = [0, sin(psi), 0, cos(psi), u*cos(psi)-v*sin(psi), 0];
pd(4,:) = [0, (cos(delta_f)*dF_yf(2)+dF_yr(2))/m - r, 0, ...
    (cos(delta_f)*dF_yf(4)+dF_yr(4))/m, 0, (cos(delta_f)*dF_yf(6)+dF_yr(6))/m - u];
pd(5,:) = [0, 0, 0, 0, 0, 1];
pd(6,:) = [0, temp7*dF_yf(2) - b*dF_yr(2)/I_z, 0, temp7*dF_yf(4) - b*dF_yr(4)/I_z, ...
    0, temp7*dF_yf(6) - b*dF_yr(6)/I_z];
end
```

Figure 3: state part by hand 2

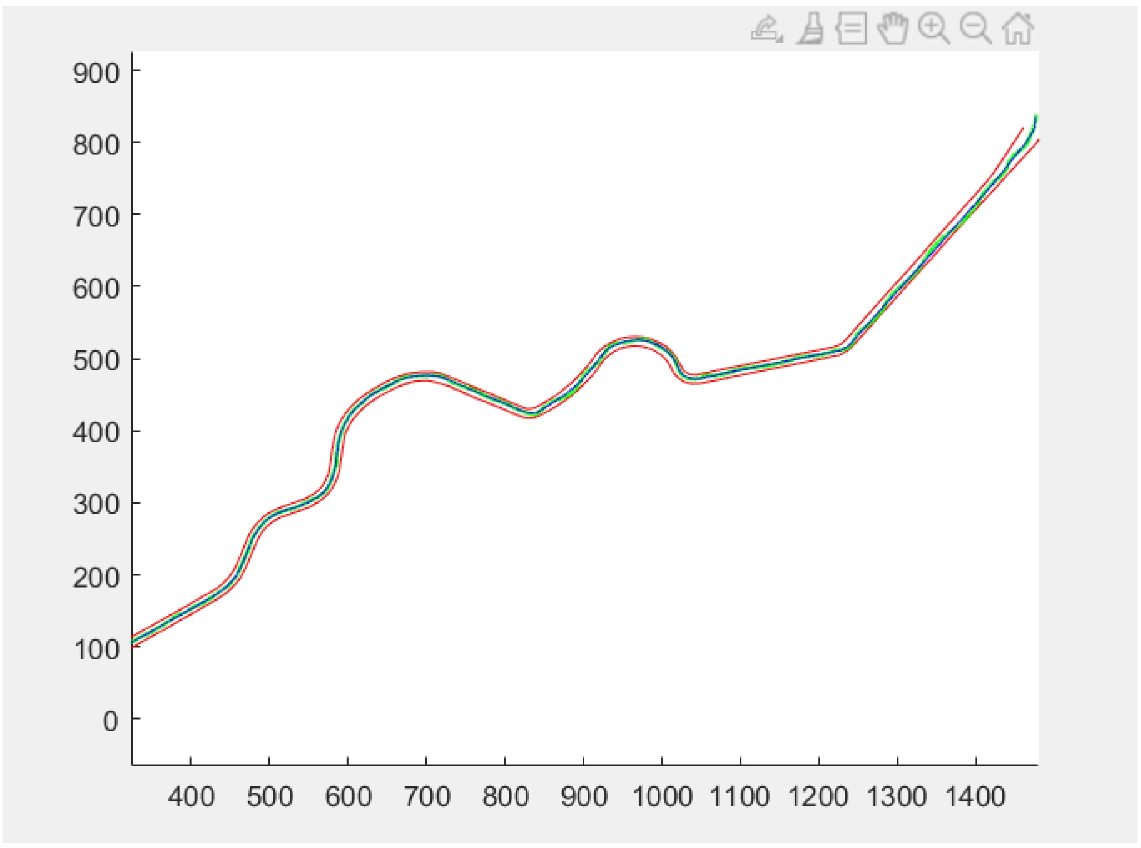


Figure 4: MPC implement on Task 1

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.

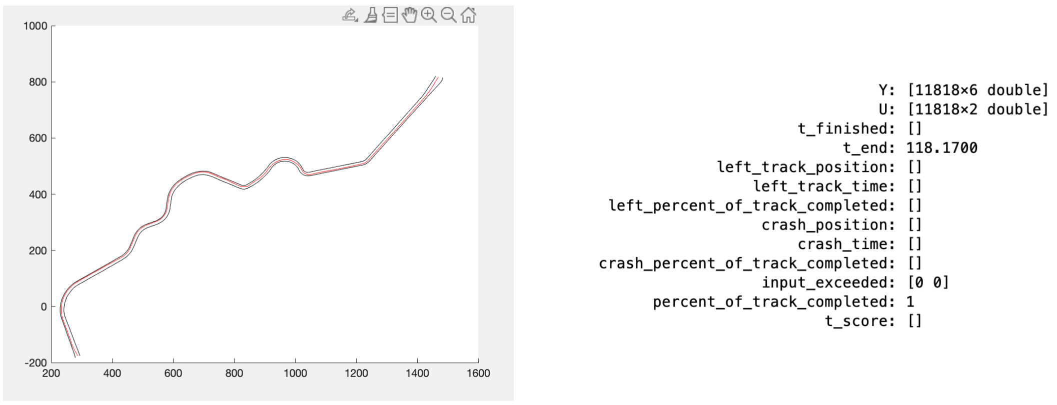
2.2 Task 2

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

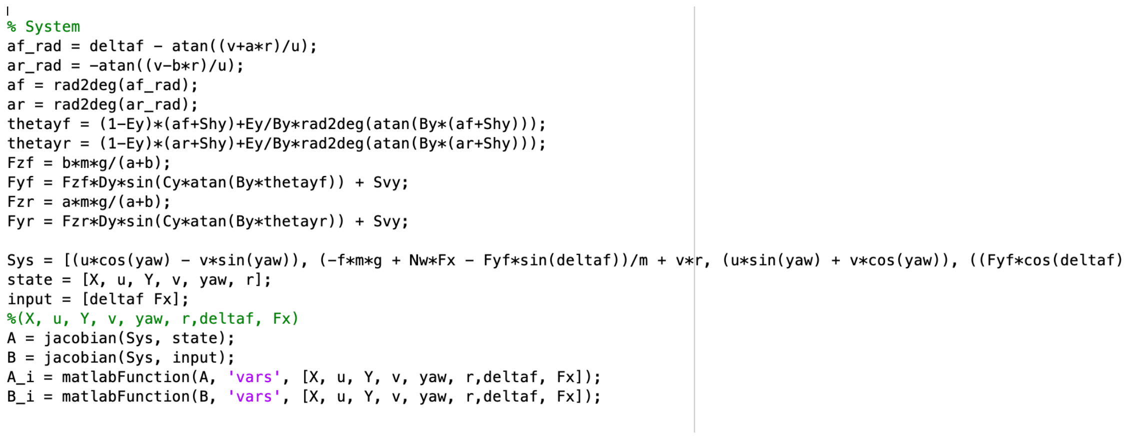
- **Zimin Lu** Control Task: mainly focus on task1. Generate the control input matrix using PID methods successfully.
- **Chengtian Zhang** Control Task: Explore MPC method for task1 and calculate the Jacobin matrix by hand. Generate the reference lane for task2.

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



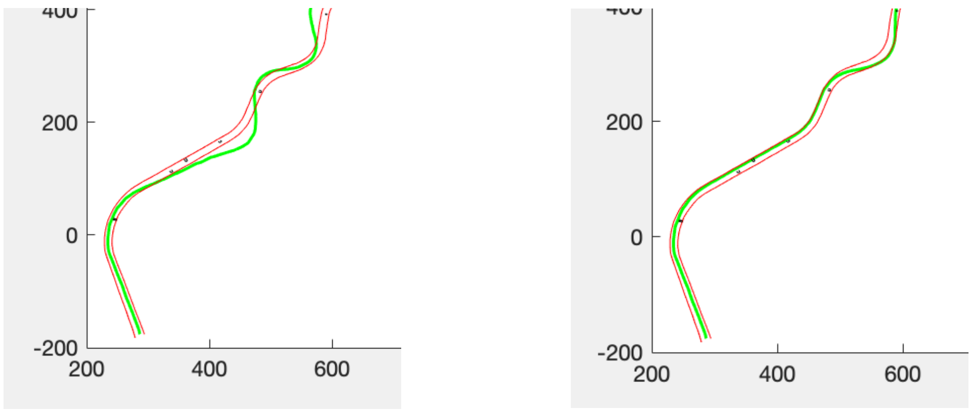
```
                              Y: [11818×6 double]
                              U: [11818×2 double]
                     t_finished: []
                          t_end: 118.1700
            left_track_position: []
                left_track_time: []
 left_percent_of_track_completed: []
                 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