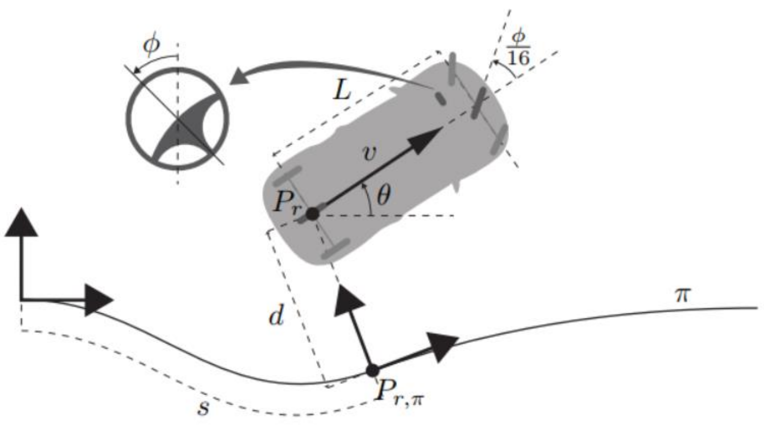
École Polytechnique Fédérale de Lausanne

Multivariable Control And Coordination Systems (EE-477)

Case Study: Path Tracking for an Autonomous Vehicle



Jianhao ZHENG (sciper: 323146)

4th January 2021

0. Introduction

The general objective of the case study is to design a closed-loop controller with observer to make the vehicle follow the reference trajectories. Trajectories are expressed in a parametric form. Parameter s specifies the displacement of the vehicle along the trajectory, parameter d specifies the lateral deviation between vehicle and path, parameter θ_e specifies the heading error, parameter L specifies the length of the car, parameter v specifies the speed, parameter ϕ specifies the position of the steering wheel, parameters v_{ref} and ϕ_{ref} specify the speed reference and steering wheel angle reference, parameter σ_v and σ_ϕ specifies the dynamic of the actuators and parameter $\kappa(s)$ specifies the curvature of the path at s.

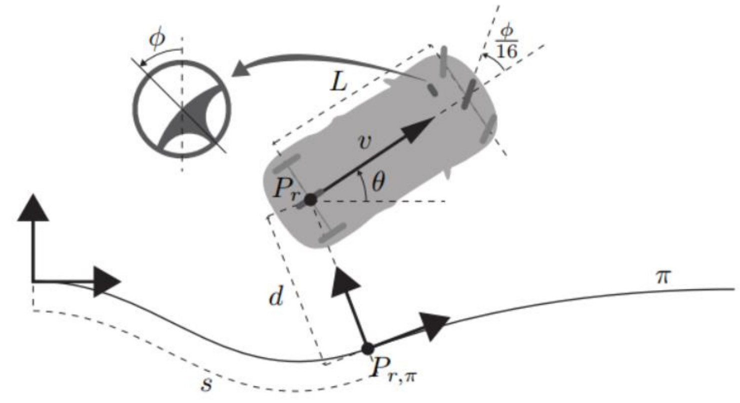


Figure 0.1. Coordinate system

0.1 The non-linear model in the standard form $\dot{x} = f(x, u)$

Firstly, I define the state and the input as following equation:

$$\mathbf{X} = [x_1, x_2, x_3, x_4, x_5] = [s, d, \theta_e, v, \phi] \tag{1}$$

$$\mathbf{U} = [u_1, u_2] = [v_{ref}, \phi_{ref}] \tag{2}$$

The non-linear state space model is the following formula,

$$\begin{aligned} \dot{x_1} &= \frac{x_4 \cos(x_3)}{1 - x_2 \kappa(x_1)} \\ \dot{x_2} &= x_4 \sin(x_3) \\ \dot{x_3} &= \frac{x_4}{L} \tan\left(\frac{x_5}{16}\right) - \kappa(x_1) \frac{x_4 \cos(x_3)}{1 - x_2 \kappa(x_1)} \\ \dot{x_4} &= \sigma_v (u_1 - x_4) \\ \dot{x_5} &= \sigma_\phi (u_2 - x_5) \end{aligned} \tag{3}$$

The output equation is,

$$\mathbf{y} = \mathbf{x} \tag{4}$$

0.2 Linearizing the system around a nominal point

Let's first try to find a nominal point:

$$\begin{aligned}
\dot{\bar{x}}_1 &= \frac{\bar{x}_4\cos(\bar{x}_3)}{1-\bar{x}_2\kappa(\bar{x}_1)} = 0\\
\dot{\bar{x}}_2 &= \bar{x}_4\sin(\bar{x}_3) = 0\\
\dot{\bar{x}}_3 &= \frac{\bar{x}_4}{L}\tan\left(\frac{\bar{x}_5}{16}\right) - \kappa(x_1)\frac{\bar{x}_4\cos(\bar{x}_3)}{1-\bar{x}_2\kappa(\bar{x}_1)} = 0\\
\dot{\bar{x}}_4 &= \sigma_v(\bar{u}_1 - \bar{x}_4) = 0\\
\dot{\bar{x}}_5 &= \sigma_\phi(\bar{u}_2 - \bar{x}_5) = 0
\end{aligned} \tag{5}$$

By the first two equation in equation (5), we can get $\bar{x}_4 = 0$. As long as $\bar{x}_4 = 0$, the value of $\dot{\bar{x}}_1, \dot{\bar{x}}_2$ and $\dot{\bar{x}}_3$ will always be 0. Thus, $\bar{x}_1$, $\bar{x}_2$ and $\bar{x}_3$ can take arbitrary value. The last two equations require $\bar{u}_1 = \bar{x}_4$ and $\bar{u}_2 = \bar{x}_5$.

The reason why we don't linearize the system around a nominal point is mainly because of $\bar{x}_4$ can only take value 0. Note that the linearized model can behave in the same way as the original non-linear state-space model only for small deviations around the nominal point. In this case, the linearized model will only work when x_4, which is the speed of the vehicle v, is around the value 0. However, this it not the usual case in the practice. That would be terrible if our vehicle can only work with speed near to 0.

0.3 Calculating a nominal trajectory

A nominal trajectory representing the constant-speed perfect tracking situation for a path of constant curvature $\boldsymbol{\kappa_{ref}}$ is calculated. The constant speed is noted as $\boldsymbol{v_{ref}}$.

As described above, we now have $\bar{x}_4 = v_{ref}$, $\kappa(x_1) = \kappa_{ref}$ and $\bar{x}_2 = 0$. By solving the equation derived in the first problem, we can get:

$$\bar{\mathbf{X}} = [\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5] = [v_{ref}t, 0, 0, v_{ref}, 16\arctan(L\kappa_{ref})] \tag{6}$$

$$\bar{\mathbf{U}} = [\bar{u}_1, \bar{u}_2] = [v_{ref}, 16\arctan(L\kappa_{ref})] \tag{7}$$

0.4 Linearizing the system around a nominal trajectory

The linearized model is here:

$$\dot{\tilde{\mathbf{x}}}(\boldsymbol{t}) = \begin{bmatrix} 0 & \kappa_{ref}v_{ref} & 0 & 1 & 0 \\ 0 & 0 & v_{ref} & 0 & 0 \\ 0 & -\kappa_{ref}{}^2 v_{ref} & 0 & 0 & \frac{v_{ref}}{16L}\left((L\kappa_{ref})^2+1\right) \\ 0 & 0 & 0 & -\sigma_v & 0 \\ 0 & 0 & 0 & 0 & -\sigma_\phi \end{bmatrix} \tilde{\boldsymbol{x}}(\boldsymbol{t}) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \sigma_v & 0 \\ 0 & \sigma_\phi \end{bmatrix} \tilde{\boldsymbol{u}}(\boldsymbol{t})$$

$$\tilde{\boldsymbol{y}}(\boldsymbol{t}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{x}}(\boldsymbol{t}) \tag{8}$$

Note that $A(3,5) = \frac{v_{ref}}{16L} sec^2\left(\arctan(L\kappa_{ref})\right) = \frac{v_{ref}}{16L}\left((L\kappa_{ref})^2 + 1\right)$

0.5 Discretizing the system using Euler approximation

Euler approximation:

$$\boldsymbol{x}(\mathbf{k}+\mathbf{1}) \approx \boldsymbol{x}(\boldsymbol{k}) + hf[\boldsymbol{x}(\boldsymbol{k}), \boldsymbol{u}(\boldsymbol{k}), kh] = (I + Ah)\boldsymbol{x}(\boldsymbol{k}) + Bh\boldsymbol{u}(\boldsymbol{k}) \tag{9}$$

Discretized system:

$$\tilde{\boldsymbol{x}}(\mathbf{k}+\mathbf{1}) = \Phi\tilde{\boldsymbol{x}}(\boldsymbol{k}) + \Gamma\tilde{\boldsymbol{u}}(\boldsymbol{k}) \tag{10}$$

where,

$$\Phi = I + Ah = \begin{bmatrix} 1 & \kappa_{ref} v_{ref} h & 0 & h & 0 \\ 0 & 1 & v_{ref} h & 0 & 0 \\ 0 & -\kappa_{ref}{}^2 v_{ref} h & 1 & 0 & \frac{v_{ref}}{16L}\left(\left(L\kappa_{ref}\right)^2 + 1\right)h \\ 0 & 0 & 0 & 1-\sigma_v h & 0 \\ 0 & 0 & 0 & 0 & 1-\sigma_\phi h \end{bmatrix} \tag{11}$$

$$\Gamma = Bh = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \sigma_v h & 0 \\ 0 & \sigma_\phi h \end{bmatrix} \tag{12}$$

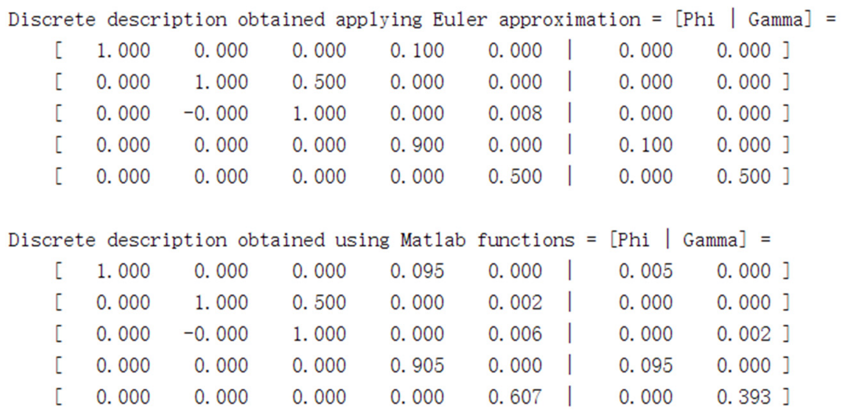
$$\tilde{\boldsymbol{y}}(\boldsymbol{k}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{x}}(\boldsymbol{k}) \tag{13}$$

1. Case study 1

In this case study, simulations on the system with continuous non-linear model, continuous linear model and discrete-time linear model has been done in MATLAB and SIMULINK. It's an open loop control. The input sequence and the initial states are all set manually. At the end, the differences between the behaviors of the three models will be discussed.

1.1 The arrays Φ and Γ obtained using the Euler approximation method, the Matlab command c2d and the Taylor-series approximation

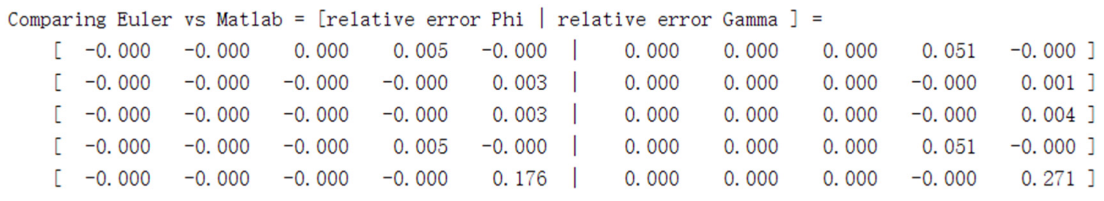
When using the Matlab command **c2d**, I discretized the continuous model by method 'Zero-order hold'. Following is the arrays **Φ** and **Γ** obtained using two different methods and their comparison:



```
Discrete description obtained applying Euler approximation = [Phi | Gamma] =
    [  1.000   0.000   0.000   0.100   0.000  |   0.000   0.000 ]
    [  0.000   1.000   0.500   0.000   0.000  |   0.000   0.000 ]
    [  0.000  -0.000   1.000   0.000   0.008  |   0.000   0.000 ]
    [  0.000   0.000   0.000   0.900   0.000  |   0.100   0.000 ]
    [  0.000   0.000   0.000   0.000   0.500  |   0.000   0.500 ]

Discrete description obtained using Matlab functions = [Phi | Gamma] =
    [  1.000   0.000   0.000   0.095   0.000  |   0.005   0.000 ]
    [  0.000   1.000   0.500   0.000   0.002  |   0.000   0.000 ]
    [  0.000  -0.000   1.000   0.000   0.006  |   0.000   0.002 ]
    [  0.000   0.000   0.000   0.905   0.000  |   0.095   0.000 ]
    [  0.000   0.000   0.000   0.000   0.607  |   0.000   0.393 ]
```

Figure 1.1. matrix **Φ** and **Γ** of discrete state space model by Euler and **c2d**



```
Comparing Euler vs Matlab = [relative error Phi | relative error Gamma ] =
    [  -0.000   -0.000    0.000    0.005   -0.000  |   0.000    0.000    0.000    0.051   -0.000 ]
    [  -0.000   -0.000   -0.000   -0.000    0.003  |   0.000    0.000    0.000   -0.000    0.001 ]
    [  -0.000   -0.000   -0.000   -0.000    0.003  |   0.000    0.000    0.000   -0.000    0.004 ]
    [  -0.000   -0.000   -0.000    0.005   -0.000  |   0.000    0.000    0.000    0.051   -0.000 ]
    [  -0.000   -0.000   -0.000   -0.000    0.176  |   0.000    0.000    0.000   -0.000    0.271 ]
```

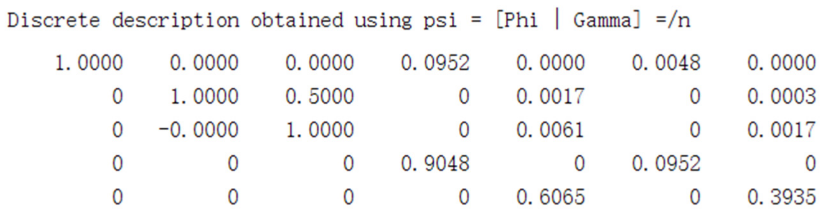
Figure 1.2. Comparison of matrix **Φ** and **Γ** obtained by Euler and **c2d**

According to figure 1.2, we can see that the matrix obtained by two different methods has a large error regarding to the state x_5, where $\dot{x_5} = \sigma_\phi(u_2 - x_5)$. The Euler approximation for x_5 is based on the formula $x_5(\mathrm{kh}+\mathrm{h}) \approx x_5(kh) + h\dot{x_5}(kh)$, while the real value should be $x_5(\mathrm{kh}+\mathrm{h}) = x_5(kh) + \int_0^h \dot{x_5}(kh+t)dt$. If $\dot{x_5}(kh)$ doesn't change very sharply with time, $h\dot{x_5}(kh)$ will be approximately equal to $\int_0^h \dot{x_5}(kh+t)dt$. However, the value of σ_ϕ is 5, which is large enough to cause significant change on $\dot{x_5}$. That leads to the error on the discrete model obtained by Euler approximation.

I also tried to implement the Taylor-series approximation of the exponential matrix. The series ψ is evaluated by the following way:

$$\psi = I + \frac{Ah}{2}\left(I + \frac{Ah}{3}\left(\quad \cdots \frac{Ah}{N-1}\left(I + \frac{Ah}{N}\right)\right)\right) \tag{14}$$

The parameter N is set to be 100. Figure 1.3 shows the matrix **Φ** and **Γ** obtained by Taylor-series approximation. The matrix is quite similar to that obtained by method **c2d** according to figure 1.1 and figure 1.3.



```
Discrete description obtained using psi = [Phi | Gamma] =/n

    1.0000    0.0000    0.0000    0.0952    0.0000    0.0048    0.0000
         0    1.0000    0.5000         0    0.0017         0    0.0003
         0   -0.0000    1.0000         0    0.0061         0    0.0017
         0         0         0    0.9048         0    0.0952         0
         0         0         0         0    0.6065         0    0.3935
```

Figure 1.3. matrix **Φ** and **Γ** of discrete state space model by Taylor-series approximation

1.2 Implementation of the Non-linear model – continuous time block

Figure 1.4 shows how I implement the non-linear continuous model.

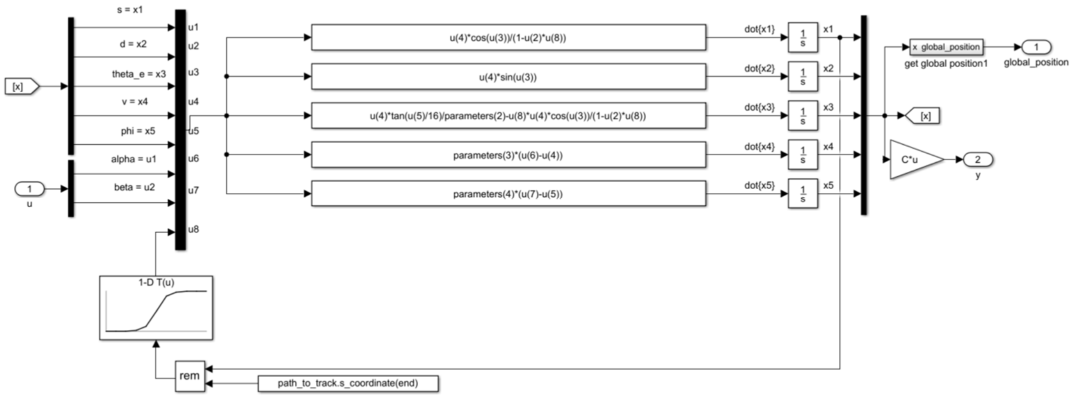


Figure 1.4. Non-linear model – continuous time block

1.3 The block diagram implemented in SIMULINK

Figure 1.5 shows the overall SIMULINK scheme for open loop control in **open_loop_experiments.slx**.

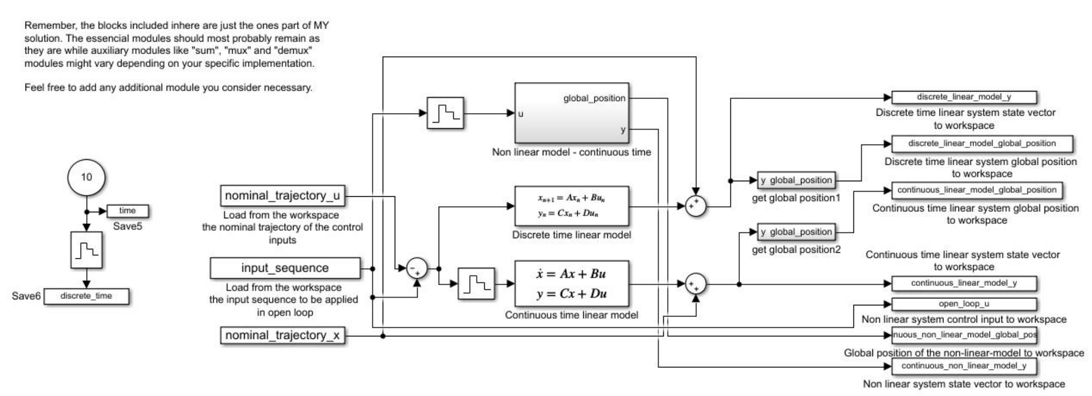

Remember, the blocks included inhere are just the ones part of MY solution. The essencial modules should most probably remain as they are while auxiliary modules like "sum", "mux" and "demux" modules might vary depending on your specific implementation.
Feel free to add any additional module you consider necessary.
10
time
Save5
Save6
discrete_time
nominal_trajectory_u
Load from the workspace the nominal trajectory of the control inputs
input_sequence
Load from the workspace the input sequence to be applied in open loop
nominal_trajectory_x
global_position
u
y
Non linear model - continuous time
$x_{n+1} = Ax_n + Bu_n$
$y_n = Cx_n + Du_n$
Discrete time linear model
$\dot{x} = Ax + Bu$
$y = Cx + Du$
Continuous time linear model
y global_position
get global position1
get global position2
discrete_linear_model_y
Discrete time linear system state vector to workspace
discrete_linear_model_global_position
Discrete time linear system global position to workspace
continuous_linear_model_global_position
Continuous time linear system global position to workspace
Continuous time linear system state vector to workspace
continuous_linear_model_y
open_loop_u
Non linear system control input to workspace
nuous_non_linear_model_global_pos
Global position of the non-linear-model to workspace
continuous_non_linear_model_y
Non linear system state vector to workspace

Figure 1.5. block diagram implemented in **open_loop_experiments.slx**

1.4 The simulation results

Three representative simulation results are presented in this section.

The initial state of experiment 1 and experiment 2 are both

$$\mathbf{X}(0) = \overline{\mathbf{X}}(0) = \left[0,0,0, v_{ref}, 16\arctan\left(L\kappa_{ref}\right)\right] \tag{15}$$

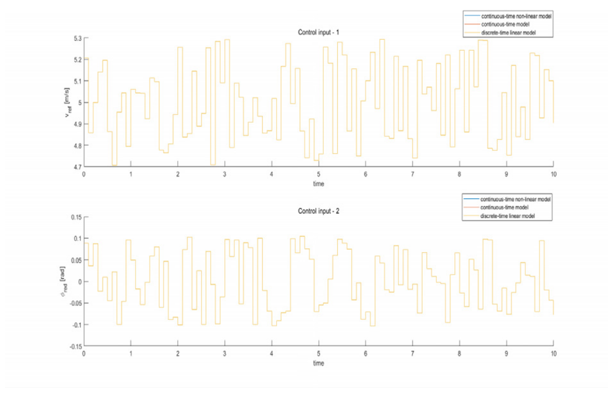
The initial state of experiment 3 is

$$\mathbf{X}(0) = \overline{\mathbf{X}}(0) = \left[0,0,\frac{\pi}{4}, v_{ref}, 16\arctan\left(L\kappa_{ref}\right)\right] \tag{16}$$

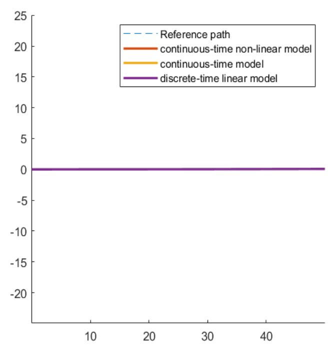
The sequence of inputs applied in **experiment 1**, $\mathbf{U} = [\boldsymbol{u_1}, \boldsymbol{u_2}]$, is a sequence of random value, where $u_1(k)$ takes random value between $[\bar{u}_1 - 0.3 \ , \ \bar{u}_1 + 0.3]$ and $u_2(k)$ takes random value between $[\bar{u}_2 - \frac{\pi}{30} \ , \ \bar{u}_2 + \frac{\pi}{30}]$. The inputs, states and trajectory of the vehicle are showed in figure 1.6.

The inputs applied in **experiment 2** is also random values. $u_1(k)$ takes random value between $[\bar{u}_1 - 5 \ , \ \bar{u}_1 + 5]$ and $u_2(k)$ takes random value between $[\bar{u}_2 - \frac{\pi}{3} \ , \ \bar{u}_2 + \frac{\pi}{3}]$. The inputs, states and trajectory of the vehicle are showed in figure 1.7.

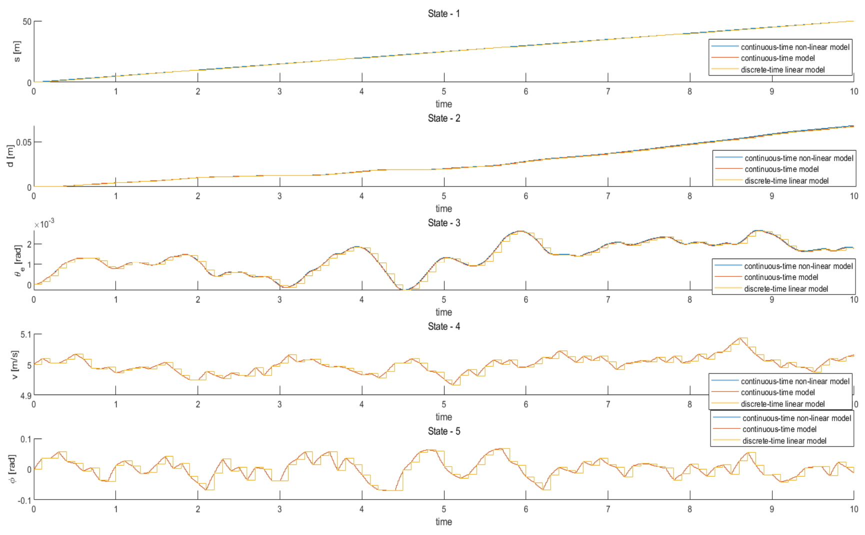
The inputs applied in **experiment 3** is that $u_1(k)$ takes constant value $\bar{u}_1 - 2 = 3$ and $u_2(k)$ takes random value between $[\bar{u}_2 - \frac{\pi}{30} \ , \ \bar{u}_2 + \frac{\pi}{30}]$. The inputs, states and trajectory of the vehicle are showed in figure 1.8.



(a) Control inputs

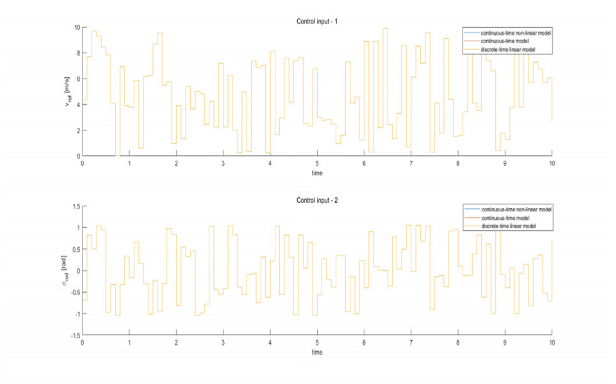


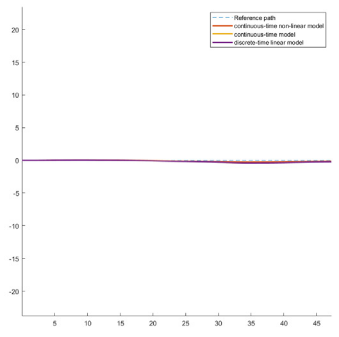
(b) Trajectory



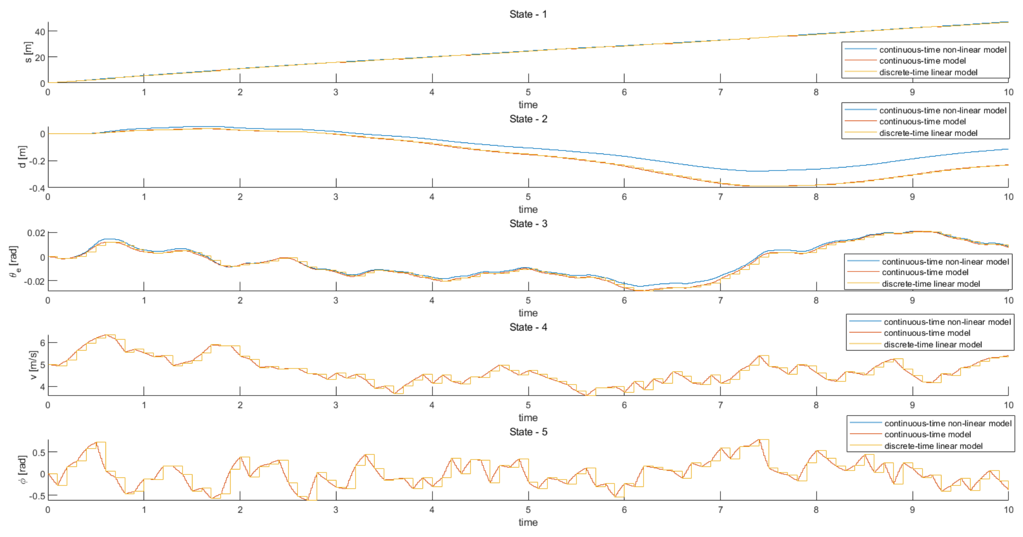
(c) States

Figure 1.6. Result of trajectory in experiment 1



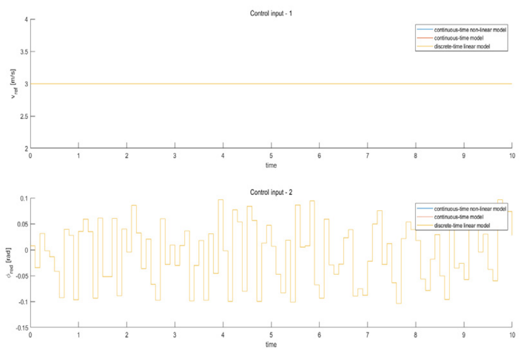


(a) Control inputs **(b)** Trajectory

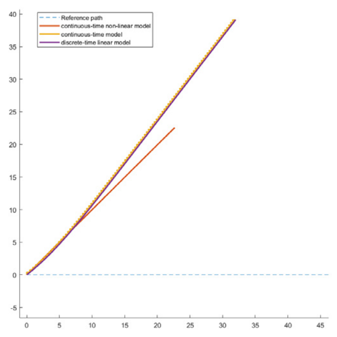


(c) States

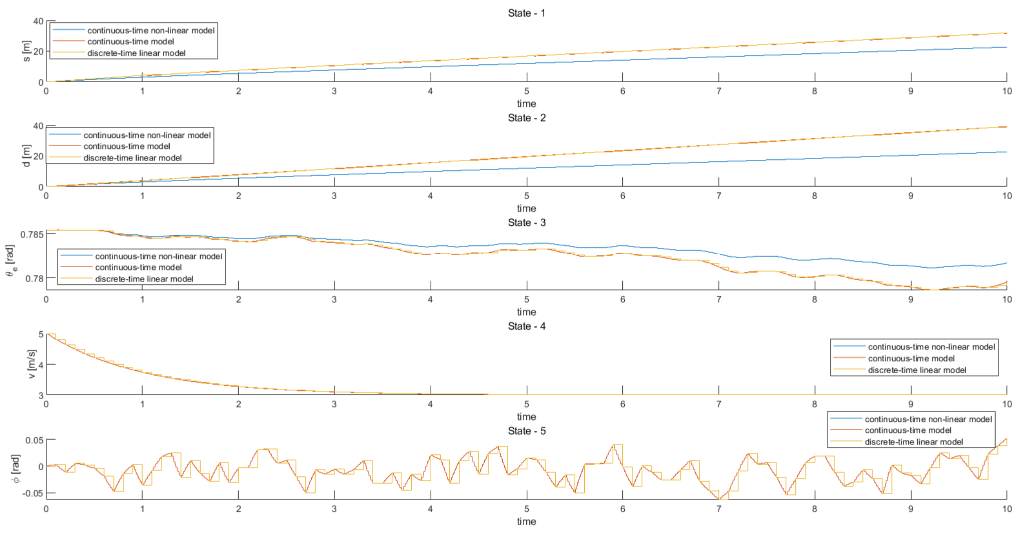
Figure 1.7. Result of states in experiment 2



(a) Control inputs



(b) Trajectory



(c) States

Figure 1.8. Result of states in experiment 3

1.5 Comparison of the linear model and the non-linear model

Whether the behavior obtained by linear model resemble that by non-linear model depends on how far the states and inputs are from the nominal trajectory. The further we are from the reference trajectory, the wronger the outcomes of the linear model.

When the real states and inputs are closed to the nominal trajectory with small deviation, the linear model resemble sufficiently well the behavior of the non-linear model. In experiment 1, the initial state is exactly the state of nominal trajectory and the input takes value approximate to that of nominal trajectory with small deviation. In this case, the behavior of the linear model and the non-linear one resembles well and the curves of the three models overlap, as we can see in figure 1.6.

However, if the real states or inputs go too far from the nominal trajectory, the linear model will have different behavior from the non-linear one. According to figure 1.7 and figure 1.8, states x_2 and x_3 has a significant error between two models since a wider scale of deviation is implemented in these two experiments. The states x_4 and x_5 of two models still remain the same is because the original functions of these states are already linear. Our linearized model doesn't change anything on these states.

An interesting thing is that state x_1 resemble in both models in experiment 2 while the difference occurs in experiment 3. That's because of the value of x_3. Let $g_1(\mathbf{x}) = \dot{x_1} = \frac{x_4\cos(x_3)}{1-x_2\kappa(x_1)}$, where $\kappa(x_1) = 10^{-10} \approx 0$. Then, $\frac{\partial g_1}{\partial x} = \begin{bmatrix} 0 & \frac{kx_4\cos(x_3)}{(1-kx_2)^2} & -\frac{x_4\sin(x_3)}{1-kx_2} & \frac{\cos(x_3)}{1-kx_2} & 0 \end{bmatrix}$. In experiment 2, x_3 takes value near 0, which leads to $\frac{\partial g_1}{\partial x} \approx [0 \quad 0 \quad 0 \quad 1 \quad 0]$. That means x_1 only depends on x_4 in linearized model. As mentioned before, x_4 remains the same in both models. So, x_1 keeps the same. However, in experiment 3, x_3 takes value near $\frac{\pi}{4}$. The state x_1 also depends on x_2 and x_3, which leads to the deviation.

1.6 Comparison of the continuous-time model and the discrete-time model

The results obtained by discrete-time linear model resemble those obtained by continuous-time model very well as we can see from figure 1.6 (c), 1.7 (c) and 1.8 (c). That's because in the matlab code, we implemented $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ obtained by **c2d,** which is precise enough. However, I tried to use the matrix obtained by Euler approximation, the results on state 5 has nonnegligible error, as showed in figure 1.9. The reason of the error has been discussed in question 1.

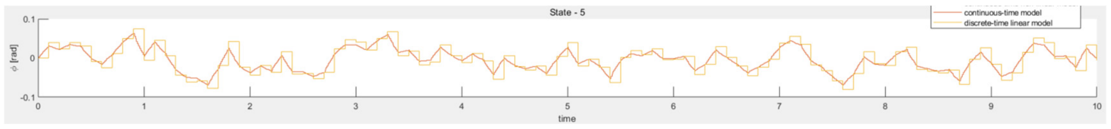


Figure 1.9. Results of state 5 when discretizing by Euler approximation

It seems that many researchers or engineers are more interested in using the discrete-time linear model. I think the first reason is that discrete-time linear model is much easier to compute. If using continuous-time model, we will use integral, which increase the complexity of computation.

Furthermore, I think the discrete-time model can be better when doing simulation. In computer, all the data is stored as discrete value. So, even for the continuous-time linear model we solve in this case study, **SIMULINK** may do the integral by adding up multiple discrete data. (It's my first time to use **SIMULINK**, I am not quite sure how it works. But I believe it should be the rough idea). Thus, there can't be a perfectly pure continuous-time simulation in computer. Since we will discretize the data when computing the results of continuous-time linear model, it is more convenient to directly using discrete model, which does not lose precision.

The last reason is that doing integral in continuous model may require much more discrete data. In a complex model, this will increase the running time and space for data storage.

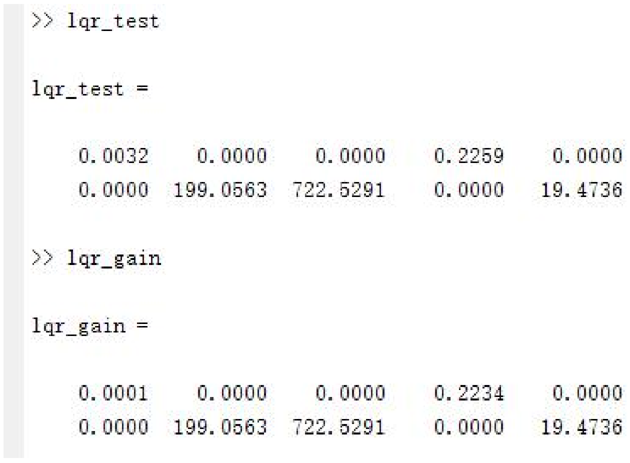
2. Case study 2

This case study is based on the work in case study 1. In this one, we are using the discrete-time linear model and design an LQR controller and an observer based on that model. Then, the observer is implemented to reconstruct the non-measured state and the controller computes the control input based on the estimated states from the observer. The control input is to be implemented to control the nonlinear system.

2.1 Design of LQR controller

The LQR gain is computed based on the proposed $\mathbf{Q_1}$ and $\mathbf{Q_2}$ in the document, where $\mathbf{Q_1}$ is diag(1e-5, 50, 0.5, 0.5, 0.5) and $\mathbf{Q_2}$ is diag(1, 2*1e-5). The value of the LQR gain is showed in figure 2.1. The **lqr_test** is the gain calculated by the command **dlqr ( )**. We can see that the error between the gain obtained by my code and that obtained by **dlqr ( )** is tiny. The LQR gain should be accurate.

In my case, I didn't simply set the number of the iteration. A tolerance valued as 1e-3 is defined. The number of tolerance iteration (**tol_iter**) will be added by 1 if the difference of each elements between one LQR gain and that computed in its last iteration is less than the tolerance. The iteration won't stop until the number of tolerance iteration reach ten. This will guarantee that S will converge. The evolution of the singular values of S is in figure 2.2.



```
>> lqr_test

lqr_test =

    0.0032    0.0000    0.0000    0.2259    0.0000
    0.0000  199.0563  722.5291    0.0000   19.4736

>> lqr_gain

lqr_gain =

    0.0001    0.0000    0.0000    0.2234    0.0000
    0.0000  199.0563  722.5291    0.0000   19.4736
```

Figure 2.1. LQR gains computed by solving equation iteratively and by dlqr ( )

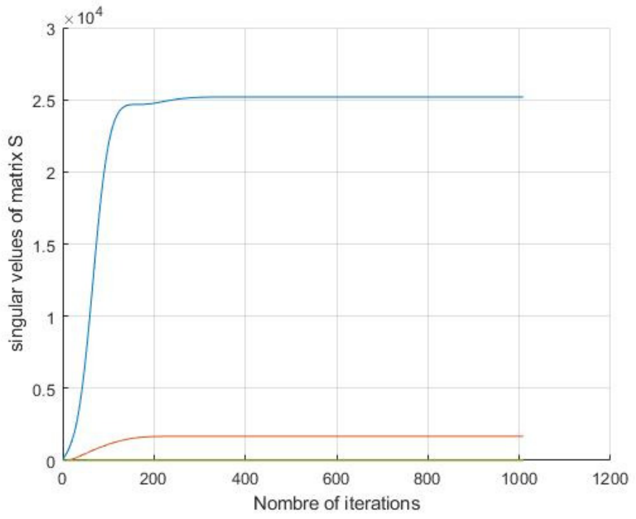


Figure 2.2. The evolution of the singular values of S

2.2 Design of the observer

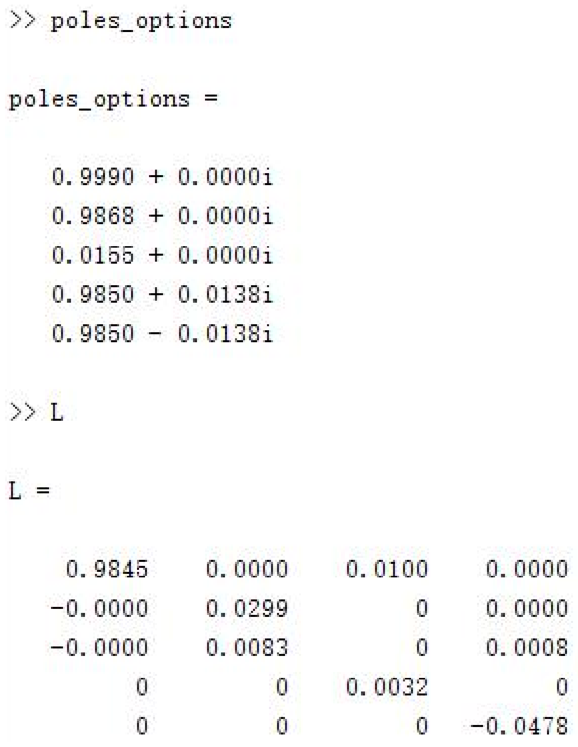
Initially, our C matrix is just identity matrix. That enables us to observe all states directly from the outputs. Thus, we use a new matrix C', which enables the outputs **y'** consists of all the states except x_3. The value of C' is:

$$\boldsymbol{C'} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

By checking the rank of observability matrix, I found that the new system is still observable. Now, it comes to the design of the observer. I first calculated 99.9% of the poles of the closed control loop and set it to be the poles of the observation loop as proposed in the document. Then, I calculated the observer gain by the code:

```
L = place(Phi',Cprime',selected_poles)'
```

This could work because the fact that the eigenvalues of (A+LC) are the same as those of (A^T+$C^T L^T$). The value of the observation close loop poles and observer gain is displayed in figure 2.3 where **poles_options** represents the close loop poles and **L** means the observer gain.



```
>> poles_options

poles_options =

   0.9990 + 0.0000i
   0.9868 + 0.0000i
   0.0155 + 0.0000i
   0.9850 + 0.0138i
   0.9850 - 0.0138i

>> L

L =

    0.9845    0.0000    0.0100    0.0000
   -0.0000    0.0299         0    0.0000
   -0.0000    0.0083         0    0.0008
         0         0    0.0032         0
         0         0         0   -0.0478
```

Figure 2.3. The observation close loop pole and the observer gain

2.3 Final SIMULINK scheme and the submodules

Figure 2.4 shows the overall SIMULINK scheme. Figure 2.5 to figure 2.9 are the submodules I completed.

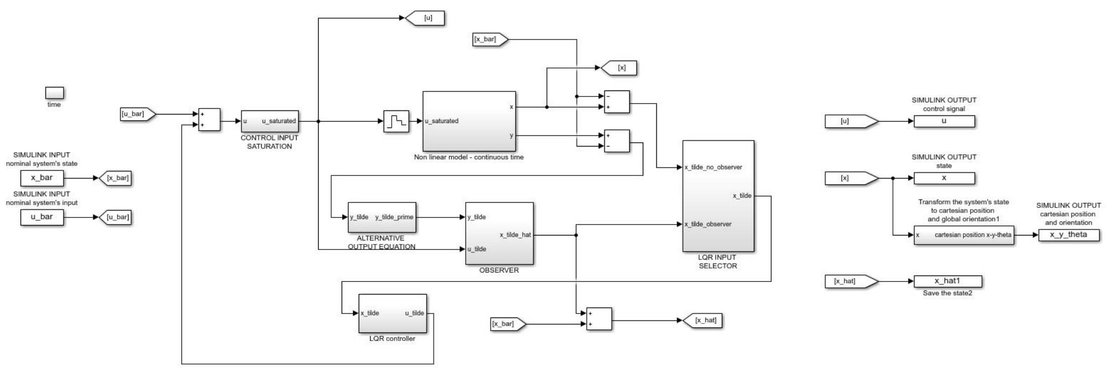


Figure 2.4. Final SIMULINK scheme

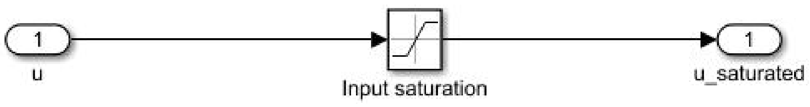


Figure 2.5. Submodule: Input Saturation

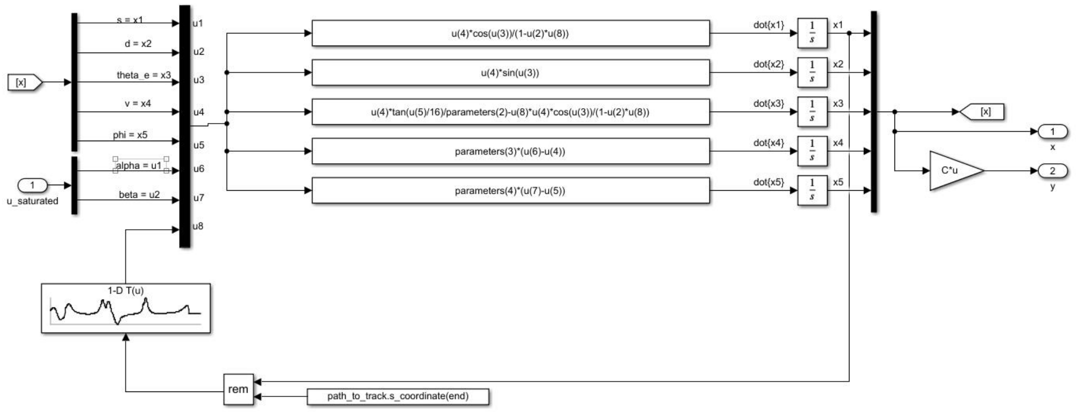

[x]
s = x1
d = x2
theta_e = x3
v = x4
phi = x5
1
u_saturated
alpha = u1
beta = u2
u1
u2
u3
u4
u5
u6
u7
u8
u(4)*cos(u(3))/(1-u(2)*u(8))
u(4)*sin(u(3))
u(4)*tan(u(5)/16)/parameters(2)-u(8)*u(4)*cos(u(3))/(1-u(2)*u(8))
parameters(3)*(u(6)-u(4))
parameters(4)*(u(7)-u(5))
dot{x1}
dot{x2}
dot{x3}
dot{x4}
dot{x5}
1/s
x1
x2
x3
x4
x5
[x]
1
x
C*u
2
y
1-D T(u)
rem
path_to_track.s_coordinate(end)

Figure 2.6. Submodule: Non-linear model – continuous time

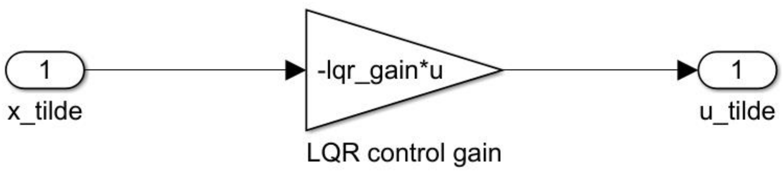


Figure 2.7. Submodule: LQR controller

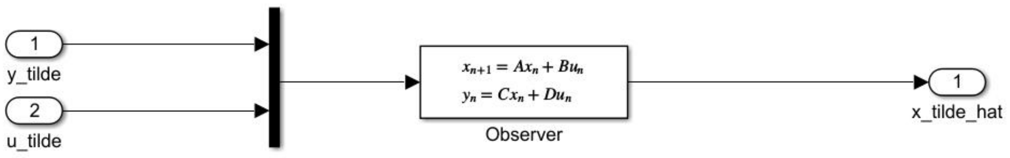


Figure 2.8. Submodule: Observer

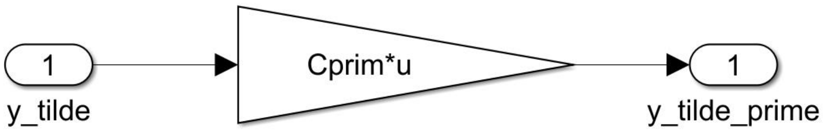


Figure 2.9. Submodule: Alternative output equation

2.4 Simulation results for the proposed values.

The simulation results using the first reference path for the proposed values is shown in figure 2.10 – figure 2.12. The performance of the LQR controller without the observer is quite good. The vehicle just follows the reference with slight deviation, which means the proposed parameters in penalty matrix $\mathbf{Q_1}$ and $\mathbf{Q_2}$ has been tune to a proper value.

Nevertheless, the observer gets some problem on states x_2, x_3 and x_4. Unneglectable deviations appear between the estimated states and the real states. Therefore the parameters for the observer needs to be further tuned. Details will be discussed in 3.7.

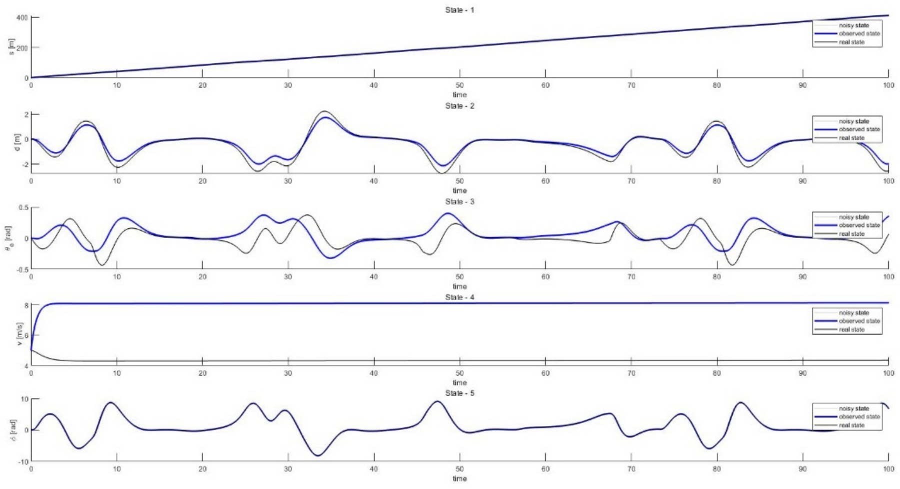
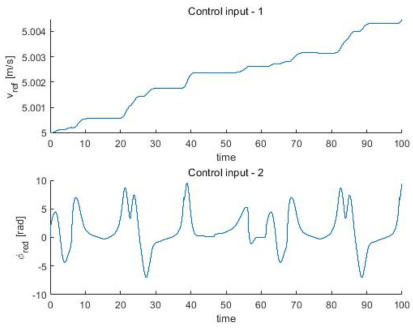
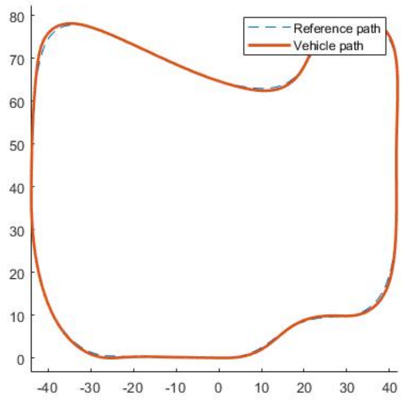


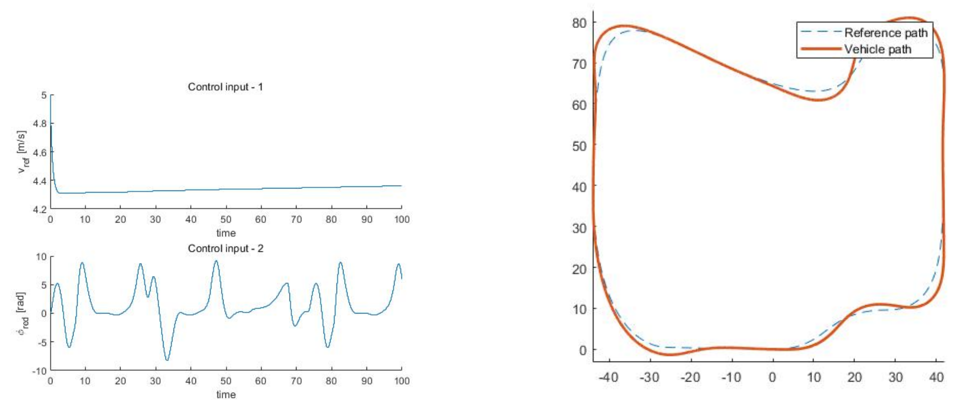
Figure 2.10. Observed state and real state obtained by proposed value





(a) Inputs **(b)** Path

Figure 2.11. Performance of the LQR controller for the proposed values (without the observer)



(a) Inputs **(b)** Path

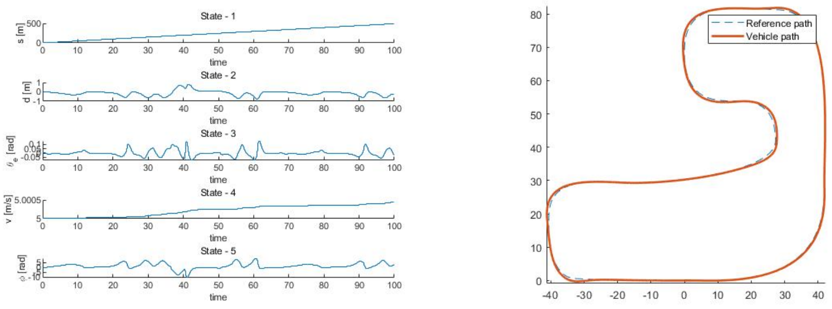
Figure 2.12. Performance of the LQR controller for the proposed values (including the observer)

2.5 Analysis on the value of $\mathbf{Q_1}$

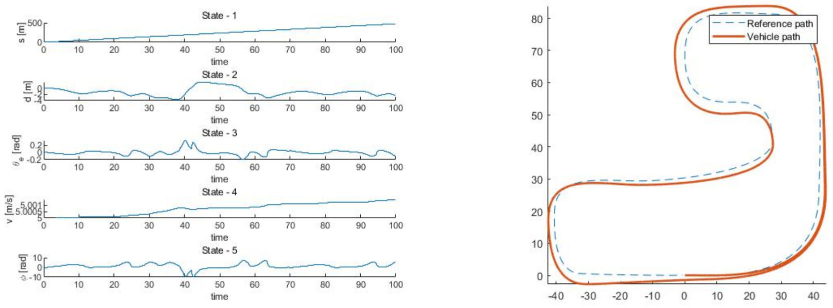
The proposed value of $\mathbf{Q_1}$ penalized little on x_1. That is because the main goal of our controller is to make sure the automated vehicle will follow the paths. The state x_1 physically means the distance along the path and deviation on the distance, i.e. $\tilde{x}_1$, will not cause as much as deviation on the path than deviation on other states. Thus state x_1 is less important than other states in our controller and we should penalize less on it. Besides, we also penalize properly on state x_4, i.e. the velocity of the vehicle, which can help control x_1. The good control on the velocity can guarantee the control on x_1.

Also, I tried a different $\mathbf{Q_1}$ penalizing increasing highly on heading deviation. In this section, the second reference path is used for simulation for a better illustration of the impact of penalizing highly on heading deviation error. For the new settings of the cost function, the first weight for the state cost function is always 0.00001 and the fourth weight and fifth weight are always set to be 0.5, while $\mathbf{Q_2}$ always keeps the same. By setting more and more importance on heading error, the controller will increasingly focus on fixing the deviation of heading angle and other states will be views as less and less important, even nearly ignored in the worst case.

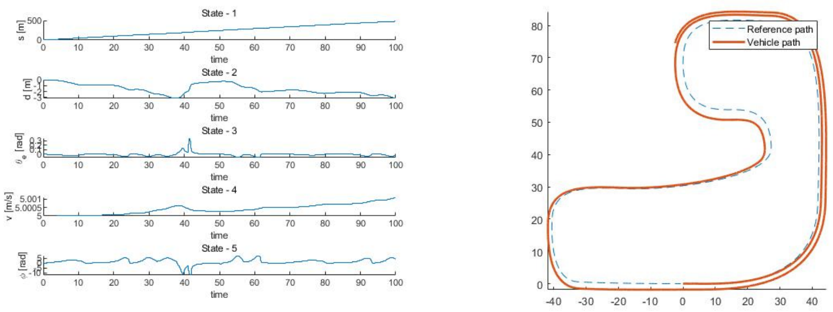
The proposed parameters are set in the first experiment. Illustrated in figure 2.13 (a), the vehicle sticks to the reference path very well, just as good as its performance on the first path. By increasing the penalty of x_3 error and setting the penalty of x_2 to be equal to that of x_4 and x_5, the controller turns to pay more attention in fixing heading error so that higher lateral deviations appears in figure 2.13 (b). If we continue to increase the penalty of x_3, the controller will nearly ignore the deviations of other states and focus on solving the heading error. As showed in figure 2.13 (c), the controller only keeps the heading angles and just let the lateral error grows liberally.



(a) $\mathbf{Q_1}$ = diag(0.00001, 50, 0.5, 0.5, 0.5), $\mathbf{Q_2}$ = diag(1, 0.00002)



(b) $\mathbf{Q_1}$ = diag(0.00001, 0.5, 500, 0.5, 0.5), $\mathbf{Q_2}$ = diag(1, 0.00002)



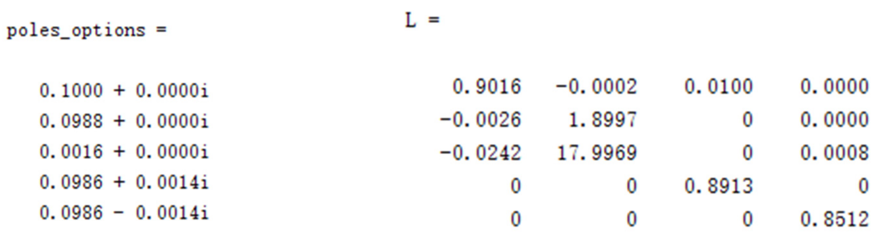
(c) $\mathbf{Q_1}$ = diag(0.00001, 0.5, 5000, 0.5, 0.5), $\mathbf{Q_2}$ = diag(1, 0.00002)

Figure 2.13. Performance of the LQR controller with different penalty matrices

2.6 Analysis on the placement for the observation close loop poles appropriate

I think the proposed placement for the observation close loop poles is not appropriate. As, we can see from figure 2.10, the estimated states don't match the real states very well. The reason for the bad performance of the observer should be wrongly chosen poles. The proposed poles are 0.99, 0.98, 0.01, 0.99 + 0.008i, 0.99 - 0.008i. Though they are all set to be inside the unit circle, most of them are just to close to the boundary, i.e. near to 1. Thus, the error between the real states and the estimated one will converge to 0 very slowly

Unlike the controller, the dynamic of the observer has neither physical limitation nor amplitude of the signals. The states in the observer are just estimated one. Hence, choosing poles that let the observer converges as fast as possible is a better option. The fastest one is set them all to be zero, i.e. the dead-beat observer. However, too fast convergence may lead the system to be sensitive to noise or bad transients. Though there won't be any noise in the simulation, the poles are set to be 10% of the poles of the closed control loop for a more general and robust observer. The specified values of observer poles and matrix **L** are listed in figure 2.14.


```
poles_options =

   0.1000 + 0.0000i
   0.0988 + 0.0000i
   0.0016 + 0.0000i
   0.0986 + 0.0014i
   0.0986 - 0.0014i
```

```
L =

    0.9016   -0.0002    0.0100    0.0000
   -0.0026    1.8997         0    0.0000
   -0.0242   17.9969         0    0.0008
         0         0    0.8913         0
         0         0         0    0.8512
```

Figure 2.14. The modified observation close loop pole and the observer gain

In this section, we use the third reference path to do simulations. The simulations obtained by observer with proposed poles and new poles are illustrated in figure 2.15 and figure 2.16. We can see that the modified observer estimates the states more accurate.

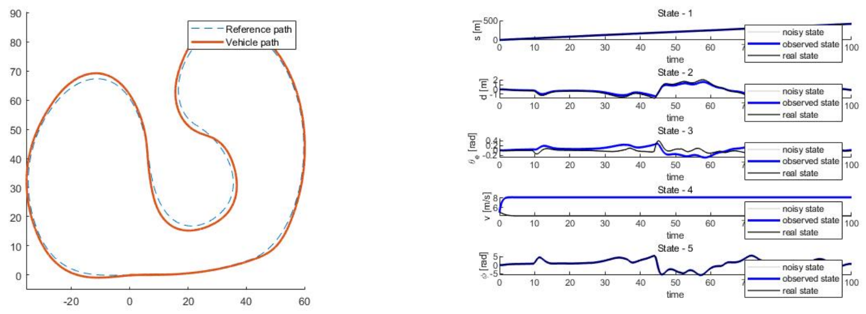


Figure 2.15. Performance of the observer with proposed poles

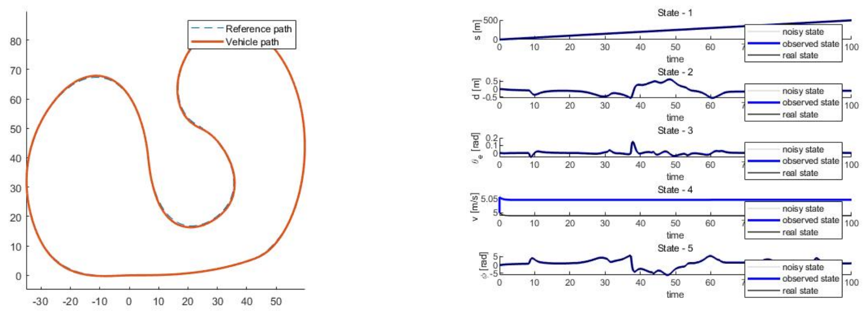


Figure 2.16. Performance of the observer with new poles