Multivariable control and coordination systems

Case study report

Chevalley Arthur 283178

Autumn 2020

Contents

1 Modeling, linerization and discretization

1.1 Vehicle's model

The aim of this case study is to model and control a car following a path. The vehicle's model is formated as follows[1] :

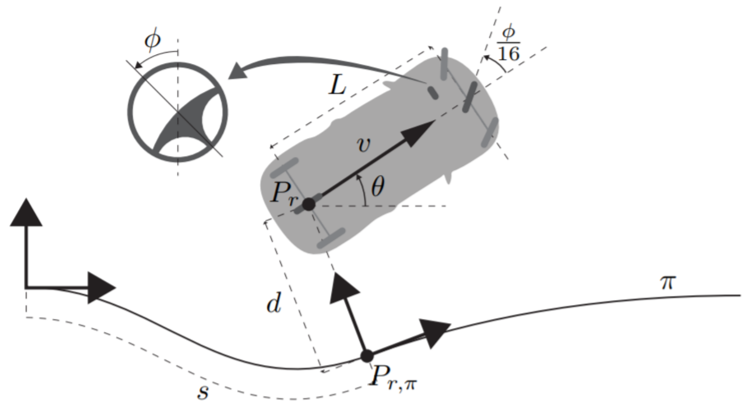


Figure 1.1.1: Car model

With : s the distance along the path corresponding to the vehicle's projections, v the speed, θ_e the heading error, d the lateral error, k(s) the curvature of the path, L the vehicle's length, ϕ the steering wheel, σ_v dynamic with which speed reference is follow, σ_{phi} dynamic with which steering referece is follow, ϕ_{ref} steering wheel's reference position and v_{ref} the longitudinal speed reference. Putting it all together gives the model dynamics equations:

$$\dot{s} = \frac{v \cdot cos(\theta_e)}{1 - d \cdot k(s)}$$

$$\dot{d} = v \cdot sin(\theta_e)$$

$$\dot{\theta_e} = \frac{v}{L} \cdot tan(\frac{\phi}{16}) - k(s) \cdot \dot{s}$$

$$\dot{v} = \sigma_v \cdot (v_{ref} - v)$$

$$\dot{\phi} = \sigma_\phi \cdot (\phi_{ref} - \phi)$$

With:

$$k(s) = 1 \cdot e^{-10}, L = 4, \sigma_v = 1, \sigma_\phi = 5, v_{ref} = 5$$

[1]Figure from the case-study description file

1.2 Non linear model in standard form

In order to compute the state space model we need to have the state and input vectors. The following were chosen:

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5] = [s, d, \theta_e, v, \phi],\ \mathbf{u} = [\alpha, \beta] = [v_{ref},\ \phi_{ref}]$$

With α the speed reference and β the steering wheel angle refrence.

In order to compute the system dynamics, the physical equations must be reformulate in the following form:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

To achieve this, the physical model was express using the states and inputs which gives the following state space representation:

$$\dot{\mathbf{x}} = \begin{cases} \dot{x_1} = \cos(x_3) \cdot \frac{x_4}{1-x_2 \cdot k} \\ \dot{x_2} = x_4 \cdot \sin(x_3) \\ \dot{x_3} = \tan(\frac{x_5}{16}) \cdot \frac{x_4}{L} - k \cdot \dot{x_1} = \tan(\frac{x_5}{16}) \cdot \frac{x_4}{L} - k \cdot \cos(x_3) \cdot \frac{x_4}{1-x_2 \cdot k} \\ \dot{x_4} = \sigma_v \cdot (u_1 - x_4) \\ \dot{x_5} = \sigma_\phi \cdot (u_2 - x_5) \end{cases} \tag{1.2.1}$$

1.3 Linerization

As the model is nonlinear, one must linearize it. It was chosen to use a nominal trajectory linerizaiton and giving the perfect path as the reference.

By looking at the equations we can see that it wouldn't be a good idea to linearize around a single point because there is a time dependancy. Indeed, the trajectory change over time so one must use a nominal trajectory and not a nominal point for the linerization.

The nominal trajectory, following $\dot{\bar{x}} = \mathrm{f}(\bar{x},\ \bar{u})$, represents the constant-speed, perfect tracking situation with a constant curvature of the path. This means lateral and heading error equals to zero, as well as their derivatives, which gives: $\bar{x_2} = \bar{x_3} = 0$. In addition, the system being at a constant speed:

$$\bar{x_4} = v_{ref}\ \ and\ \ \dot{\bar{x_4}} = 0$$

By remplacing the state by the nominal state in the non linear model, i.e. equation 1.2.1, one can compute the following full nominal trajectory and nominal inputs:

$$\bar{\mathbf{x}} = \begin{cases} \bar{x}_1 = v_{ref} \cdot t \\ \bar{x}_2 = 0 \\ \bar{x}_3 = 0 \\ \bar{x}_4 = v_{ref} \\ \bar{x}_5 = 16 \cdot tan(k \cdot L)^{-1} \\ \bar{u}_1 = v_{ref} \\ \bar{u}_2 = 16 \cdot tan(k \cdot L)^{-1} \end{cases} \tag{1.3.1}$$

Note that the state $\bar{x_1}$ is compute ny integration of $\dot{\bar{x_1}}$ Using the nominal trajectory linerization we can compute A and B, the matrices containing the partial derivative evaluated at the nominal state, i.e. equation 1.3.1, which gives us the linearized system shown in equation 1.3.2.

$$A = \frac{\partial f(\mathbf{x}}{\partial \mathbf{x}}\bigg|_{\substack{x=\bar{x}\\u=\bar{u}}} = \begin{pmatrix} 0 & k \cdot v & 0 & 1 & 0 \\ 0 & 0 & v & 0 & 0 \\ 0 & -v \cdot k^2 & 0 & 0 & v \cdot \frac{1+(k \cdot L)^2}{16 \cdot L} \\ 0 & 0 & 0 & -\sigma_v & 0 \\ 0 & 0 & 0 & 0 & -\sigma_s \end{pmatrix}, \quad B = \frac{\partial f(\mathbf{u}}{\partial \mathbf{u}}\bigg|_{\substack{x=\bar{x}\\u=\bar{u}}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \sigma_v & 0 \\ 0 & \sigma_s \end{pmatrix} \tag{1.3.2}$$

With $\mathbf{x}$ the state vector, $\mathbf{u}$ the input vector, $(\bar{x}, \bar{u})$ the nominal trajectory and $f(\mathbf{x}, \mathbf{u}) = \dot{\mathbf{x}}$ the function representing the non linear behavior of the system.
Finally as one have $\mathbf{y} = \mathbf{x}$, the matrix C is an identity matrix I_{5x5} and the matrix D is a 5 by 2 matrix filled with zeros.
To have the linear state space representation, the variable must be changed according to: $\tilde{x} = x - \bar{x}$, $\tilde{u} = u - \bar{u}$ and $\tilde{y} = y - \bar{y}$. Putting it all together gives us the continuous linearize dynamics:

$$\begin{cases} \dot{\tilde{x}} = A \cdot \tilde{x} + B \cdot \tilde{u} \\ \tilde{y} = I \cdot \tilde{x} \end{cases} = \begin{cases} \dot{\tilde{x}} = \begin{pmatrix} 0 & k \cdot v & 0 & 1 & 0 \\ 0 & 0 & v & 0 & 0 \\ 0 & -v \cdot k^2 & 0 & 0 & v \cdot \frac{1+(k \cdot L)^2}{16 \cdot L} \\ 0 & 0 & 0 & -\sigma_v & 0 \\ 0 & 0 & 0 & 0 & -\sigma_s \end{pmatrix} \tilde{x} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \sigma_v & 0 \\ 0 & \sigma_s \end{pmatrix} \tilde{u} \\ \tilde{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tilde{x} \end{cases} \tag{1.3.3}$$

1.4 Discretization

1.4.1 Discretization using Euler method

Now that the system is linearize, one need to discretize it in order to use it. Hence, the euler appoximation was used and compared with the *C2D* matlab function.
To discretize using the Euler approximation the following equality must be respected:

$$\dot{x}_k \approx \frac{x_{k+1} - x_k}{h}$$

Applied to our system one have the following discretization of the linearize continuous time model:

$$\frac{x_{k+1} - x_k}{h} = A \cdot x_k + B \cdot u_k \tag{1.4.1}$$

Which can be reformulate:

$$\begin{cases} \tilde{x}_{k+1} = (h \cdot A + I) \cdot \tilde{x_k} + h \cdot B \cdot \tilde{u_k} \\ \tilde{y_k} = C \cdot \tilde{x}_k \end{cases} \tag{1.4.2}$$

Finally the Phi and Gamma matrices are:

$$\phi = h \cdot A + I = \begin{pmatrix} 1 & h \cdot k \cdot v & 0 & h & 0 \\ 0 & 1 & h \cdot v & 0 & 0 \\ 0 & h \cdot (-v) \cdot k^2 & 1 & 0 & h \cdot v \cdot \frac{1+(k \cdot L)^2}{16 \cdot L} \\ 0 & 0 & 0 & h \cdot (-\sigma_v) + 1 & 0 \\ 0 & 0 & 0 & 0 & h \cdot (-\sigma_s) + 1 \end{pmatrix}$$

$$\Gamma = h \cdot B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ h \cdot \sigma_v & 0 \\ 0 & h \cdot \sigma_s \end{pmatrix}$$

Hence the linearized & discretized system is:

$$\begin{cases} \tilde{x}_{k+1} = \phi \cdot \tilde{x}_k + \Gamma \cdot \tilde{u}_k \\ \tilde{y}_k = C \cdot \tilde{x}_k \end{cases} \tag{1.4.3}$$

1.4.2 Discretization using the Taylor-series approximation of the matrix exponential

Alternativly one can choose to discretize the system by computing the Taylor-series approximation of the matrix exponential, which is the ideal discretization. This is written as:

$$\phi = e^{Ah} = I + \sum_{n=1}^{\infty} \frac{(Ah)^n}{n!} \tag{1.4.4}$$

$$\Gamma = \int_0^h e^{A\nu} d\nu \cdot B = [Ih + \sum_{n=1}^{\infty} \frac{A^n h^{n+1}}{(n+1)!}]B \tag{1.4.5}$$

By isolating $\Psi = I + \sum_{n=1}^{\infty} \frac{(Ah)^n}{(n+1)!}$ from equation 1.4.5, one can find $\Gamma = \Psi h B$ and $\phi = I + Ah\Psi$. In order to compute this with MATLAB, one can change Ψ computation into a simple *for* loop and check when convergence is reached. This is done by reformulating Ψ as follow:

$$\Psi_{n+1} = \Psi_n + \frac{(Ah)^n}{(n+1)!}, \qquad \forall n \geqslant 0 \tag{1.4.6}$$

By forcing Ψ_0 equals to a zero matrix and iterating until converge is reached and Ψ is found. As the second terms decrease according to $\mathcal{O}(n!)$ the converge is quickly reached. In my case, even though around five iterations already gives really good approximation, fifteen iterations were chosen in order to have really good results. As the computational time being close for five or fifteen, the choice of fifteen instead of five does not change much. Finally, computing $\Gamma = \Psi h B$ and $\phi = I + Ah\Psi$ gives the discretize system matrices.

1.4.3 Discretization using matlab C2D function

In order to evaluate the discrization we compute the discretized model with a zero order hold using the function *C2D*. In the next section the matrices obtained with both methods will be compared.

2 Model implementation and simulation

2.1 ϕ & Γ comparaison

As we have computed the ϕ & Γ matrices in three different ways, we have ϕ_{Euler} & Γ_{Euler} computed with the euler approximation, , ϕ_{Taylor} & Γ_{Taylor} computed with the Taylor-series approximation of the exponential matrix and ϕ_{Matlab} & Γ_{Matlab} computed with the matlab functions *C2D*. First of all, the Taylor-series approximation is almost equivalent to *C2D* result. It can be forecast as the equation 1.4.4 & 1.4.5 are the exact solution of the disretized system. Hence, the result of Ψ calculated with equation 1.4.6 is really close to the optimal one, calculated by matlab. Consequently the errors are too small to be shown with the same digits as the matrices here. Hence, they will not be shown here as it would only congest this report. However, the euler and *C2D* method gives different result and are shown here.

$$\phi_{Euler} = \begin{pmatrix} 1.000 & 0.000 & 0.000 & 0.100 & 0.000 \\ 0.000 & 1.000 & 0.500 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.000 & 0.000 & 0.008 \\ 0.000 & 0.000 & 0.000 & 0.900 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.500 \end{pmatrix}, \Gamma_{Euler} = \begin{pmatrix} 0.000 & 0.000 \\ 0.000 & 0.000 \\ 0.000 & 0.000 \\ 0.100 & 0.000 \\ 0.000 & 0.500 \end{pmatrix}$$

$$\phi_{Matlab} = \begin{pmatrix} 1.000 & 0.000 & 0.000 & 0.095 & 0.000 \\ 0.000 & 1.000 & 0.500 & 0.000 & 0.002 \\ 0.000 & 0.000 & 1.000 & 0.000 & 0.006 \\ 0.000 & 0.000 & 0.000 & 0.905 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.607 \end{pmatrix}, \Gamma_{Matlab} = \begin{pmatrix} 0.005 & 0.000 \\ 0.000 & 0.000 \\ 0.000 & 0.002 \\ 0.095 & 0.000 \\ 0.000 & 0.393 \end{pmatrix}$$

By looking at those matrices one can see that they are really similar. In order to evaluate the differences we define:

$$\phi_{Difference} = |\phi_{Euler} - \phi_{Matlab}| \text{ and } \Gamma_{Difference} = |\Gamma_{Euler} - \Gamma_{Matlab}|$$

$$\phi_{Difference} = \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.005 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.002 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.002 \\ 0.000 & 0.000 & 0.000 & 0.005 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.107 \end{pmatrix}, \Gamma_{Difference} = \begin{pmatrix} 0.005 & 0.000 \\ 0.000 & 0.000 \\ 0.000 & 0.002 \\ 0.005 & 0.000 \\ 0.000 & 0.107 \end{pmatrix} \tag{2.1.1}$$

Taking a look at the matrices, i.e. matrices 2.1.1, one can see that except $\phi_{Difference}(5{,}5)$ and $\Gamma_{Difference}(5{,}2)$ which are in the order of the tenth, the differences are in the order of the thousandth, if not less, which is really good. One can see that both discrtization are really similar even if the euler method is quite simple. For this system, using the Taylor-series approximation is a good alternative as the discretization is really good and quite computationally friendly as convergence is quickly reached in a simpe *for* loop.

2.2 Block diagram

The full openloop system implemented in Simulink is shown in figure 2.2.1 bellow.

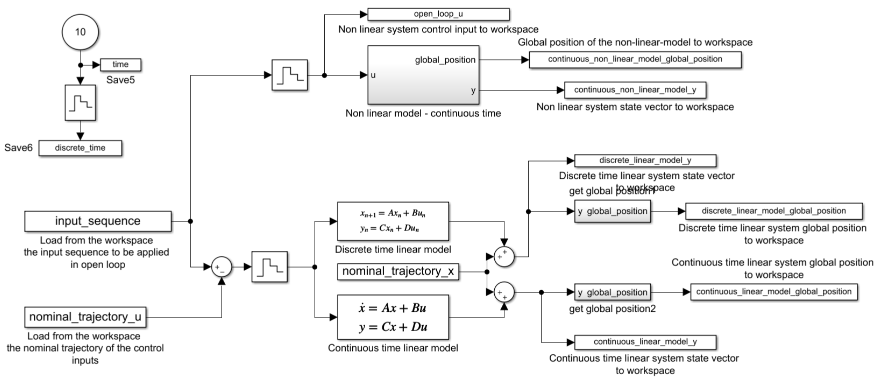

10
time
Save5
Save6
discrete_time
open_loop_u
Non linear system control input to workspace
global_position
u
y
Non linear model - continuous time
Global position of the non-linear-model to workspace
continuous_non_linear_model_global_position
continuous_non_linear_model_y
Non linear system state vector to workspace
discrete_linear_model_y
Discrete time linear system state vector to workspace
get global position1
input_sequence
Load from the workspace the input sequence to be applied in open loop
$x_{n+1} = Ax_n + Bu_n$
$y_n = Cx_n + Du_n$
Discrete time linear model
y global_position
discrete_linear_model_global_position
Discrete time linear system global position to workspace
nominal_trajectory_x
Continuous time linear system global position to workspace
continuous_linear_model_global_position
get global position2
nominal_trajectory_u
Load from the workspace the nominal trajectory of the control inputs
$\dot{x} = Ax + Bu$
$y = Cx + Du$
Continuous time linear model
continuous_linear_model_y
Continuous time linear system state vector to workspace

Figure 2.2.1: Block diagram

2.2.1 Non linear model continuous time block

To implement the non linear model, the equations 1.2.1, described in section 1.2, were used. The equations were simply express in terms of simulink numbering of the input. The resulting implementation is given in figure 4.1.2 bellow.

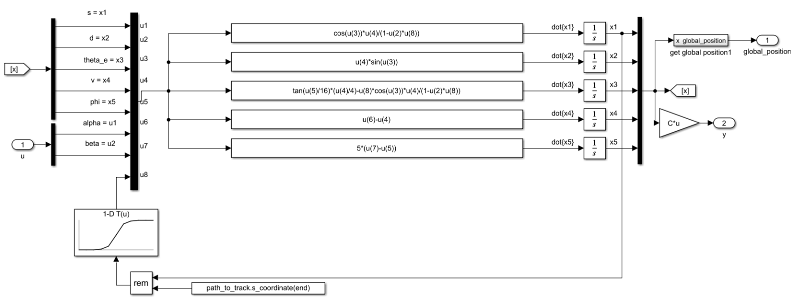


Figure 2.2.2: Non-linear model continuous time bloc

3 Simulation results & analysis

3.1 System comparison for differents inputs

All the simulations were done with the initial state equal to: $x(0) = [0, 0, 0, \bar{x_4}, \bar{x_5}]$.

To have the same output results for all implementation, which can be seen on images 3.1.1, the input u_1 is kept constant, at a value of five, and u_2 is equal to zero. By taking a constant value for both input it is intuitive that all models will have the same output as the nonlinearities will be equal to zero by looking at equation 1.2.1. The simulation result, i.e. figure 3.1.1, shows that all states are equal meaning a perfect equivalence between all models.

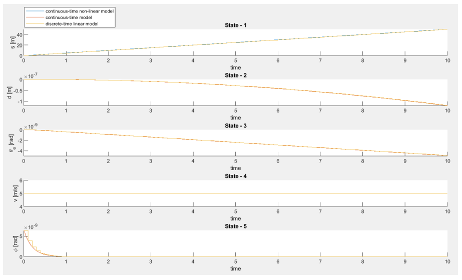


Figure 3.1.1: State for a constant speed reference of 5 $[\frac{m}{s}]$ and a zero angle input

The linear models follow the non linear model for other inputs. Indeed for a constant input and a small enough constant angle the errors will be small enough for a good fit. For u_1 is constant at a value of five for all time step and u_2 equal to zero half the simulation time and then equal to one radian. As the inputs are still close to the nominal inputs, the linearization is good. As we can see on figures 3.1.2 and 3.1.3 the models are still really close to each others. Althought we still have similar results between the models, for bigger angle values the nonlinearites leads to bigger errors and big differences bewteen models, which are not shown here.

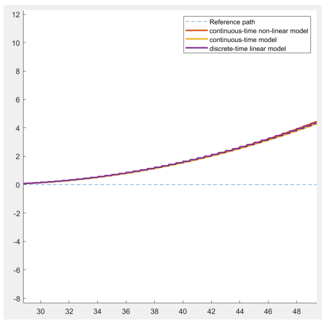


Figure 3.1.2: Simulation result for a constant speed input of 5 $[\frac{m}{s}]$ and a step from zero to one radian for the angle input

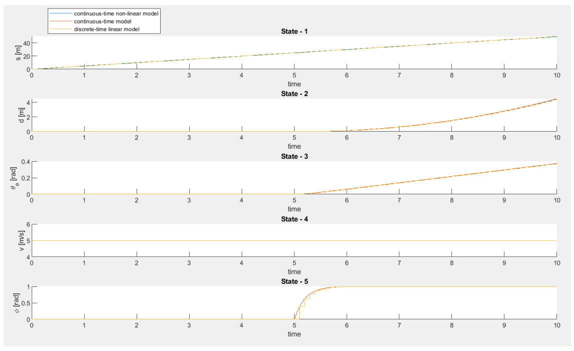


Figure 3.1.3: States for the modelsfor a constant speed input of five $[\frac{m}{s}]$ and a step from zero to one radian for the angle

Altought the results above shows models achieving equivalent results, there is a lot of inputs leading to differences between the models. This can be seen in figure 3.1.4. To have those differences, the inputs shown in image 3.1.5 are those applied in order to see the differences. As for the previous case u_1 is constant at five and u_2 is a ramp fonction increasing monotonically from 0 to 2π. Those inputs being clearly in the nonlinear region, the differences model is not surprising. Indeed, the equation 1.2.1, evaluated with the choosen inputs, clearly keeps the nonlinearites which makes the linear models inefficent. Furthermore, the linearization is not efficent as the nominal trajectory is too far away.

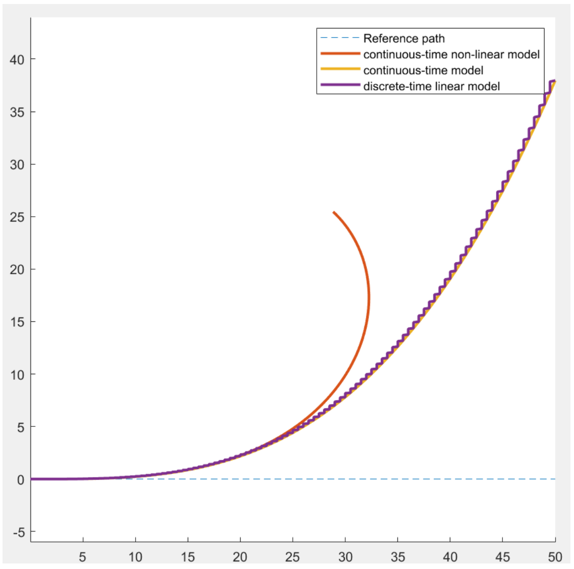


Figure 3.1.4: Differences between implementations

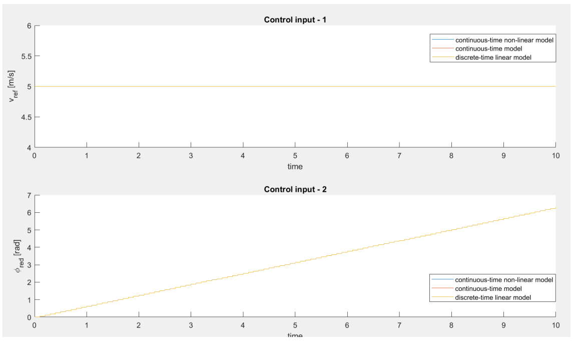


Figure 3.1.5: Input inducing differences

3.2 System analysis & disscusions

From the different simulations one can see that for small, constant, angle input and constant speed the models outputs will be the same. In addition if we have a zero input angle, the linear model will nicely represent the nonlinear one. The speed input can follow a step or ramp profile, with low or high values without damaging the linearize model results. This is intuitve as the nonlinearites are coming from the angle as we can see in equation 1.2.1. However, the various models will quickly diverge if we go too far away of the nominal trajectory. It is intuitive as the model is linearize around this trajectory, the linear model will badly approximate the nonlinear behavior far from the nominal trajectory.

If we compare both linear models, as we can see in figure 3.1.2, they are really close to each other if not exactly the same. The step patern due to the discrete time sampling period is clearly visible. Hence the smiliraties between both linear model highly depends on the sampling period. For small enough sampling time the discrete time linear model will have small steps so it will be closer to the continous time model. In opposition, higher sampling time will leads to bigger steps of the discrete linear model and bigger differences eventhough the values at each sampling period will be the same.

Finally, the discrete time linear model is needed to design digital control as there is no analytical solution to the continuous time general problem. In addition we can forecast the need of a discretize system to compute controller and observer to actually use and control any physical model.

4 Control and Observation

With the physical system ready to be use to actually achieve the path following behavior, one must design a controller and observer. The controller which will *drive* the car is an LQR controller. In addition one must implement a Luenberger observer to estimate the non measurable states.

4.1 LQR design

With the given cost matrices, i.e. equation 4.1.2, and the discrete linear model, i.e. equation 1.4.3, one can compute the LQR controller gain using the *dlqr* function from MATLAB or using the discrete time Ricatti equation given in equation 4.1.1.

$$S = \phi^T \cdot (S - S \cdot \Gamma \cdot (Q_2 + \Gamma^T \cdot S \cdot \Gamma)^{-1} \cdot \Gamma^T \cdot S) \cdot S \cdot \phi + Q_1 \tag{4.1.1}$$

In order to observe the convergence of the Ricatti iteration, one must plot the singular values of the matrix S, c.f. equation 4.1.1. For the given Q_1 and Q_2:

$$Q_1 = \begin{pmatrix} 0.00001 & 0 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{pmatrix}, Q_2 = \begin{pmatrix} 1 & 0 \\ 0 & 0.00002 \end{pmatrix} \tag{4.1.2}$$

We can observe the convergence of the Ricatti equation in Figure 4.1.1. In this simulation the Ricatti equation converged in ~ 200 iterations. However, to ensure the convergence for variations in the cost matrices, Q_1 and Q_2, one would prefer to keep a little margin for the convergence and hence take a higher number of iterations. The number of iterations was set to: $n_{iterations} = 700$.

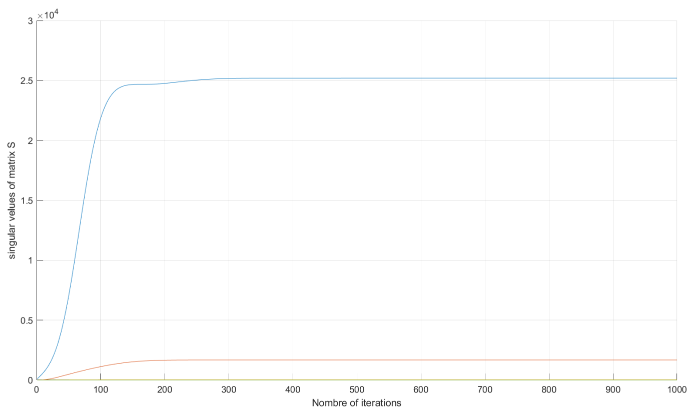


Figure 4.1.1: Ricatti equation convergence depending on the number of iterations

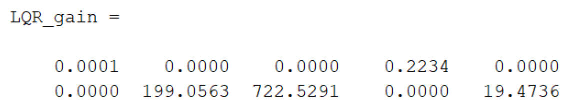
To compute the optimal controller gain using the Ricatti equation one must compute the following equation:

$$LQR_{Gain} = R_{\infty}^{-1} \cdot \Gamma^{T} \cdot S_{\infty} \cdot \phi \tag{4.1.3}$$

The R_{∞} and S_{∞} matrices are the result of the Ricatti equation,i.e. 4.1.1. To solve it the S and R matrices were computed $n_{iterations}$ times. At $n_{iterations}$ the convergence is reached as one can see by looking at the singular values of S, i.e. 4.1.1. Each time the equations 4.1.4 were computed.

$$\begin{cases} R = Q_2 + \Gamma^{T} \cdot S \\ M = S - S \cdot \Gamma \cdot R^{-1} \cdot \Gamma^{T} \cdot S \quad \cdot \Gamma \\ S = \phi^{T} \cdot M \cdot \phi + Q_1 \end{cases} \tag{4.1.4}$$

Putting it all together we finally arrived at a LQR gain equal to:



```
LQR_gain =

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

Figure 4.1.2: LQR gain for initial parameters

4.2 State Observer design

As the system is not fully measurable, an observer must be computed. The aim of the observer is to approximate the non-measurable heading error state, x_3, knowing the other states and the system dynamics. The observer follow the equation 4.2.1 but before computing the various matrices needed to implement the observer, one must check how many states are not observable. This is done by computing the observability matrix Q using *obsv* function with matrix Cprime, the matrix selecting all states except x_3, i.e. $\tilde{y}'(k) = Cprime \cdot \tilde{x}(k)$, given bellow in equation 4.2.1, and ϕ the discrete time matrix. By substrating to the number of state the rank of the Q matrix, the number of non observable states is given. The initial observer states were set to zero as it should be quite close to the intial state of the system given the nominal trajectory, equation 1.3.1, and that $\tilde{x} = x - \bar{x}$. Finally, the equations implementing the observer is given bellow:

$$\begin{cases} \hat{x}_{k+1} = AObs \cdot \mathbf{x}(k) + BObs \cdot \mathbf{u}(k) = [\phi - L \cdot Cprime] \cdot \mathbf{x}(k) + \begin{bmatrix} \Gamma \\ L \end{bmatrix} \cdot \mathbf{u}(k) \\ \hat{y}_k = CObs \cdot \mathbf{x}(k) + DObs \cdot \mathbf{u}(k) = I_{5x5} \cdot \mathbf{x}(k) + \mathbf{0} \cdot \mathbf{u}(k) \\ \mathbf{u}(k) = \begin{bmatrix} \tilde{u}(k) \\ \tilde{y}(k) \end{bmatrix} \end{cases} \tag{4.2.1}$$

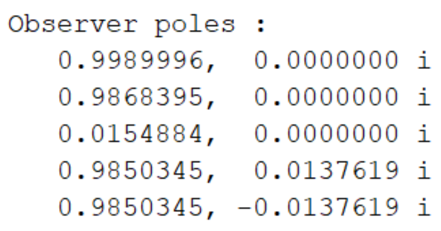
With

- ϕ & Γ the discrete time system dynamics
- L the observer gain computed in equation 4.2.2
- Cprime = $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ in order to select the known state (x_1, x_2, x_4, x_5)
- Note that **x** and **u** are not the real system state and input but the local one

The observer gain L is computed using the *place* function from MATLAB. As the function is construct in order to compute the gain K for an LQR controller, the matrices given to the function must be tweak a little. Indeed, the *place* function is designed to compute the controller K having the closed loop form $\phi_{cl} = (A - B \cdot K)$. Even though the observer gain follow a different dynamics it can be reformulate to have the same form. Indeed, the closed loop form for an observer is $\phi_e = (\phi - L \cdot C)$ which can be reformulated as $\phi_e^T = (\phi^T - C^T \cdot L^T)$ from which we can recognize the same structure as ϕ_{cl}. By identification one can compute equation 4.2.2. Finally, in order to computed the observe gain one must wisely choose the poles. At first, one gives almost the same poles of the as the control closed-loop, $poles_{observer} = 0.99 \cdot poles_{closed-loops}$. With $poles_{closed-loop}$ equals to the eigenvalues of ϕ_{cl}. Finally, the observer gain is implemented as follow using MATLAB:

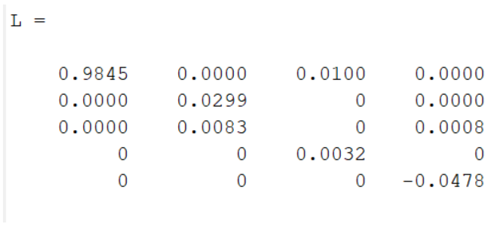
$$L = place(\phi^T, Cprime^T, poles_{observer}) \tag{4.2.2}$$

Using MATLAB, the closed-loop eigenvalues and observer gain are:



```
Observer poles :
    0.9989996,  0.0000000 i
    0.9868395,  0.0000000 i
    0.0154884,  0.0000000 i
    0.9850345,  0.0137619 i
    0.9850345, -0.0137619 i
```

(a) Observer poles chosen in order to be 99 % of $poles_{closed-loop}$



```
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
```

(b) Observer gain computed according to equation 4.2.2

Figure 4.2.1: Observer result computed with matlab

4.3 Simulink implementation & Simulation results

The full closed-loop system implementing the LQR controller and the observer.

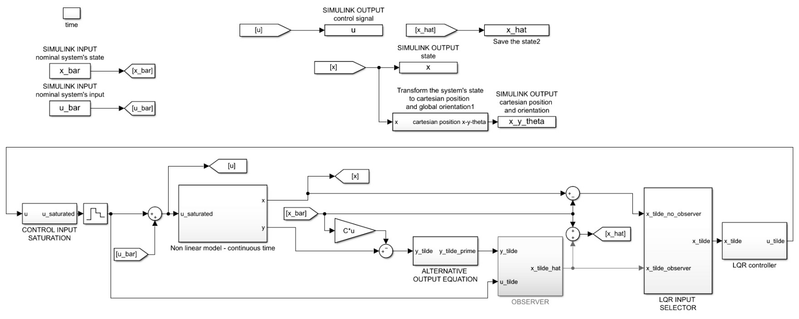
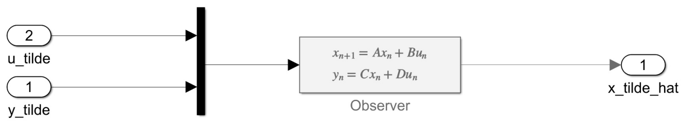
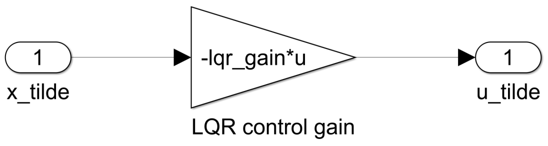


Figure 4.3.1: Observer and LQR controller implementation

In order to implement the observer, one concatenates $\tilde{y}$ and $\tilde{u}$ as it is the inputs of the observer.



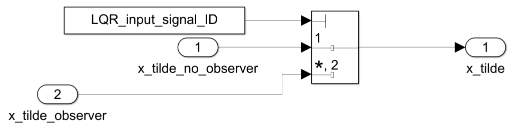
(a) Observer implementation



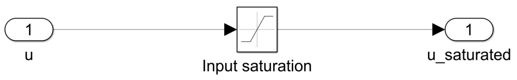
(b) LQR Implementation

Figure 4.3.2: Observer and controller subsystem

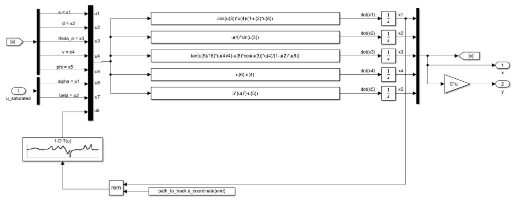
The five other subsystems allow to handle the system non-linearity, c.f. figure 4.3.3c, choose to take into account, or not, the observer, c.f. figure 4.3.3a, the coordinate transforamtion block, c.f. figure 4.3.3d, and finally the saturated module, c.f. figure 4.3.3b, are given bellow. Those subsystem will not be explained here as it is simply the implementation of the equations 1.2.1 and 4.2.1.



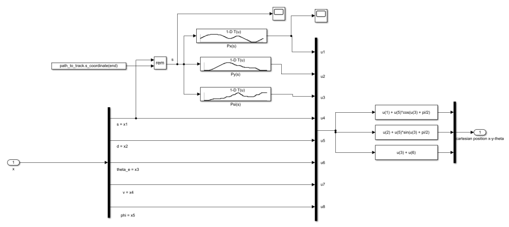
(a) Selector implementation



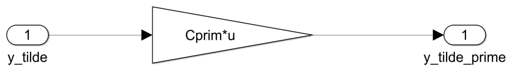
(b) Handle the input saturation



(c) Non-linear continuous time model



(d) Transform from state coordinate to cartesian coordinate

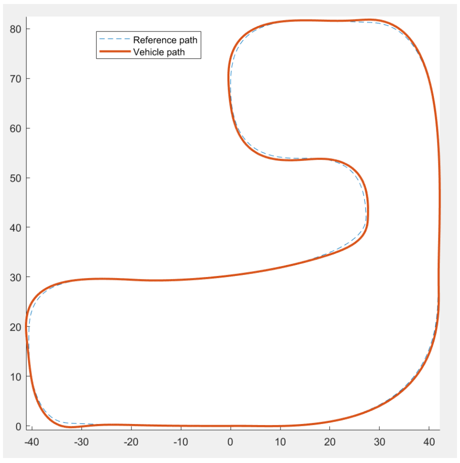


(e) Select the state for the observer, i.e. ignore state x_3

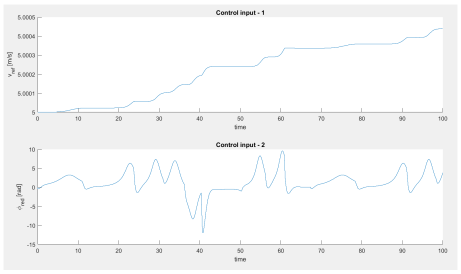
Figure 4.3.3: Models of implementing the system in simulink

4.4 Simulation Results

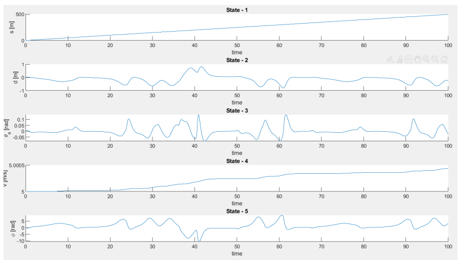
The first simulation is done without using the observer module. As the states are assume to be measurable the figure 4.4.1c shows the exact states. By looking at figure 4.5.1a one can see that the car follow really well the path. However there is some error in the corner as the heading error is not enough penalize in the matrices $Q_{1,2}$.



(a) LQR controller without observer simulation



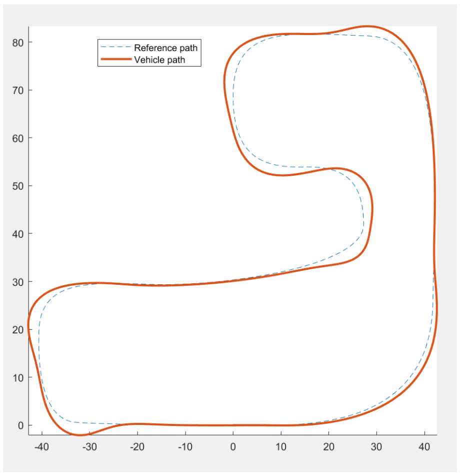
(b) LQR controller inputs evolution



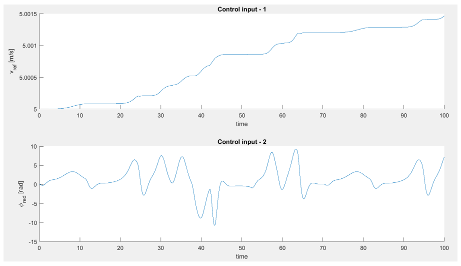
(c) LQR controller states evolution

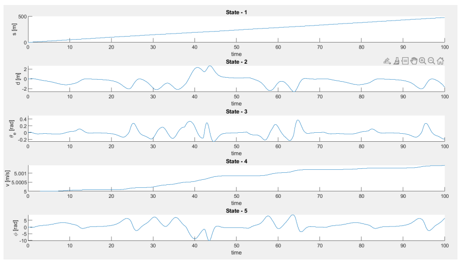
Figure 4.4.1: Simulation result with states & inputs evolution using the observer

The second simulation takes into account the observer in order to evaluate the unknown state. In order using the given Q_1 and Q_2 matrices, c.f equation 4.1.2, the poles placed at 99% of the closed-loop poles. As we can see in figure 4.5.1a,the heading error is too large, for exemple in the corners, and is too far away from the path on some places. Thoses issues are corrected in section 4.5.



(a) LQR controller with observer simulation





(b) LQR controller & Observer inputs evolution

(c) LQR controller & states evolution

Figure 4.4.2: Simulation result with states & Observer inputs evolution using the observer

As one can see on figure 4.4.2a the path is not well followed which means that the states are different from the exact ones. To enhance those deviations, a comparison of the state is shown on figure 4.4.3 where we can clearly see that the estimated state, the third one, is not very well estimated for sudden changes. Those changes are for exemple when a big turn must be done.

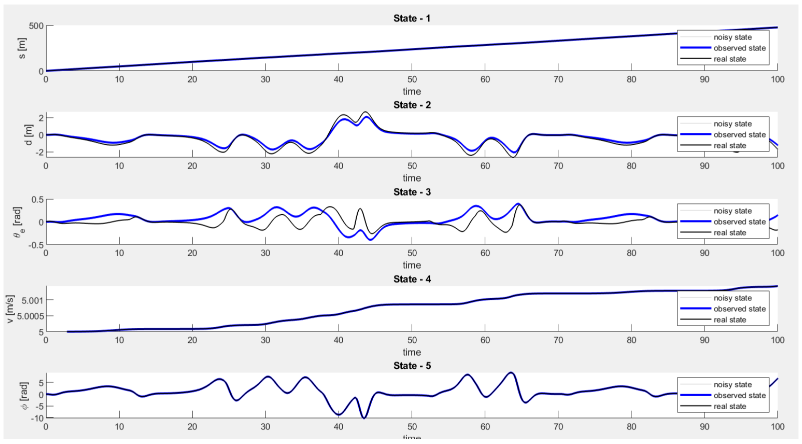
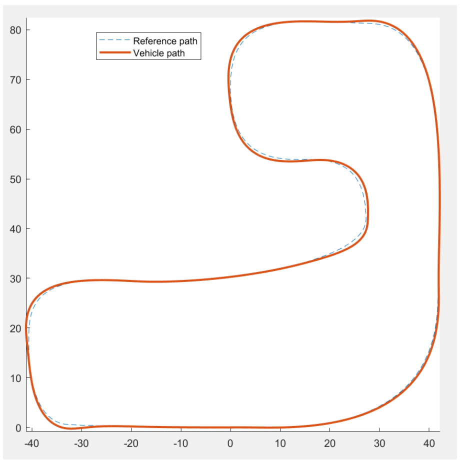


Figure 4.4.3: Comparison of the states evolution

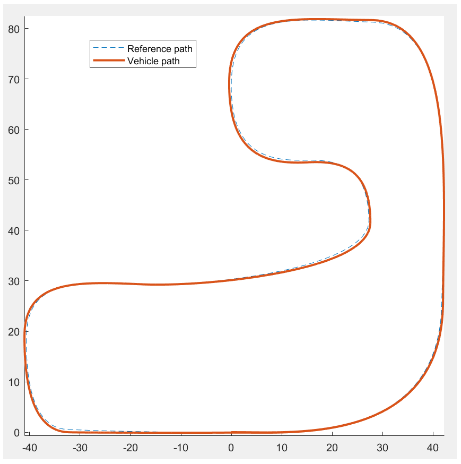
4.5 LQR controller and Observer tweaking

The proposed Q_1 matrix , i.e. equation 4.1.2, gives very little importance to the first state, medium importance fot the last three states and a very high importance to the second one. This implies that the model focus on having a good lateral tracking of the path, meaning a small lateral error but do not gives much imortance to tracking accurately the travelled distance along the path. This make sense as one want the car to be really close to the road meaning having a small lateral error but if the car is under/over evaluating the traveled distance it is not that important.

To have a better angle following behaviour, the weight associated to θ_e in the Q_1 matrix, i.e. indice (3,3) in matrix Q_1 from equation 4.1.2, must be tweak. There is no need to modify the Q_2 matrix as it only has an impact on the inputs cost. By putting a big parameter for the heading error, here 10000, this state become much more important. By doing this, one can see, looking at figure 4.5.1b, that the heading direction is closer to the reference path hence the modification of the cost matrix was successful. Even though it is slightly better, there is not a big difference as the inital model is already really good. However, the heading is better but the distance to the path is not better as the other costs are now way smaller, and then less important, than the heading error.



(a) LQR controller with initial cost matrices



(b) LQR controller with tuned cost matrices

Figure 4.5.1: Comparison of the result for different Q_1 weigth in fully measurable system

To achieve almost perfect tracking one can increase the lateral error cost, in my case to 2500, i.e. weight (2,2) of matrix Q_1, and keeping a high cost for the heading error. This forces to have better accuracy on heading and lateral tracking while the other cost are much smaller. This can be done as the remaining three states are less important. Indeed, as mentionned earlier errors on the first state does not influence much the path following behvior. Similarly angular and linear speed tracking are not as important as the lateral and heading error in case of tracking. The resulting simulation using those parameters is given in figure 4.5.2 and one can see almost perfect tracking.

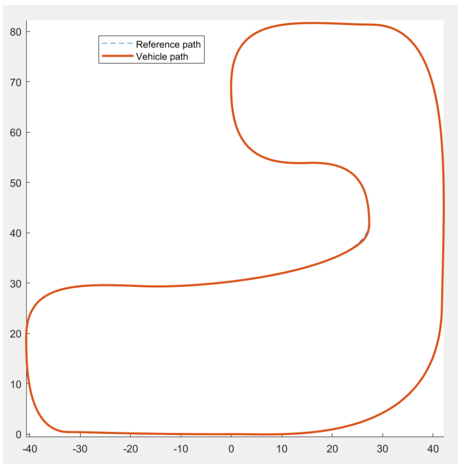
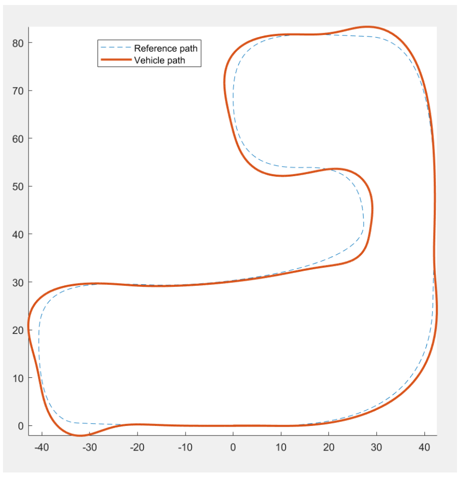
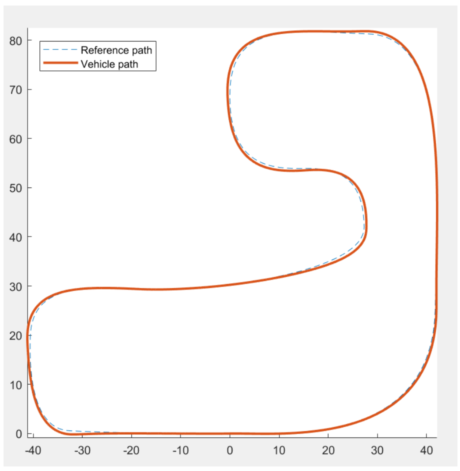


Figure 4.5.2: Cost tuned for perfect tracking with fully measurable system

In order to tweak the observer gain, the poles can be changed. Having poles closer to the closed-loop poles gives a stronger noise robustness but longer settling time. In opposition, for poles further away the settling time is shorter but the system is less robust. As the test are made using a perfect simulation, smaller poles will gives best results. By trying multiple percentage, 5 % was choosen in order to have $poles_{observer} = 0.05 \cdot poles_{closed-loops}$ which gaves the best behaviour following the path. As one can see when comparing figure 4.5.3b and 4.5.3a .



(a) Simulation with intial percentage of the closed loop poles



(b) Simulation with the percentage of the closed loop poles ajusted for the observer implementation

Figure 4.5.3: Comparison of the simulation with different poles for the observer

Finally, to achieve an even better tracking with the observer, choosing the same cost matrices as for the LQR controller with fully measurable states leads to really good path following as we can see on figure 4.5.4.

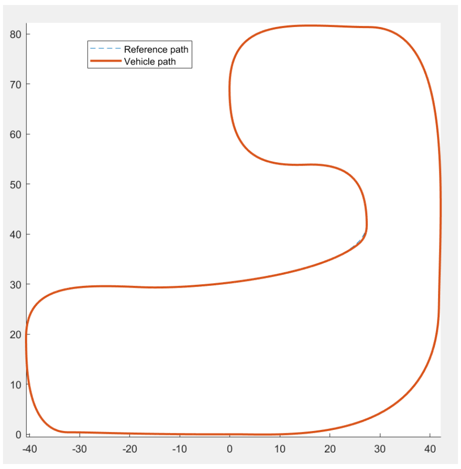


Figure 4.5.4: Cost tuned for perfect tracking with implementation of the observer