A5 - Yaw Rate Controller

Danilo Rameschandra

System Design
- Problem 1

Modeling Part
- Problem 2
- Problem 3
  - Literature
  - Mathematical definition
- Problem 4
  - Plant parameters
- Problem 5
- Problem 6
- Problem 7
  - Results

Estimator Design
- Problem 8
- Problem 9
  - **Callout**
  - Results

Implementation
- Problem 10
- Problem 11

Verification and validation
- Problem 12
- Problem 13

Torque Vectoring Model
- Plant model
- Controller Implementation
- Difficulties and Improvements

System Design

Problem 1

Mobility system: Yaw rate Controller for a vehicle

Problem formulation: Minimize yaw rate error by using a controller and estimator

Problem specification:

- rise time: (10–90%): 0.25–0.35 s
- settling time ≤ 0.8 s
- overshoot ≤ 5%

Modeling Part

Problem 2

For this project we're asked to choose at least five reference papers. Although from the five only 3 were used with more focus the goal was trying to absorb as much knowledge as possible to make a compliant model. The reference papers are the following:

- Ahmed, A.A., Barood, F.A.E. and Khalifa, M.S., 2021. *Vehicle yaw rate simulation and control based on single-track model*. *World Journal of Advanced Research and Reviews*, 10(1), pp.19–29. doi:10.30574/wjarr.2021.10.1.0127 [1]
- Barroso, D., 2020. *Torque Vectoring Control for Electric Vehicles*. Master's Thesis, Instituto Superior Técnico, Lisbon, Portugal. [2]
- Almeida, A.C., 2019. *Torque Vectoring for an Electric Vehicle Based on a Single Track Model*. Master's Thesis, Instituto Superior Técnico, Lisbon, Portugal. [3]
- Morera Torres, E., 2021. *Torque Vectoring System for a Four-Wheel Drive Electric Vehicle*. Bachelor's Thesis, Universitat Politècnica de Catalunya, Barcelona, Spain. [4]
- Stoop, A., 2014. *Design and Implementation of Torque Vectoring for the Forze Racing Car*. Master's Thesis, Delft University of Technology, Delft, Netherlands.

Problem 3

Literature

The first intention of the model was to develop a torque vectoring system, i.e a controller for torques for which the end goal is to minimize the yaw rate error between yaw rate state and a yaw rate reference. Unfortunately, due to the complexity of the model that came with it and an increasing debug, this model had to be postponed.

As a consequence, a much simpler approach to yaw stability control was made based on reference paper [1].

In this paper, the concern is to minimize the error, by directly controlling yaw rate using a delta correction instead of Torques.

Mathematical definition

Looking into the mathematical part we can split into four different block calculations:

1. Front/rear **slip angles:**

$$\beta_f = \tan^{-1}(\tan\beta + \frac{l_f}{\cos v} r)$$
$$\beta_r = \tan^{-1}(\tan\beta - \frac{l_r}{\cos v} r)$$

2. **Linear tire** lateral forces

$$F_{y,f} = C_f(\delta - \beta_f)$$
$$F_{y,f} = C_r(0 - \beta_r) = -C_r\beta_r$$

3. **Lateral/yaw dynamics**

$$\dot{\beta} = \frac{F_{y,f} + F_{y,r}}{mv} - r$$
$$\dot{r} = \frac{l_f F_{y,f} - l_r F_{y,r}}{I_z} - r$$

4. **Kinematics/trajectory (the non-linearity):**

$$\dot{\psi} = r,$$
$$\dot{Y} = \int_0^t v \sin(\beta + \psi) = v \sin(\beta + \psi)$$
$$\dot{X} = \int_0^t v \cos(\beta + \psi) = v \cos(\beta + \psi)$$

Having at least 4 derivatives or _dot signals, we can define our 4 states and onecontrol input as:

$$x = \begin{bmatrix} \beta & r & \psi & Y \end{bmatrix}$$
$$u = \begin{bmatrix} \delta \end{bmatrix}$$

Problem 4

For this part we need to implement the non linear model in A5 skeleton inside “plant model”

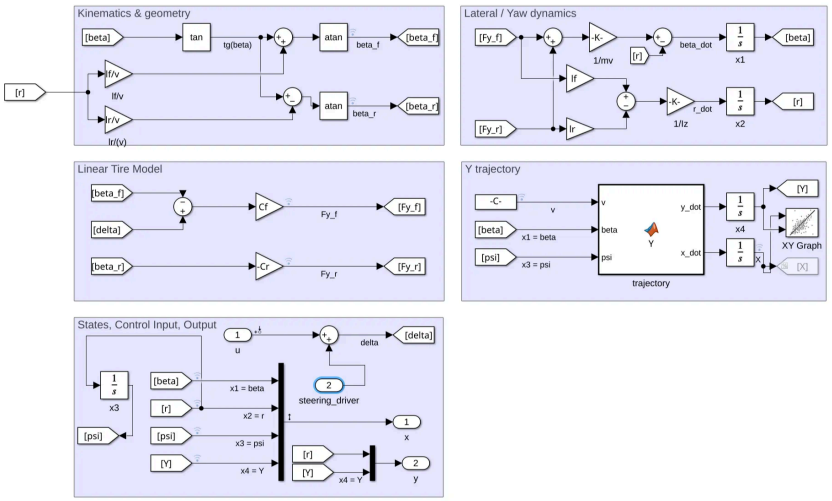

Figure 1 - Plant model

A sine wave was used as input to test it out how the plant behaves:

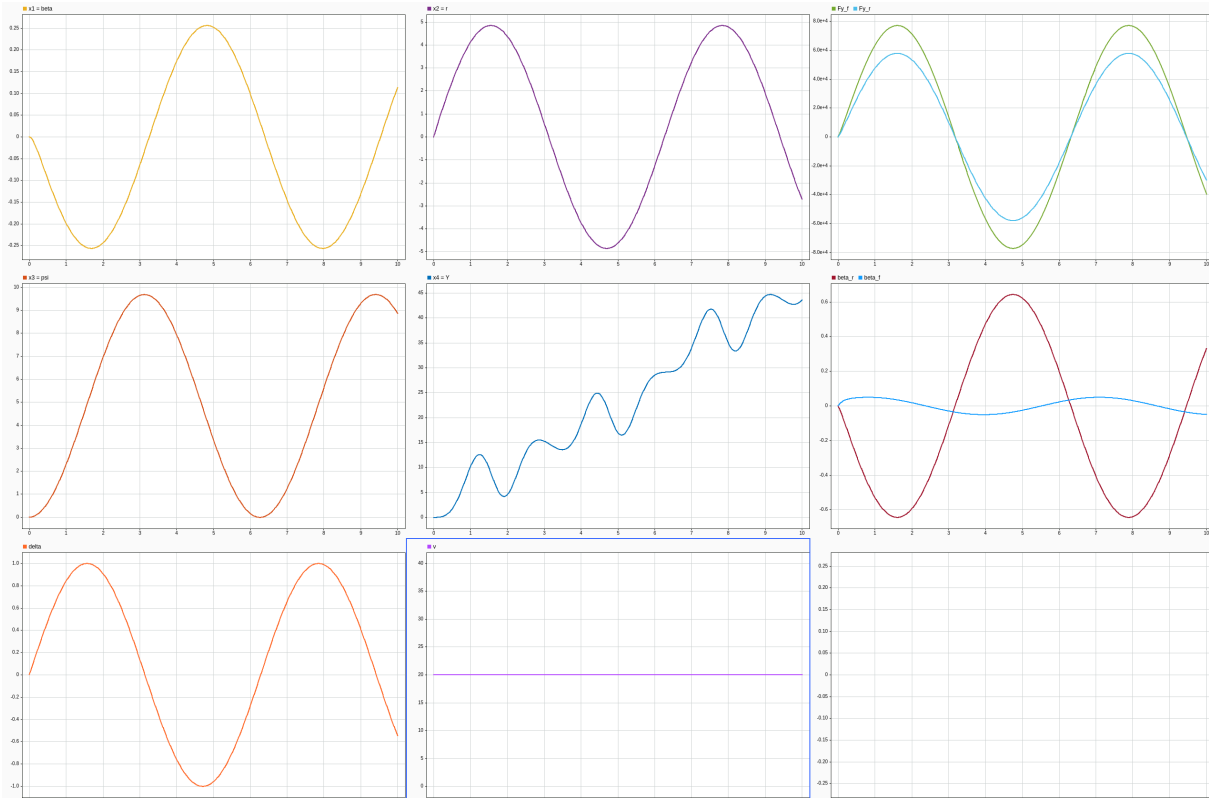

Figure 2 - Plant response to a sine wave

Plant parameters

Parameters	Value	Units
m	1000	kg
Iz	1700	kg m²
lf = lr	1.2	m
v	20	m/s
Cf= Cr = 9e04	9e04	N/rad

Problem 5

For this problem we needed to linearize the previously constructed model in p3. To do so, we can both use simulink's model linearizer app and similarly compare with the by hand method of applying partial derivatives that follows:

$$A = \left.\frac{\partial f_i(x,u)}{\partial x_i}\right|_{(x_0,u_0)}$$
$$B = \left.\frac{\partial f_i(x,u)}{\partial u}\right|_{(x_0,u_0)}$$

So considering the equilibrium point:

$$\delta = 0\ , \beta = 0\ , r = 0\ , \psi = 0\ Y = 0,\ v = v_0 = 0$$

We can apply the *Jacobian* command in matlab as seen in the following code:

```
% Jacobians to find A, B
A_jac = jacobian(f,x); % df/dx
B_jac = jacobian(f,u); % df/du

% A, B at equilibrium point
A_eq = simplify(subs(A_jac,{beta,r,psi,Y,delta},{0,0,0,0,0}));
B_eq = simplify(subs(B_jac,{beta,r,psi,Y,delta},{0,0,0,0,0}));
```

From where we got the following state space matrices:

$$A = \begin{bmatrix} -\frac{C_f+C_r}{mv} & \frac{-L_fC_f+l_rC_r}{mv^2}-1 & 0 & 0 \\ \frac{-l_fC_f+l_rC_r}{I_z} & -\frac{l_f^2C_f+l_r^2C_r}{I_zv} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ v & 0 & v & 0 \end{bmatrix} \quad B = \begin{bmatrix} \frac{C_f}{mv} \\ \frac{l_fC_f}{I_z} \\ 0 \\ 0 \end{bmatrix}$$

Similarly, we also used model linearizer app. To do so, we needed to set an input perturbation (delta = u) and output perturbation (states). The result was the same we found using Jacobian.

Problem 6

To start off the controller design we need to check if the system is reachable. In order to do that we can check if the rank of the reachability matrix is equal to the number of states, in this case 4.

```
disp(rank(ctrb(A,B))==4);% output=true, system is reachable
```

Using the *ctrb* and *rank* command the output is 4, so the system is reachable

Problem 7

Since our system is reachable we can design a state feedback controller following the form:

$$\Delta u = -K\Delta x + k_r\Delta r \quad [eq1]$$

To choose the K controller gain matrix LQR was the chosen method. To give some context, by using a LQR we want to minimize the following cost function:

$$\min \int_0^\infty (x(t)^T Q x(t) + u(t)^T R u(t))\, dt$$

As seen, there are two input gain terms Q, R from which we can choose by implementing the Byrson's rule, i.e:

$$Q = diag(\frac{1}{x_i^2}) \quad R = diag(\frac{1}{u_i^2})$$

So by applying *lqr* command in matlab for A, B, Q, R we could get K:

```
% Simulink output + Bryson's rule
q1 = 1/max(xs.signals.values(:,1))^2;
q2 = 1/max(xs.signals.values(:,2))^2;
```

```
q3 = 1/max(xs.signals.values(:,3))^2;
q4 = 1/max(xs.signals.values(:,4))^2;

% Fallback for Q and R
q1 = 1; q2 = 1; q3 = 1; q4 = 1;
u1 = 1/max(uinp.signals.values(:,1))^2;

Q = diag([q1 q2 q3 q4]);
R = 1/deg2rad(2)^2; % fallback for Q (1 / 0.04)
R = u1;

% LQR
K = lqr(A,B,Q,R);
K2 = [K(1), 0, K(3), K(4)];

% Check eigenvalues / stability
disp(eig(A-B*K)); % full K
disp(eig(A-B*K2)) % without r
```

$$K = \begin{bmatrix} 1.9273 & 0.9589 & 7.0031 & 0.6593 \end{bmatrix}$$

Also, since we can't run the simulation when we don't have data from the previous simulation, i.e while opening matlab/simulink, a fallback K matrix is needed. For this fallback gain, the identity matrix was choose, so:

$$Q = diag(I) \quad R = \frac{1}{0.04}$$

$$K_{fb} = \begin{bmatrix} 0.0757 & 0.0506 & 0.5207 & 0.0349 \end{bmatrix}$$

Besides K matrix, we also need to calculate k_r. For that we can follow the equation below:

$$k_r = \frac{-1}{C(A - BK)^{-1}B}$$

Since we try to control our system, it also makes sense to have a reference for which the controller knows what to achieve. For it we used 3 reference options:

1. Lane change maneuver which a sine wave with T =0.7 s put here and $f = 8.9760\ hz$
2. Simple step function
3. Desired yaw rate formulation given by

$$r_{desired} = \frac{v_x}{L + K_u v_x^2}\delta$$

where L is the wheelbase and Ku is the understeer gradient, which in this case we set as 0 for simplification, i.e neutral steer (no under-oversteer). To use the desired value we first have to input a δ which can be sine wave, step or other signal.

Regarding the actual controller used, two similar, but, still different methods were done.

The first one follows the state feedback form presented above [eq.1]. Where the controller is acting on every state we have, i.e $x = \begin{bmatrix} \beta & r & \psi & Y \end{bmatrix}$ and output $u = \begin{bmatrix} \delta \end{bmatrix}$

Due to unexpected behavior of the states and control input, an alternative method to use the controller was used. Since the goal is to minimize the yaw rate error, an "error state" was created an inputted in a PI controller, while the other 3 states $x' = \begin{bmatrix} \beta & \psi & Y \end{bmatrix}$ were controller by $K_2 = \begin{bmatrix} k_1 & 0 & k_2 & k_3 \end{bmatrix}$

The first controller can be seen as yellow blocks, while the second as orange:

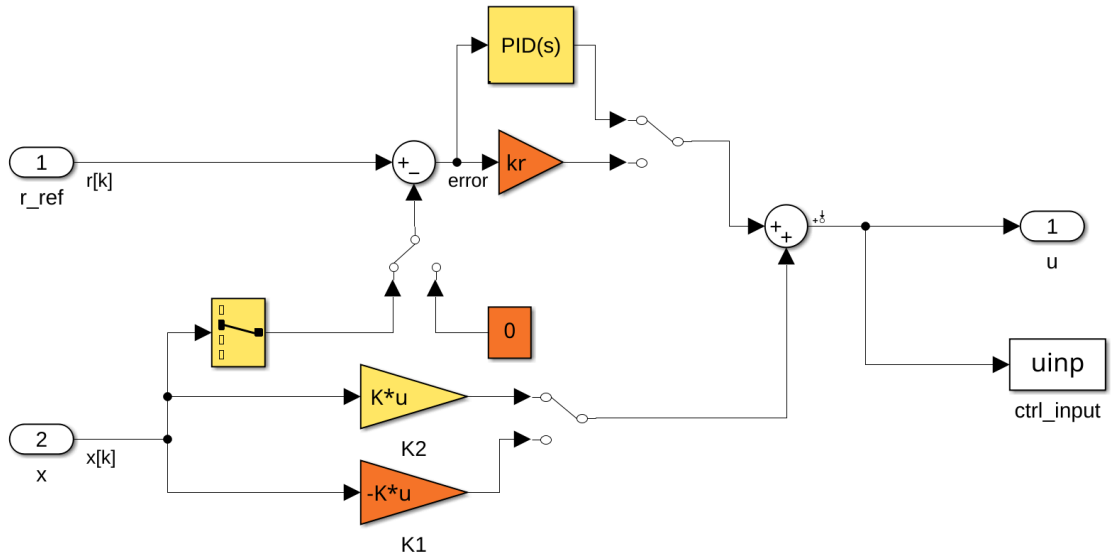

Figure 3 - Controller yellow (error state) and Controller orange

Results

In this subsection a lane change test was conducted, using the following inputs for the model and the signal we get the following responses:

Plant Model parameters

mass (kg)	1000
Inertia (kg m^2)	1700
Wheelbase (m)	2.40
Cornering. stiff front/rear (N/rad)	9e04

Input / Controller

Amplitude	0.05
Period T	0.7
t0	1.0
Frequency	2*pi/T = 8.9760
P Gain attempt 1	5
P Gain attempt 2	30

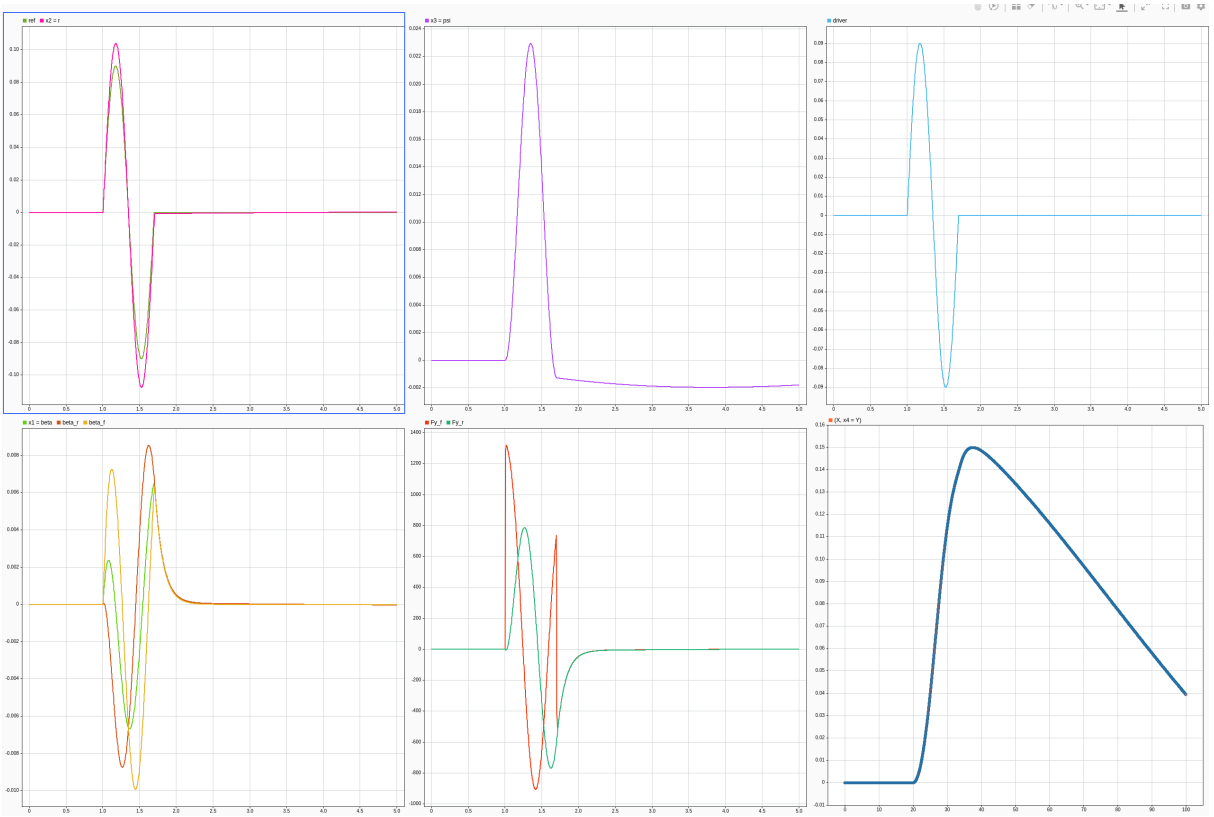

Figure 4 - Response using P gain = 5 and Q = R = diag([I])

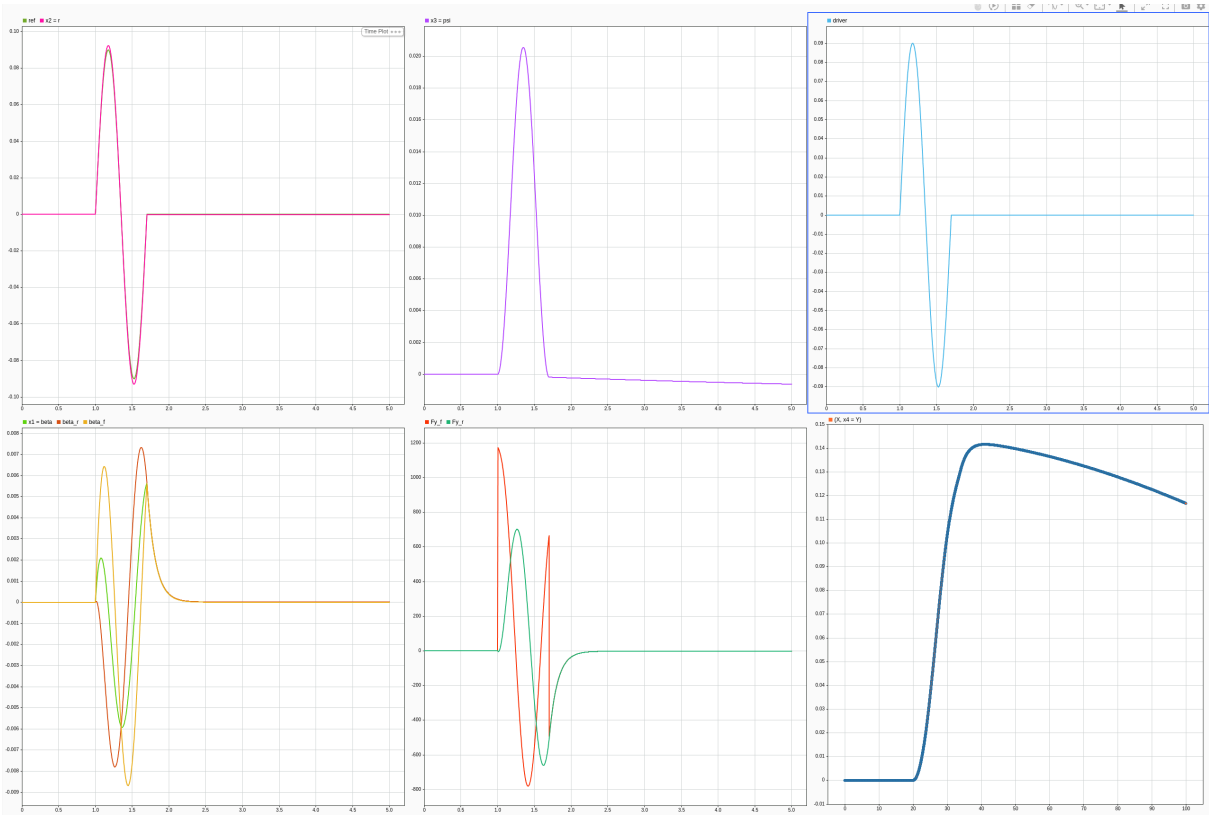

Figure 5 - Response using P gain = 30 and Q = R = diag([I])

From the results, by keeping the same $K_{2,fb}$ matrix, and tuning the P_{gain} it's clear that the yaw rate is following more closely the reference, therefore the error is decreasing.

Now using the K_2 with the the Bryon's method:

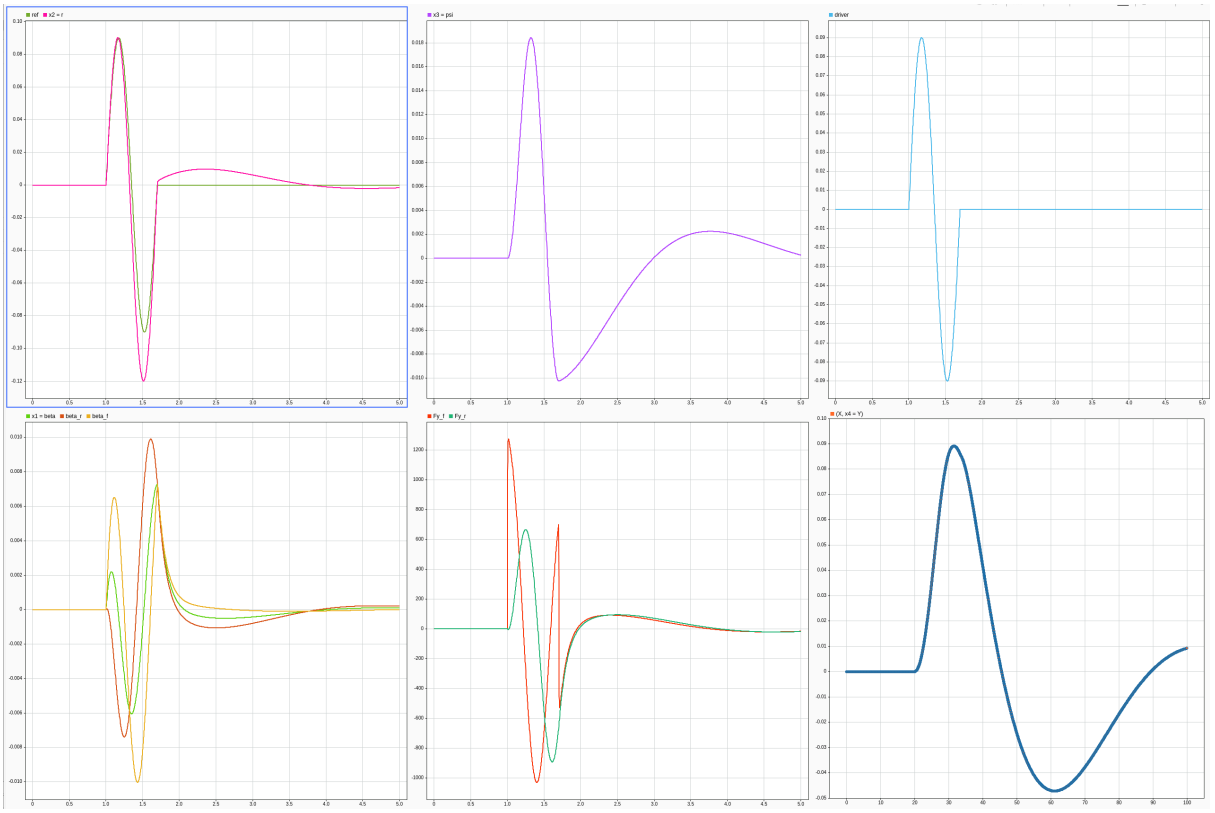

Figure 6 - Response using P gain = 5 and Bryson’s rule

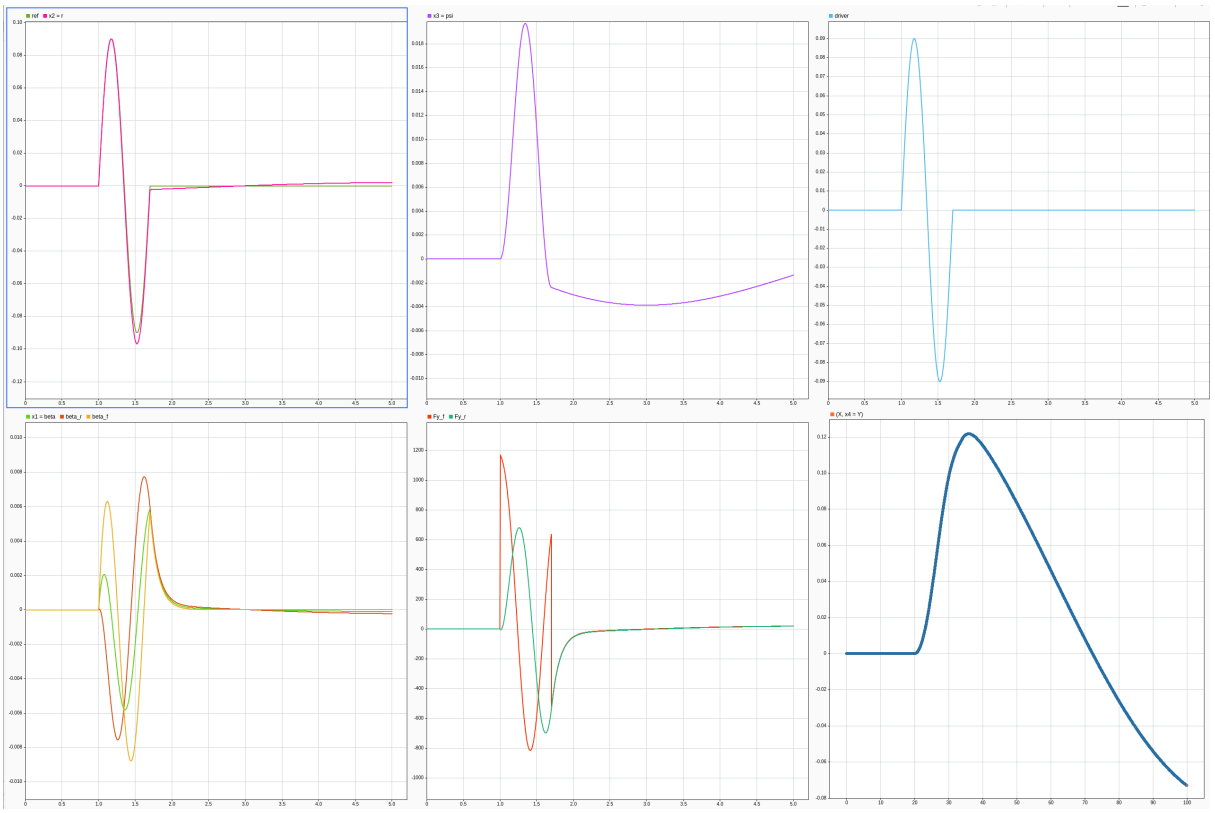

Figure 6 - Response using P gain = 30 and Bryson’s rule

Although there is an error and a slight overshoot in the case where the P gain = 5, by increasing the B gain by 4 times, the error, decrease. To emphasize this difference look how less “abrupt” is the yaw angle change (x3 = psi)

Since the goal for the controller was to implement it in $u = -K\Delta x + k_r \Delta r$ and since $k_r \to \inf$ when using the *lqr,* pole placement method was also adopted.

$$p(s) = (s + p_1)^2 (s + p_2)^2$$

For this we want to have a double poles on p_1 and desire a p_2 that is twice as fast as p_1.

By defining $p = \begin{bmatrix} -5 & -5 & -10 & -10 \end{bmatrix}$ and a $C = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$ to measure Y instead of previous r and changing the reference to a simple step with 5 s and 3.5 as final value, we got the following results:

- $k_r = 0.2186$

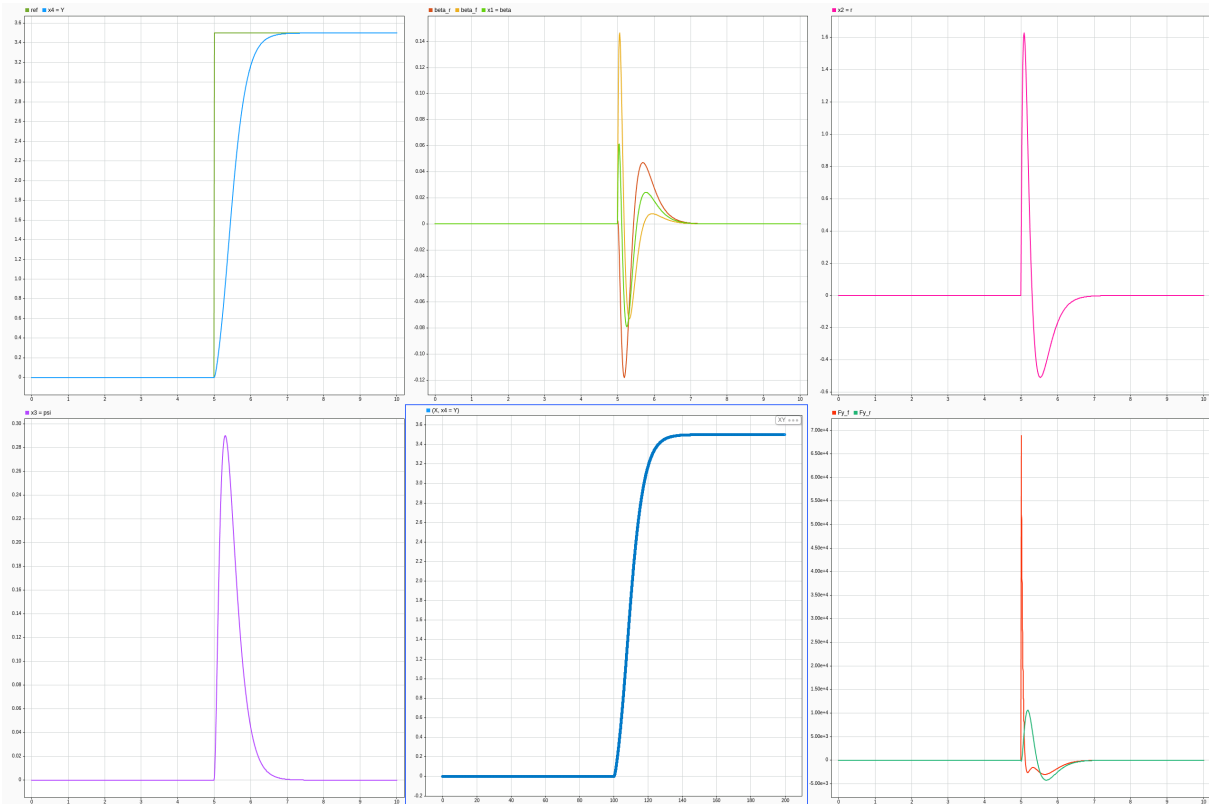

Figure 7 - Step response using pole placement

Estimator Design

Problem 8

Measuring all states might be costly and troublesome, so an observer will be developed instead. To do so, we need to verify which output signals are observerable. Using matlab command *obsv* for A and C matrices we get:

```
% C matrix candidates
C_r    = [0 1 0 0]; % measure yaw rate
C_psi  = [0 0 1 0]; % measure psi
C_Y    = [0 0 0 1]; % measure Y
C_rY   = [C_r; C_Y]; % measure r and Y
```

```
rank(obsv(A,C_r))) % output: 1
rank(obsv(A,C_psi))) % output: 2
rank(obsv(A,C_Y))) % output: 4 (observable)
rank(obsv(A,C_rY))) % output: 4 (observable)
```

Since the rank of the observability matrix C_Y, C_{rY} is the same as the number of states (4), we can use both of them or only the last state for our estimator. As an example, by using a IMU for yaw measure, and a GPS for position.

$$C_Y = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C_{r,Y} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Problem 9

Since the system is observable for $C_Y, C_{r,Y}$ we can start our estimator design by using the standard closed loop estimator formulation given by:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x})$$

To find L we are going to rely on Kalman filtering. By using Kalman filtering we will try to find a L that minimizes the error dynamics (error between true states and estimate states)

L can be calculated using the *icare* command in matlab. To use this function we need to first define what are our Q_{kf}, R_{kf} which for an initial run we will use the identity matrix.

```
C = [0 1 0 0; 0 0 0 1]; %{c1}
%C = [0 0 0 1]; %{c2}

% kalman weight gains
Qkf = diag([1, 1, 1, 1]);
Rkf = diag([1,1]);
%Rkf = 1; %{c2}

[RicattiSol, L, eigenvalues] = icare(A',C',Qkf, Rkf);
L = L' ;
eig(A - L*C) % The eigenvalues had negative real part so obs is stable!
```

The estimator gain computed was:

$$L_{kalman_{identity}} = \begin{bmatrix} -0.0039 & 0.0607 \\ 0.0653 & 0.0054 \\ 0.0078 & 1.0077 \\ 0.0054 & 6.6135 \end{bmatrix}$$

$$\lambda_{kf,identity} = \begin{bmatrix} -3.4395 + 2.9900i \\ -3.4395 - 2.9900i \\ -7.7243 + 0.0000i \\ -8.6991 + 0.0000i \end{bmatrix}$$

Additionally, in the results section below we will tune the weighting matrices so decrease the error.

Finally, since L, A, B, C are known we can build the closed loop estimator like is shown below:

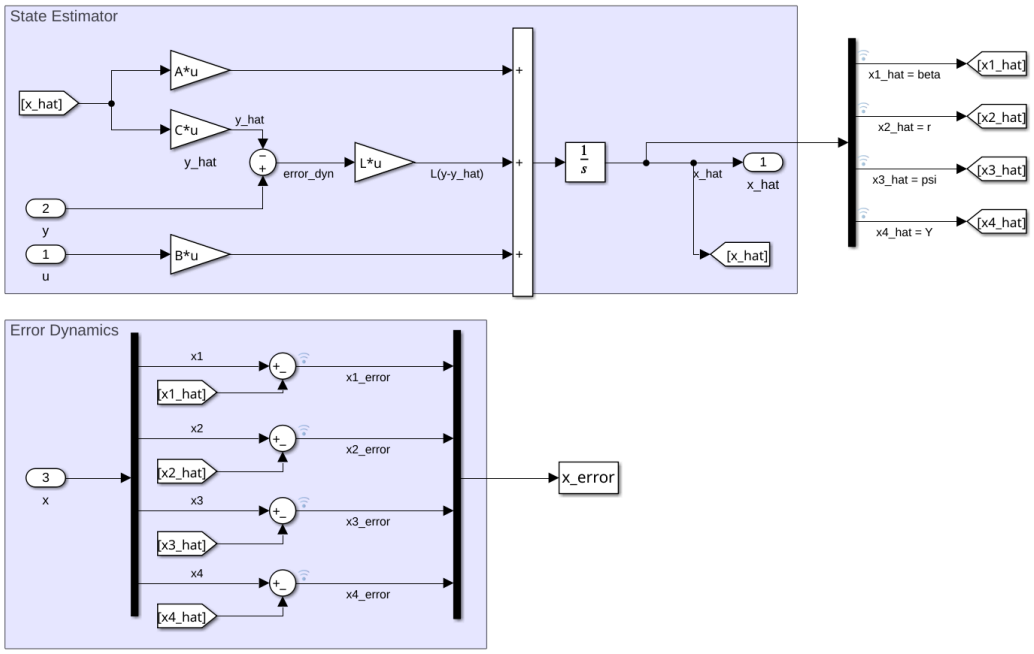

Figure 8 - State estimator subsystem

Callout

Bigger Qkf

- “I don’t trust my model” → The filter will follow the **measurements** more

Bigger Rkf

- “I don’t trust sensors” or “Sensors are noisy” → The filter **relies more on the model**

Results

Using the weighting gains as the identity matrix, the following plots show the system response:

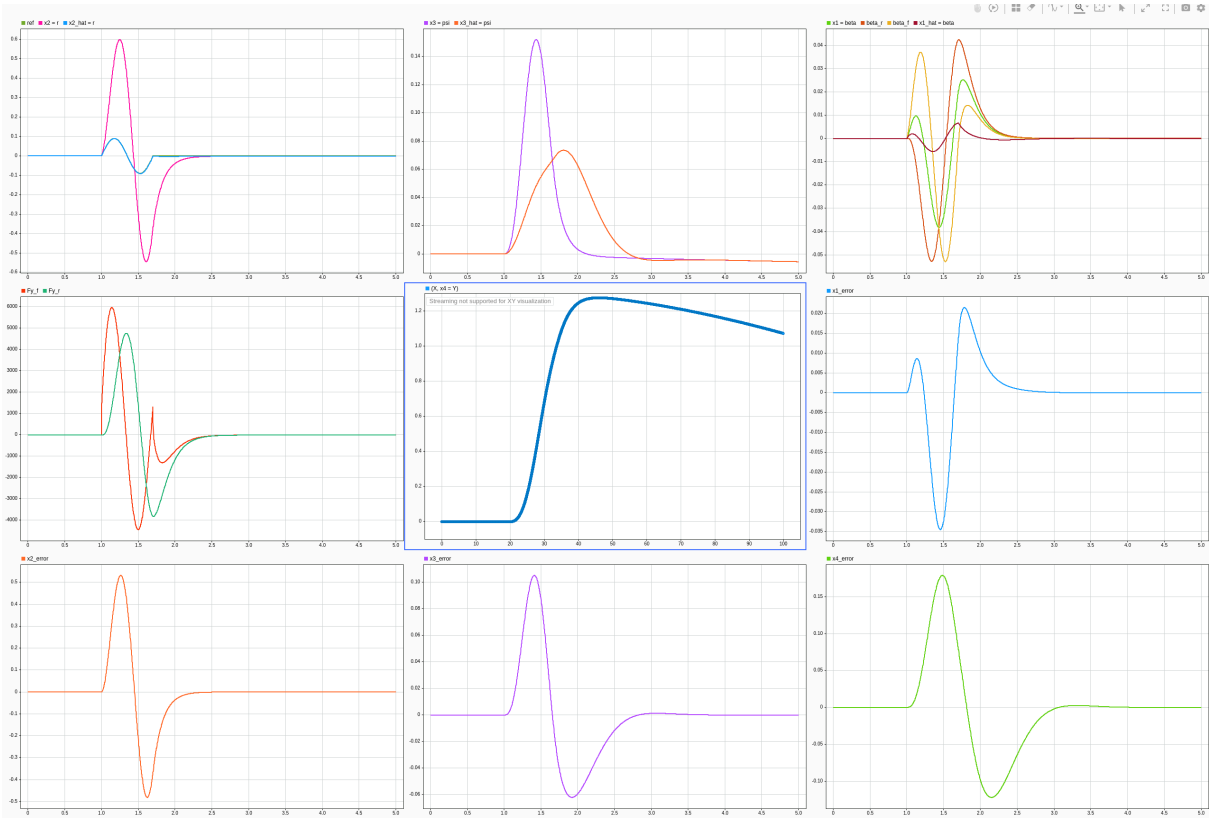

Figure 9 - States and States_hat for Qkf and Rkf set as identity matrices with different sizes

It can be seen, specially by the error plots, that the error dynamics is significant, specially for the last three states, being the highest error 0.16.

Since a lower error is desired, for the next simulation, the weighting matrices Q_{kf}, R_{kf} will be changed according to the Call-out above.

Considering that, and after a lot of trial it was seen that by increasing the Q_{kf} and decreasing R_{kf} the error will decrease, meaning we should trust more the measurements instead of rely on the model.

```
% kalman weight gains
Qkf = diag([1e4, 5e4, 1e5, 1e4]); % Modl noise on [beta,r,psi,Y] - aumentar diminui o err
Rkf = diag([0.02^2, 0.10^2]); % meas noise: yaw-rate,Y - aumentar aumenta o err {c1}
```

Finally we got the estimator $L_{kf_{tuned}}$, $\lambda_{kf_{tuned}}$

$$L_{kf_{tuned}} = 10^4 \times \begin{bmatrix} -0.0001 & 0.0135 \\ 1.1173 & 0.0000 \\ 0.0001 & 0.3162 \\ 0.0000 & 0.1064 \end{bmatrix}$$

$$\lambda_{kf_{tuned}} = 10^4 \times \begin{bmatrix} -0.0009 \\ -0.0067 \\ -0.0998 \\ -1.1180 \end{bmatrix}$$

The results are the following:

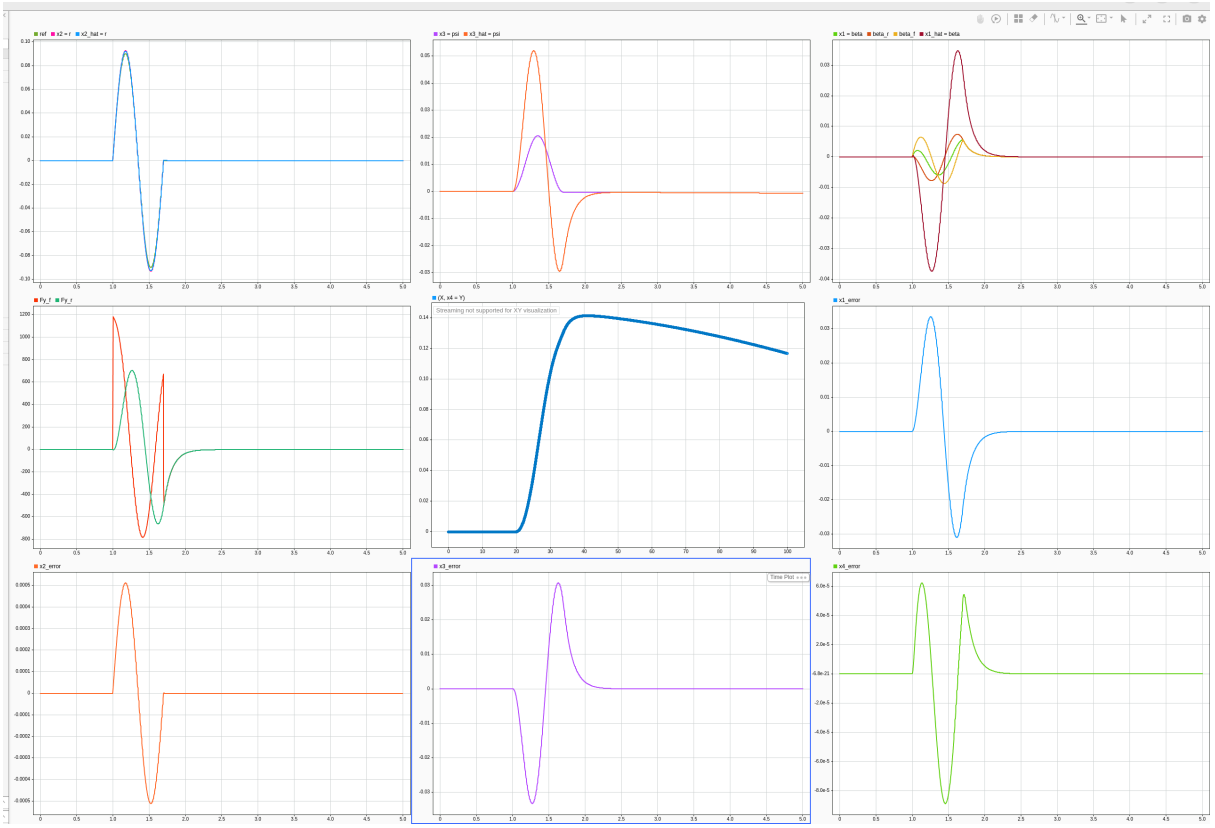

Figure 10 - States and States_hat for Qkf and Rkf tuned matrices

Although the the Q_{fk} gains are exaggerated, this approach helped understand that by not trusting enough on the model, the error dynamics will decrease.

Implementation

$$u[k] = u_e[k] - K(x[k] - x_e[k]) + k_r(r[k] - r_e[k])$$

Problem 10

Until now we have been using a continuous time plant, however since the controller and estimator are implemented inside of outboard of a computer we need to time-discretize the system.

For doing this different approaches were made. First knowing hour continuous-time state space model, we can use *c2d* command to find time-discrete state spaces matrices:

```
C = C_Y; % [0 0 0 1]
D = 0;
h = 0.01; % sample time

% Define continuous time ss
%sysC = ss((A-L*C), B, C, D);
sysC = ss(A, B, C, D);
% Discretize the contininous time system
sysD = c2d(sysC,h,'tustin');
```

Having the discrete state-space matrices we also need to find L.

To do so, we can use kalman filtering but instead of using A, C as previously, we can *sysD*, A and C matrices:

```
Qkf = diag([1 1 1 1]);
Rkf = 1;
[RicattiSol, L, eigenvalues] = icare(Ad',Cd',Qkf, Rkf);
L = L' ;% to work must be the transpose of the icare fcn output
```

Once the discrete estimator is design, we can start to implement it. For that the first approach was to follow figure 8 block, however the error between the true and estimate states was too high and hard to debug, so this method ended up being left out.

Similarly, it is comment it out the other attempt, for which, instead of sysD.A, we used the A-L*C as our A, but turn out to be too complicated, and not worth investing more time into it.

Finally the method we ended up using was the Euler method:

$$\dot{x} = (x[k+1] - x[k])/k$$

Substituing this into the closed loop estimator equation we get:

$$\hat{x}[k+1] = \hat{x}[k] + h\big(A\hat{x}[k] + Bu[k] + L\big(y[k] - C\hat{x}[k]\big)\big)$$
$$\hat{x}[k+1] = (I + h(A - LC))\hat{x}[k] + hBu[k] + hLy[k]$$

Looking into it, we can now define:

- $A_d = I + h(A - LC)$
- $B_d = hB$
- $L_d = hL$

These matrices are then used to implement the estimator in Simulink, ensuring that the discrete-time behavior closely approximates the continuous-time dynamics for sufficiently small time steps h.

Problem 11

Considering what was said in the previous problem, we ended up with the final estimator:

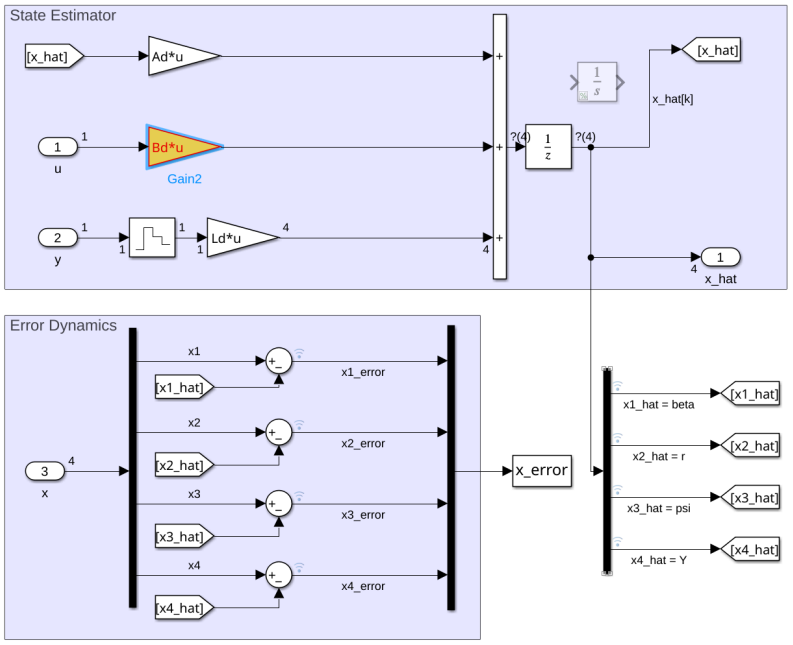

Figure 11 - State estimator using Euler method

Defining the sampling time, showed to be very important parameter to either decrease or increase the error dynamics. For example figure 10 shows the response with h = 0.0020 s, which clearly shows very inconsistent and weird results

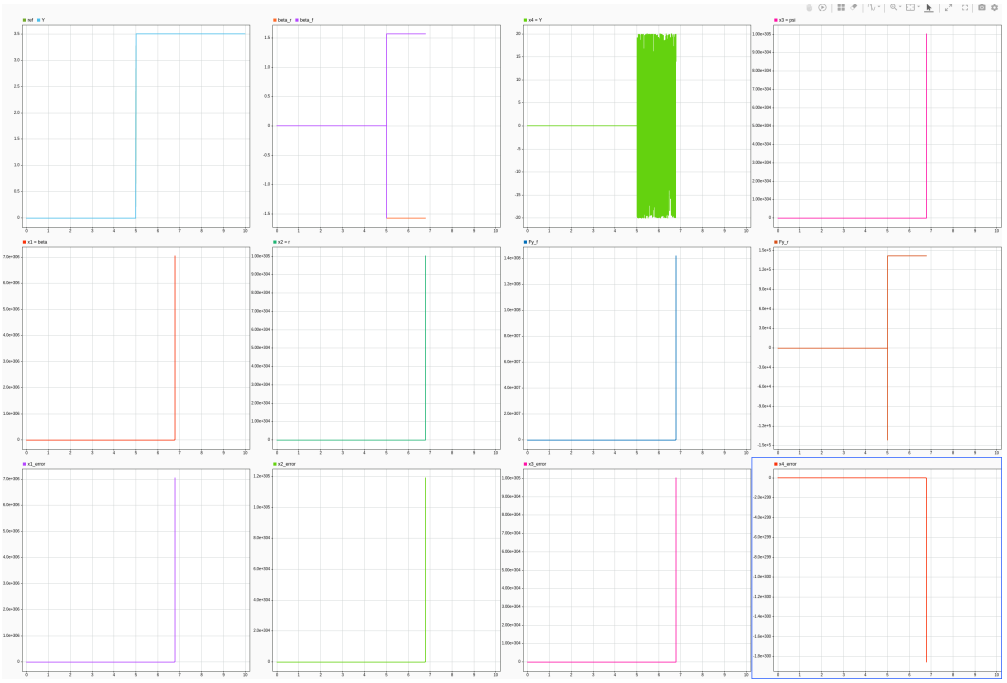

Figure 12 - Results using Euler method, and h = 0.0020 s

By defining the h = 0.1 s, we got the results below. Although not perfect, it had a error and also response improvement compared to the previously one.

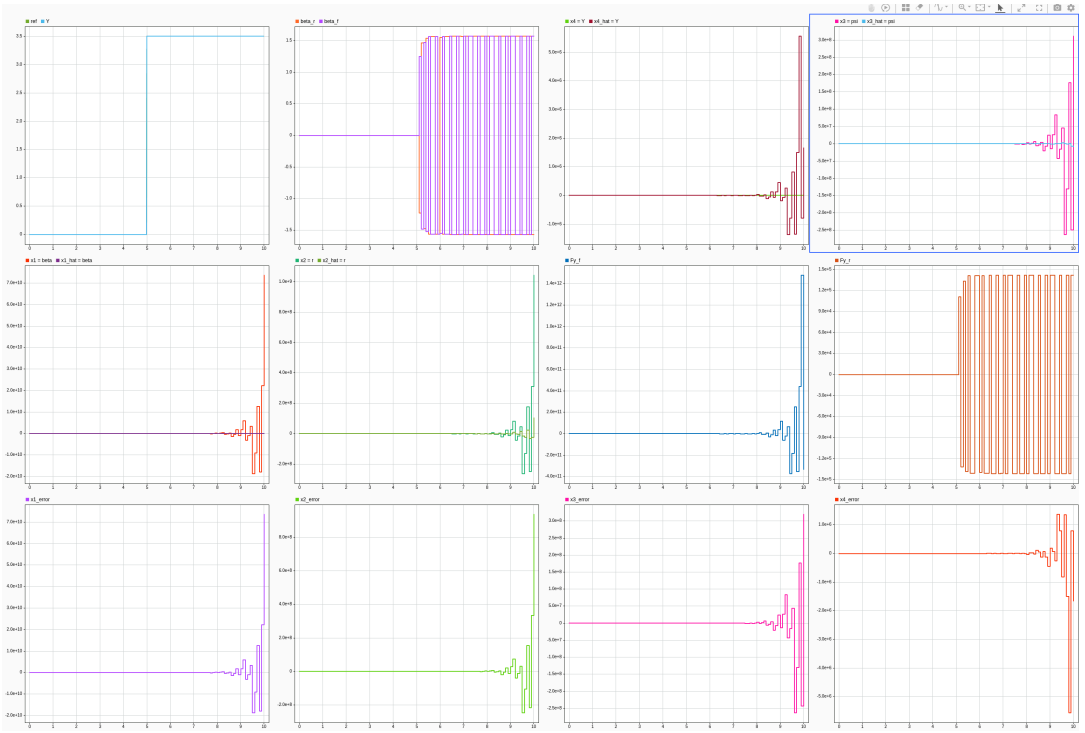

Figure 13 - Results using Euler method, and h = 0.1 s

However this ended up being counter-intuitive since a higher sampling time would aproximate the state to the estimated states, meaning the error would decrease, which is not what's happening for this case.

Verification and validation

Problem 12

Since the discrete-time wasn't properly working, this part was not achieved or completed.

Problem 13

Throughout the whole assignment like it was explain, different attempts were made to improve the model response, with and without any type of controller.

Due to the non-linearity of the plant, specially the tire model, a lot of tweaking for the cornering stiffness was done as well the mass and inertial, meaning that the final parameters values found to be the optimal ones, i.e the ones that allowed the state / or output follow the reference.

Torque Vectoring Model

⚠️ I will not try to cover / answer every question, instead just a brief explanation of the process

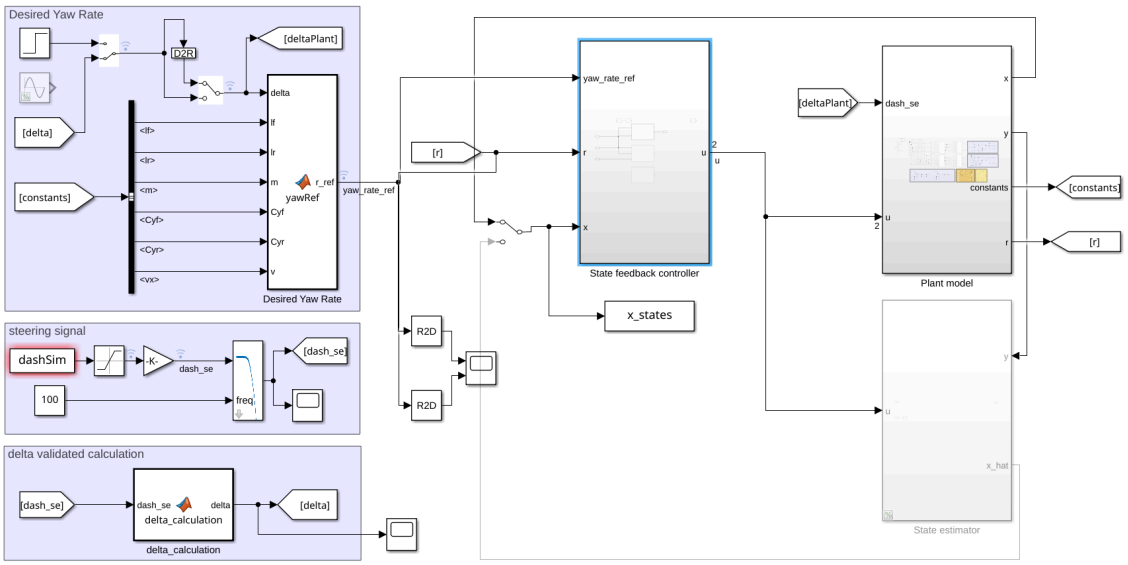

Figure 14 - Simulink of TV system

Plant model

For this model, which was supposed to be the one to deliver at first, the purpose was control the yaw rate of the vehicle using as reference, the desired yaw rate reference. To help to control the yaw, a certain M_z is created via torques of the rear wheels.

To build the model firstly we need to derive the equations using a bicycle model and a unicycle wheel model like it was done in reference paper [2].

$$\dot{v}_y = \frac{F_{y,f}\cos\delta + F_{y,r}}{m} - v_x r,$$
$$\dot{r} = \frac{l_f F_{y,f}\cos\delta - l_r F_{y,r}}{I_{zz}} + \frac{(F_{x,rr} - F_{x,rl})t_r}{2I_{zz}},$$
$$J_w\dot{\omega}_{rl} = T_{rl} - R_w F_{x,rl}, \quad J_w\dot{\omega}_{rr} = T_{rr} - R_w F_{x,rr}.$$

Using the equations above, the following states and output signals were defined:

$$x = \begin{bmatrix} v_y \\ r \\ w_{rl} \\ w_{rr} \end{bmatrix} \quad u = \begin{bmatrix} \delta \\ Tq_{rl} \\ Tq_{rr} \end{bmatrix}$$

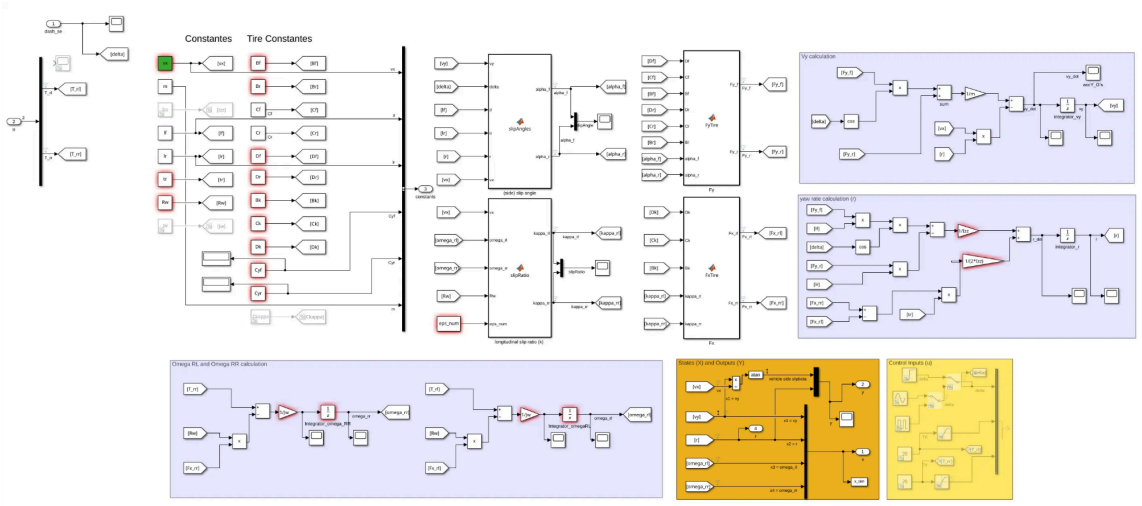

Figure 15 - Plant model

To continue with the model we also needed to linearize it, similar to what what was done in the delivered version, so, the jacobians were calculated, and the model was linerized around the following equilibrium points:

$$v_y = 0\ , r = 0\ , \delta = 0\ , \omega_{rl} = \omega_{rr} = \frac{v_x}{R_w},\ T_{rl} = T_{rr} = 0$$

```
% Vehicle parameters
m=230; Izz=12; lf=0.80; lr=0.80; tr=1.20;
vx=10; Rw=0.228; Jw=0.24;

% Tire parameters
Bf=8;  Cf=1.5; Df=1800;
Br=9;  Cr=2.0; Dr=1500;
Bk=7;  Ck=1.2; Dk=500;

Cyf = 21429; % 800 N/deg ou 21429 N/rad (N/rad worked)
Cyr = 15714; % 600 N/deg ou 15714 N/rad
```

```
% Calculations
% Slip angles
alpha_f = atan((vy + lf*r)/vx) - delta;
alpha_r = atan((vy - lr*r)/vx);

% Slip ratio
kappa_rl = (Rw*omega_rl - vx)/(vx + eps_num);% sum eps to avoid 0
kappa_rr = (Rw*omega_rr - vx)/(vx + eps_num);

% Tire forces Magic Formula
Fy_f = -2*Df * sin(Cf*atan(Bf*alpha_f));
Fy_r = -2*Dr * sin(Cr*atan(Br*alpha_r));
```

```
Fx_rl = Dk * sin(Ck*atan(Bk*kappa_rl));
Fx_rr = Dk * sin(Ck*atan(Bk*kappa_rr));
```

Controller Implementation

Once the model plant was linerized and confirmed with the model linearizer app, it was time to design the controller. To start it was confirmed that the system was reachable, and therefore a LQR was used to find K matrix. As for the k_r we use the same formula as the one used in the delivered paper.

The figure below show the controller implementation:

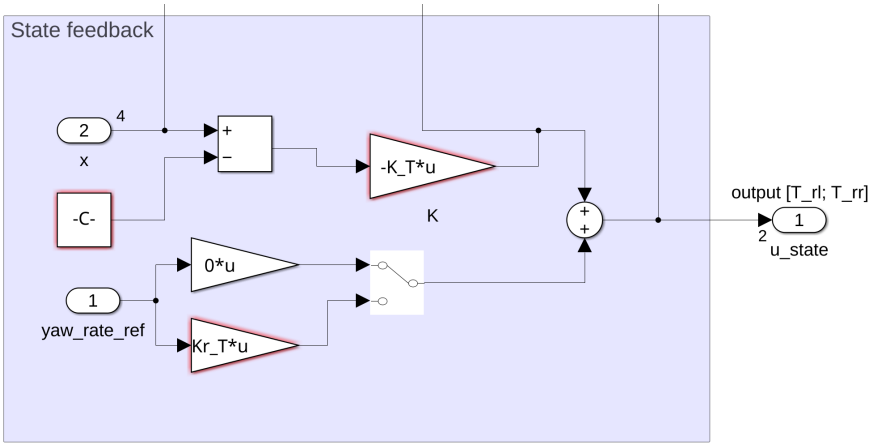

Figure 16 - Controller

As we can see the controller has C which is the vector with the equilibrium point used for subtracting the states, i.e to find Δx . Regarding the K_T computed as can be seen in the code below, is the output of the lqr command for which the B input of the command correspond to the 2nd and 3rd columns of matrix B. By doing this, we ensure that the delta used inside of the plant model is the one of the driver after being filter or computed with the desired yaw rate formulation, by using this, the delta at the wheel is ensured.

For k_r on the other hand similarly to what happened on the delivered version, it was giving very volatile values which would cost the Torque output. The output is then used inside of the plant.

```
Ctrack = [0 1 0 0];
B_T = B(:,2:3); % for torques
%B_T = B;

% % Gain max
% vy = x_states.signals.values(:,1);
% r = x_states.signals.values(:,2);
% omega_rl = x_states.signals.values(:,3);
% omega_rr = x_states.signals.values(:,4);

% LQR
Q = diag([1, 1, 1, 1]);   % [vy, r, ω_rl, ω_rr]
% Q = diag([1 1 1 1]);
R = diag([1 1]);        % [T_rl, T_rr]
K_T = lqr(A, B_T, Q, R);  % 2×4 gain torques
```

```
disp(eig(A-B_T*K_T)')

% Kr
M   = Ctrack * ((A - B_T*K_T)\ B_T);
Kr_T = -(M')/(M*M');

% With I gain
%-14.5873  -21.4266  -88.4206  -88.4206

% With bryson rule
%-14.5947  -21.4216  -88.3224  -88.3224
```

Difficulties and Improvements

Due to the complexity of the model and it's non-linearities, some results achieved were not realist enough to continue pursuing this model + controller. Some examples are of the inconsistency of the model are the really small torques given at the wheels when clearly there was a yaw error present .

If we look more in depth into the model it's clear that everything is correlated, therefore, a reason why a certain torque value is really low, must be because a certain value is either very high /or low depending if their proportionality is inverse or not.

For example in the image below we have the yaw rate calculation inside of the plant model.

r is influenced by both lateral and longitudinal forces (they're after multiply by the track or half-wheelbase to give a certain moment), these forces are then dependent on slip ratios k and slip angles α . For the slip angles, they are computed using a certain delta, which is a signal, meaning it can, or not, have noise.

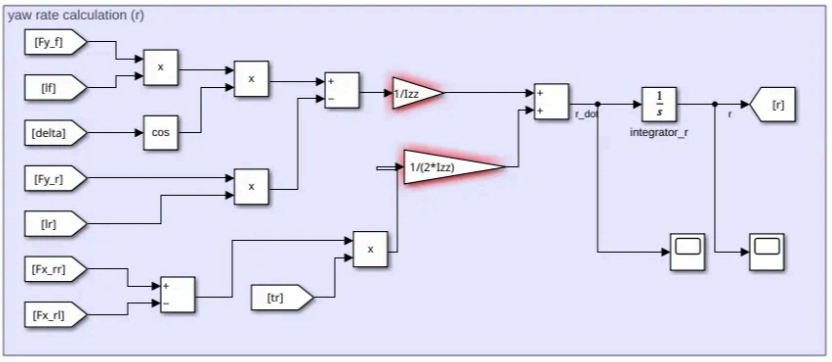

Figure 17 - Yaw rate calculation

Same for the omega calculation, they're also dependent on the slip ratio:

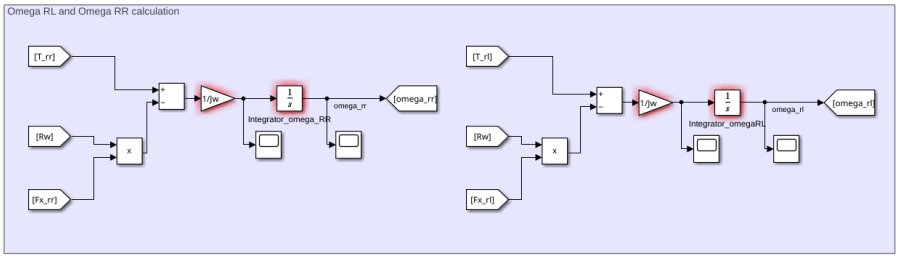

Figure 18 - Omega calculation

With this said, although the final goal wasn't achieved, the challenge and path until here was a great learning experience and I can't wait in the future to look back and hopefully find the "solution" for this problem.

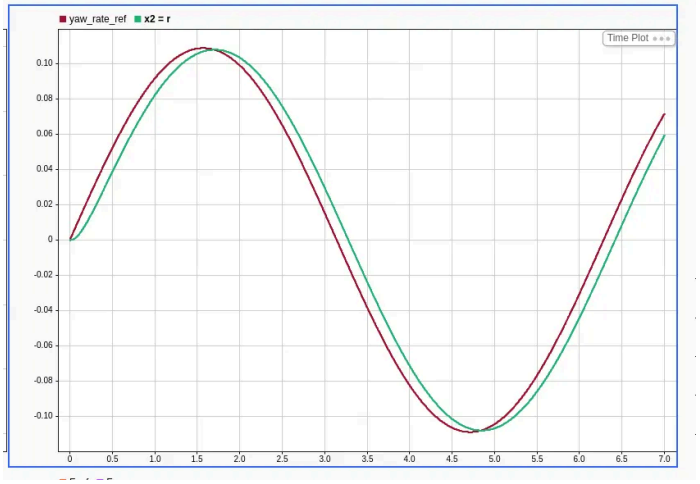

Figure 19 - Yaw rate vs Yaw rate reference