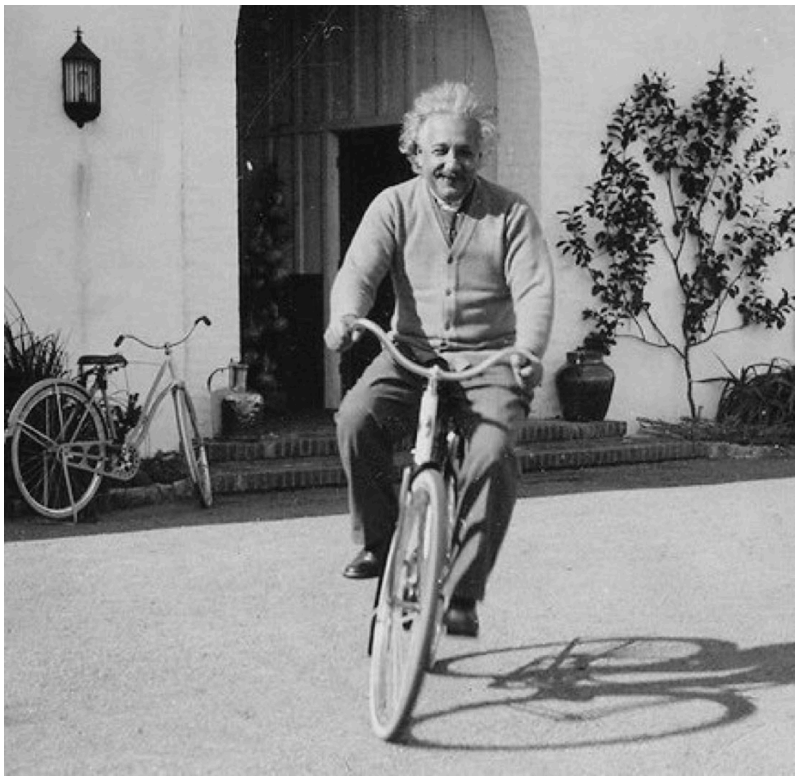


Bicycle Dynamics

Modern Control Course Project Report

Professor: Dr. Afzalian

Mohammad Hossein Soltani 98242080

Table of Contents

section	section name
1	Description
2	Parameters and Models
2.1	Parameters
2.2	Models
2.2.1	Model#0: The base model
2.2.1.1	Model#0 Stability Analysis
2.2.2	Model#1: The model with FeedBack
2.2.2.1	Model#1 Stability Analysis
2.2.3	Model#2: Whipple Model
2.2.3.1	Model#2 Stability Analysis
3	State Space Analysis
3.1	State Space Realization - The right one
3.2	Stability Analysis in State Space
3.2.1	Stable Model
3.2.2	Unstable Model
3.3	Controllability and Observability Analysis
3.4	System Implementation in Simulink
3.5	State Feedback Control Using Pole Placement Method
3.5.1	Designing the Controller

3.5.2 Implementation in Simulink

3.6 Observer for State Estimation

3.6.1 Designing the Observer

3.6.2 Implementation in Simulink

3.7 State Feedback Control Using Pole Placement Method + Luenberger Observer

3.7.1 3.7.1 - Designing the State Feedback Control + Luenberger Observer

3.7.2 Implementation in Simulink

4 Xtra - LQR Controller

4.1 Description

4.2 Designing the LQR Controller

4.3 Implementation in Simulink

5 Files

6 Conclusion

7 References

1 - Description

A bicycle is **a human-polred vehicle** consisting of two wheels connected by a frame. It typically features pedals, a chain, and a system of gears to convert rider input into motion.
The front wheel is controlled by a handlebar, allowing the rider to steer. Brakes, usually located on the wheels, enable speed control. Bicycles come in various types, including road bikes, mountain bikes, and hybrids, each designed for specific purposes.
Frames can be made from materials such as steel, aluminum, carbon fiber, or titanium, impacting light and durability. Tire types, like road or mountain, cater to different terrains.

Bicycle dynamics focuses on **understanding the mechanical behavior of a bicycle during motion**. The dynamics are governed by **a set of complex equations** that model the **interaction of forces and moments**.
Key parameters include the **bicycle's lean angle**, **steering angle**, and **wheel speeds**, critical for determining **stability** and **control**. Tire-road friction, aerodynamics, and rider inputs significantly influence the system.

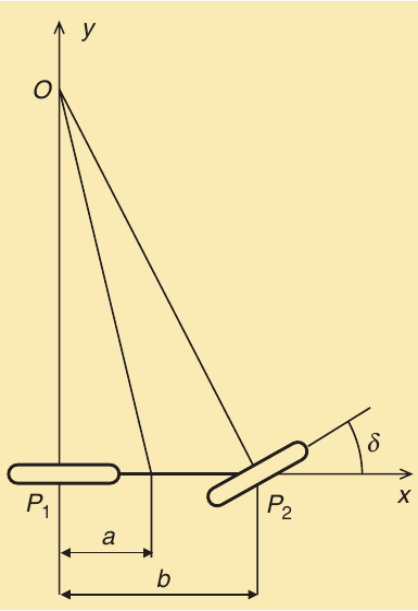
Lateral stability is **crucial**, involving the **ability of the bicycle to maintain balance during turns**.
The **self-stability mechanism** ensures that **small disturbances** lead to **corrective actions**.
Control strategies, often implemented through **feedback loops**, dynamically adjust steering and braking inputs based on real-time conditions.

Assumptions and Simplifications:

- Forces betlen the wheel and the ground are transferred without losses.

- **Control of acceleration** and **braking** is **not considered explicitly**, but I often assume that the **forward velocity is constant**.
- The bicycle **moves on a horizontal plane** and the **wheels always maintain contact with the ground**.
- The bicycle consists of **four rigid parts**, specifically, **two wheels**, **a frame**, and **a front fork with handlebars**.
- The influence of other moving parts, such as pedals, chain, and brakes, on the dynamics is **disregarded**.

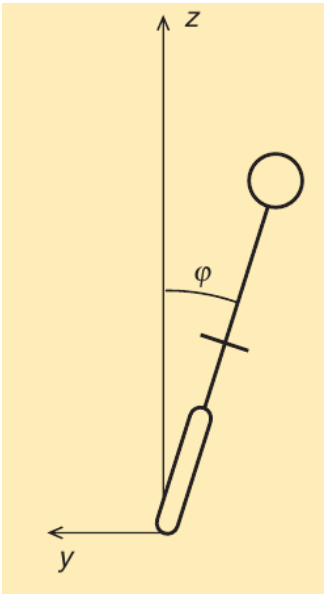
Coordinate System



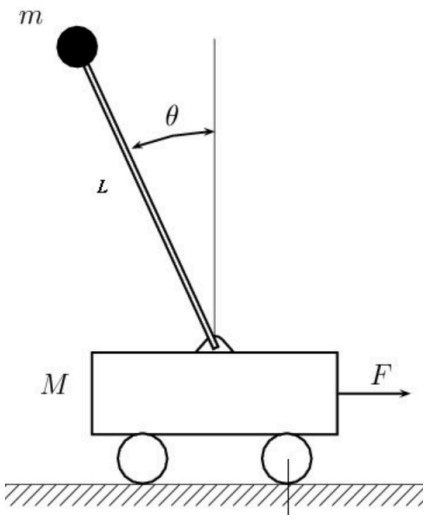
top view of the bicycle - xyz coordinate system

I assume that the coordinate system, xyz, is attached to the bicycle and has its origin at the contact point of the rear wheel. When the rider applies torque to the handlebar and causes the δ angle to change, the coordinate system will rotate around a point that belongs to the Y-axis.

- Bicycle stabilization is similar to inverted pendulum stabilization. I will see in the next section that **their transfer functions** are **very similar**.



rear view

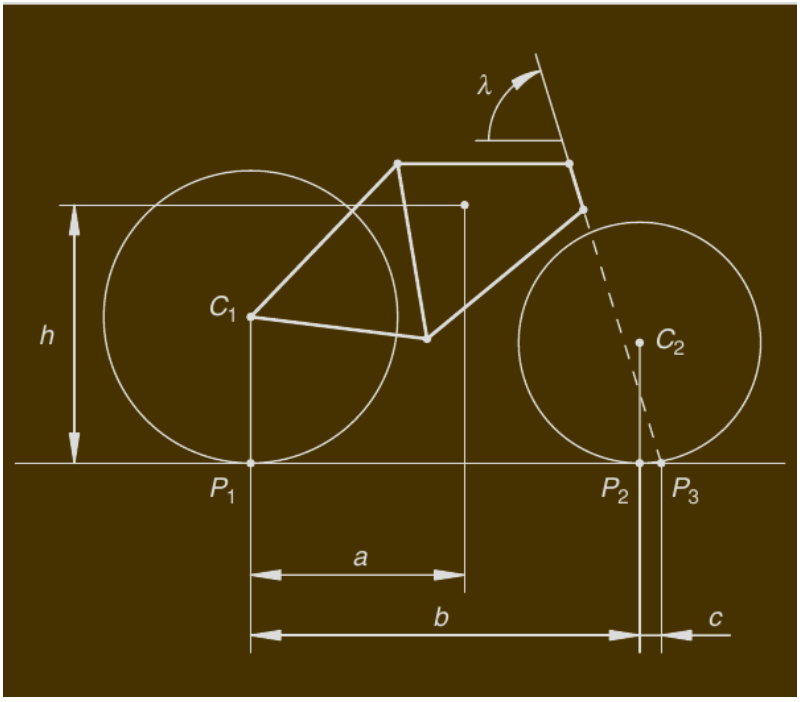


inverted pendulum

2 - Parameters and Model

2.1 - Parameters

I define parameters of the bicycle as follows:



- Origin of the xyz coordinate system is considered at P1.

parameter	description
c	trail distance betlen P2 and P3
a	x_{COM} distance from a vertical line through the center of mass to P1
h	z_{COM} height of the center of mass
m	mass of the bicycle (including the rider)
λ	head angle
b	wheelbase distance from P1 to P2
C1 and C2	wheels rotation axes
P1 and P2	the contact points of the wheels with the ground

P3	intersection of the steer axis with the horizontal plane
φ	roll angle positive when frame leans to right negative when frame leans to right
δ	steer angle
V	forward velocity of the bicycle

2.2 - Models

The first step in designing a controller for a system is to **develop an appropriate and thorough model for it**.
I have to consider our control task and then develop the model based on it.

I're going to use the Analytical Modeling or White-box modeling method, since I're using the governing **mathematical** relations of the system and different parts of it derive the model.

The control task is, as I said before, **"Bicycle Stabilization"**. In this task, only rotational movement of different parts of the bicycle matters, so our model will be built based on rotational motion relations and principles.

2.2.1 - Model#0: The base model

When there are no disturbances and forces from the outside in an ideal environment, Bicycle can be considered **an isolated system**.

So based on the **Angular Momentum Balance principle** and **Newton's second law for rotational motion,** the net torque applied to the frame is:

$$\tau_{net} = I * \frac{\partial^2 \varphi}{\partial t^2} \Rightarrow I * \frac{\partial^2 \varphi}{\partial t^2} = mgh * sin(\varphi) + \frac{DV*sin(\boldsymbol{\lambda})}{b}\frac{\partial \delta}{\partial t} + \frac{m(V^2 h - acg)*sin(\boldsymbol{\lambda})}{b}\delta$$

$$\Rightarrow I * \frac{\partial^2 \varphi}{\partial t^2} - mgh * sin(\varphi) = \frac{DV*sin(\boldsymbol{\lambda})}{b}\frac{\partial \delta}{\partial t} + \frac{m(V^2 h - acg)*sin(\boldsymbol{\lambda})}{b}\delta$$

Where:

- $I * \frac{\partial^2 \varphi}{\partial t^2}$ is the angular momentum of the frame.
- $mgh * sin(\varphi)$ is the torque generated by gravity.
- $\frac{DV*sin(\boldsymbol{\lambda})}{b}\frac{\partial \delta}{\partial t}$ is the inertial torque generated by steering.
- $\frac{m(V^2 h - acg)*sin(\boldsymbol{\lambda})}{b}\delta$ is the torque due to centrifugal forces.

This model is non-linear. Because of the **Sin term**.
Applying linearization, by considering small range of movement:

$$I * \frac{\partial^2 \varphi}{\partial t^2} - mgh\varphi = \frac{DV*sin(\boldsymbol{\lambda})}{b}\frac{\partial \delta}{\partial t} + \frac{m(V^2 h - acg)*sin(\boldsymbol{\lambda})}{b}\delta$$

$I \approx mz_{com}^2 = mh^2$ and $D = -I_{xz} \approx mx_{COM}z_{COM} = mah$:

$$\frac{\partial^2 \varphi}{\partial t^2} - \frac{g}{h}\varphi = \frac{aV*sin(\lambda)}{bh}\frac{\partial \delta}{\partial t} + \frac{(V^2 h - acg)*sin(\lambda)}{bh^2}\delta$$

Assumptions of this model:

1) Small range of movement of the frame ($-0.7\,rad \leq \varphi \leq 0.7\,rad$)
2) The rider sits rigidly on the bicycle, so the effect of his/her leaning will be neglected.

Problems of this model:

1) This model is **completely unstable** (I will prove that in the next section) -> then how come a bicycle maintains its balance? especially when the rider rides with no hands!

2) This model also doesn't take into account the amount of bicycle's velocity in its stabilization. Which is very important. For example, when $V = 0$ it's **impossible** to balance the bicycle.

But when V is **sufficiently large**, this is the time when the bicycle will be stable and won't fall on the ground. This is when it's even possible to **ride with no hands without falling**.

- The model is called the **inverted pendulum model** because of the similarity with the linearized equation for the inverted pendulum.

Transfer function of this model becomes:

$$\frac{\varphi(s)}{\delta(s)} = \frac{\frac{aV*sin(\lambda)}{bh}*s + \frac{(V^2h-acg)*sin(\lambda)}{bh^2}}{s^2-\frac{g}{h}}$$

- This model has two poles at $\pm\sqrt{\frac{g}{h}}$. So because it does have a pole at the right half plane, it's **unstable (as I said before)**.

Let's compare this model transfer function with inverted pendulum transfer function:

$$\frac{\theta(s)}{U(s)} = \frac{1}{Ml}*\frac{-1}{s^2-\frac{g}{h}}$$

- The inverted pendulum model also have two poles at $\pm\sqrt{\frac{g}{h}}$ too.

$\Rightarrow$ so based on this, both systems are **inherently unstable**.

2.2.1.1 - Model#0 Stability Analysis

- *MATLAB codes of this part are in model0.mlx*

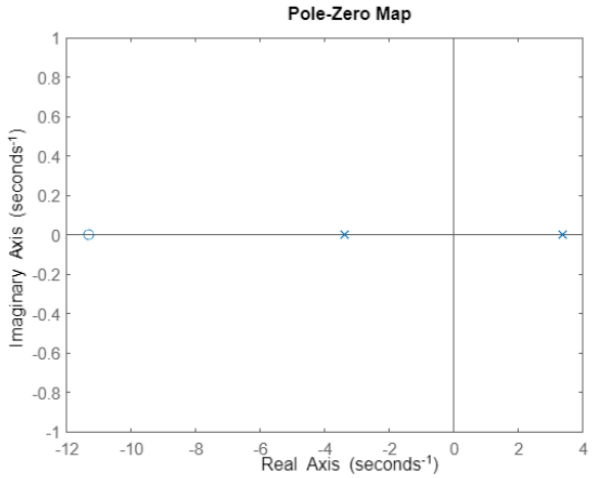
Let's analyze this model.
I choose the values of the parameters as follows:

```
b = 1.02; % meter
c = 0; % meter
lambda = pi/2;
h = 0.86;

V = 5;  % m/s
a = 0.3426;
```

Then the transfer function of this model will become:

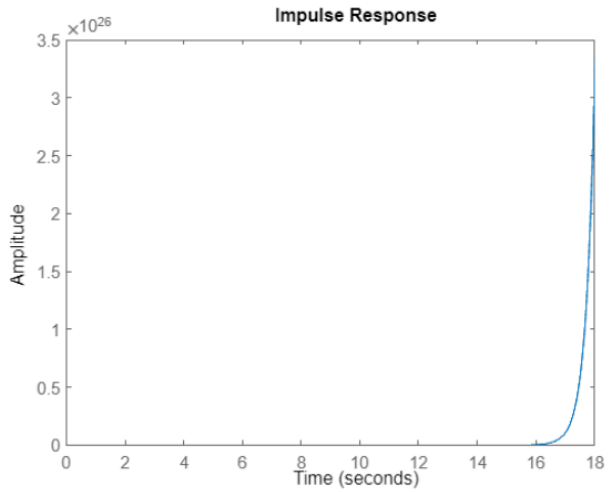
$$\frac{\varphi(s)}{\delta(s)} = \frac{1.952*s + 28.40}{s^2 - 11.4}$$

1) Pole Zero map



- This shows that this model does have **a pole at the right half plane.** So this proves that this model is **unstable**.

2) Impulse Response



- This shows that this mode's impulse response will go to infinity**.** So this proves that the system described by this model **isn't BIBO stable**.

Let's change the value of V to see the effect of this parameter on the model.

```
V = 12.22;  % meter per second
```

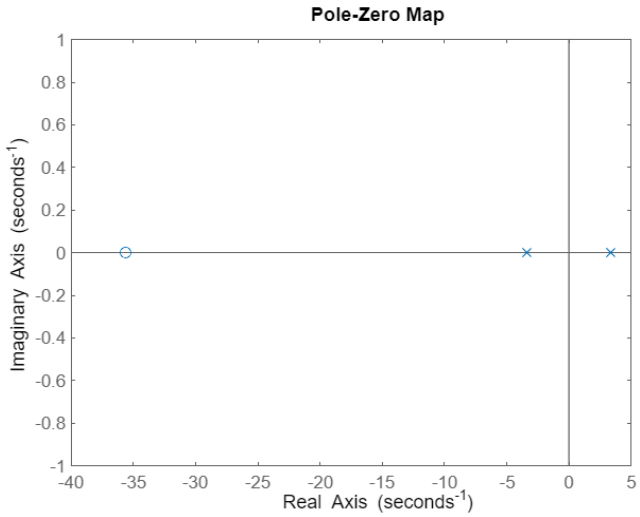
- $V = 12.22\ m/s\ (44\ km/h)$ is considered very large for the V and it's the average speed of "tour de france " riders!

The transfer function of this model will become:

$$\frac{\varphi(s)}{\delta(s)} = \frac{4.771*s + 170.2}{s^2 - 11.4}$$

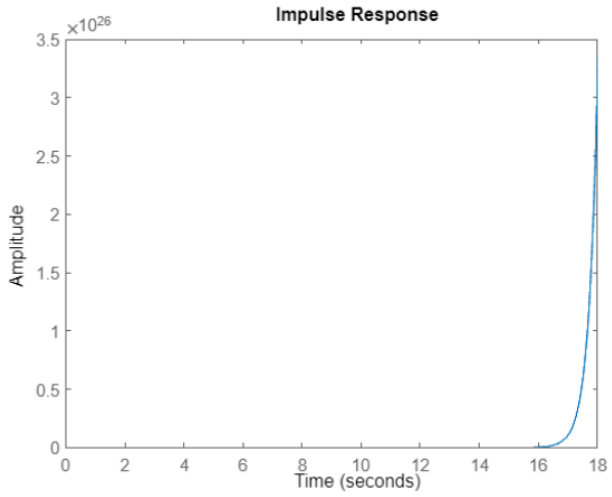
There's **no change** in the denominator of the transfer function! So I expect a similar result.

1) **Pole Zero map**



- This shows that this model does have **a pole at the right half plane.** So this proves that this model is **unstable**.

2) **Impulse Response**

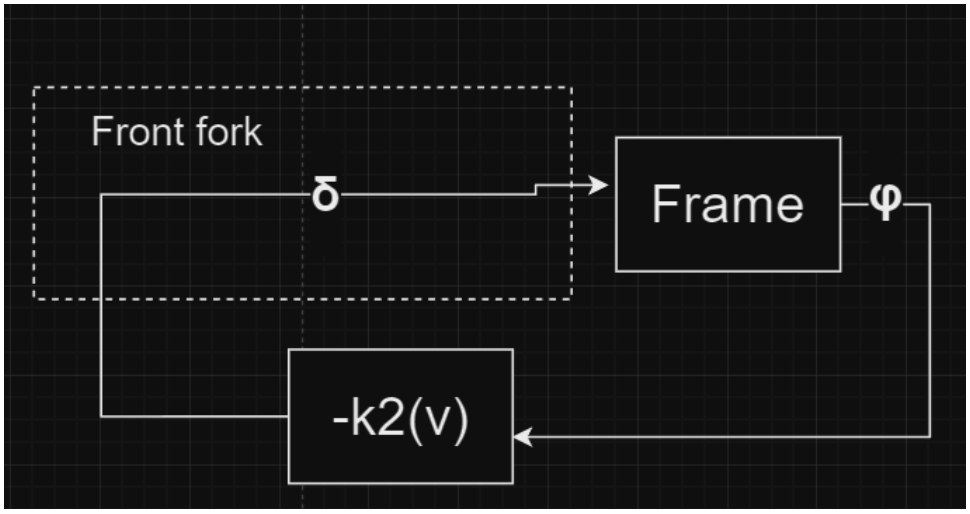


- This shows that this mode's impulse response will go to positive infinity**.** So this also proves that the system described by this model isn't BIBO stable.

What to do then?

In Control, what do I usually do to make an unstable system stable? I use **FeedBack!**

It turns out that **there's feedback betlen the frame angle (lean angle)** and **steering angle** that helps the bicycle to become and remain stable.



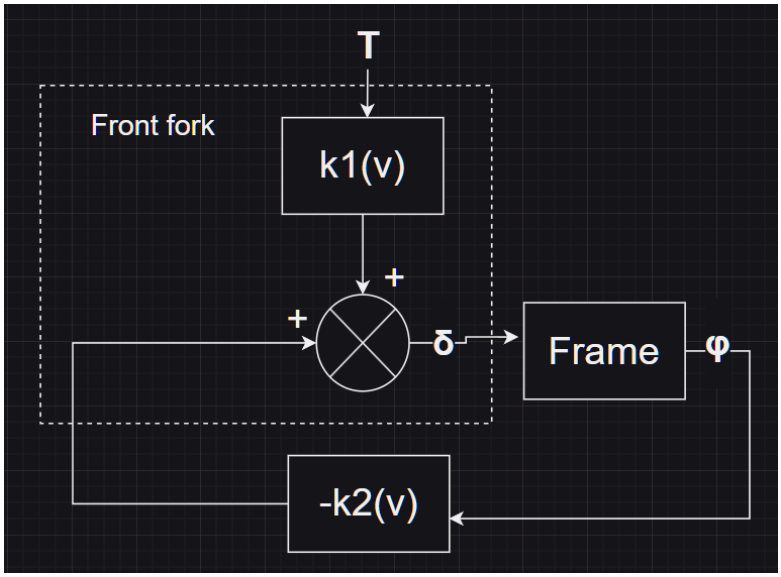
This feedback relation can explain why it’s possible to ride with no hands at sufficiently large speed.

Because the bicycle will **stabilize itself** because of this internal feedback mechanism.

2.2.2 - Model#1: The model with FeedBack

In this model:

- I also consider the feedback betlen the frame angle (lean angle) and steering angle.
- I change the **input** to be the **Torque applied to the handlebar by the rider**. This torque will **change the** δ **angle** directly.



Based on the diagram, I have a new relation as such:

$$\delta = k_1(v)T - k_2(v)\varphi$$

By substituting this into the **model#0** and doing necessary simplifications, I will have:

$$\frac{\partial^2\varphi}{\partial t^2} + \frac{aV^*sin(\lambda)}{bh}k_2(v)\frac{\partial\varphi}{\partial t} + \varphi[-\frac{g}{h} + k_2(v)\frac{(V^2h-acg)^*sin(\lambda)}{bh^2}] = \frac{aV^*sin(\lambda)}{bh}k_1(v)\frac{\partial T}{\partial t} + \frac{(V^2h-acg)^*sin(\lambda)}{bh^2}k_1(v)T$$

- So in this model, input is T and the output is φ .
- And also I have two new variables, $k_1(v)$ and $k_2(v)$, Which are the values of the gains based on the block diagram. They **both depend on** V. This will make this model a **very good model**, because it also takes into account one of the very important parameters of the bicycle in stability which is V, at a very important part of the model, **feedback gain!**

also:

$$k_1(v) = \frac{b^2}{(V^2*sin(\lambda) - bg*cos(\lambda))*mac*sin(\lambda)}$$

$$k_2(v) = \frac{bg}{(*sin(\lambda) - bg*cos(\lambda)}$$

Let's analyze this model further.

$$\frac{\varphi(s)}{T(s)} = \frac{\frac{aV^*sin(\lambda)}{bh}k_1(v)^*s + \frac{(V^2h-acg)^*sin(\lambda)}{bh^2}k_1(v)}{s^2 + \frac{aV^*sin(\lambda)}{bh}k_2(v)^*s + [-\frac{g}{h} + k_2(v)\frac{(V^2h-acg)^*sin(\lambda)}{bh^2}]}$$

- This shows that zeros of the system depend on $k_1(v)$ and poles of the system depend on $k_2(v)$.

and it shows that the $k_2(v)$ variable is very important.

Let's do pole zero analysis on this model:

$$s_{1,2} = \frac{-\frac{aV*sin(\lambda)}{bh}k_2(v) \pm \sqrt{(\frac{aV*sin(\lambda)}{bh})^2 - 4*[-\frac{g}{h} + k_2(v)\frac{(V^2h-acg)*sin(\lambda)}{bh^2}]}}{2}$$

- This also proves that the poles of the system are depend of $k_2(v)$.

2.2.2.1 - Model#1 Analysis

- *MATLAB codes of this part are in model1_unstable__controller.mlx file.*

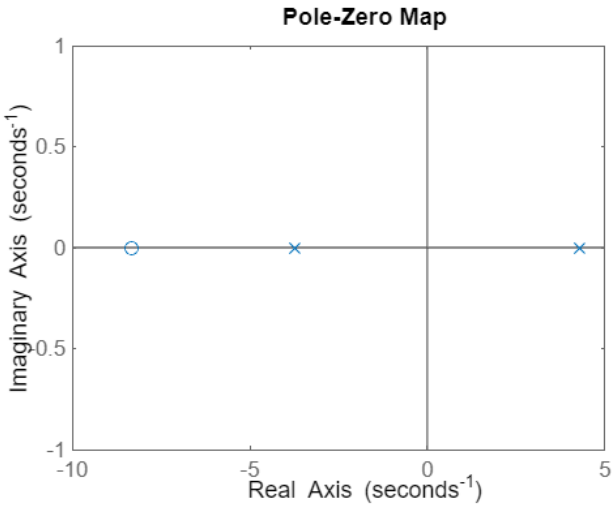
Let's analyze this model too.
I choose the values of the parameters as follows:

```
b = 1.02;
c = 0.08;
h = 0.8603;
lambda = pi/10;
V = 3; % m/s
a = 0.347;
g = 9.81;
m = 84;

k1 = b^2/((V^2*sin(lambda) - b*g*cos(lambda))*m*a*c*sin(lambda));
k2 = b*g/(V^2*sin(lambda) - b*g*cos(lambda));
```

Then the transfer function of this model becomes:

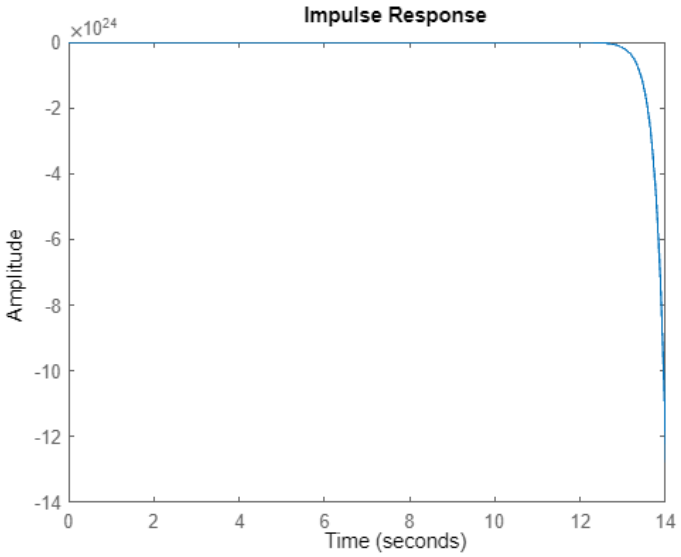
$$\frac{\varphi(s)}{\delta(s)} = \frac{-0.07859*s - 0.6555}{s^2 - 0.5446*s - 15.95}$$

1) Pole Zero analysis



- This shows that this model does have **a pole at the right half plane.** So this proves that this model is **unstable**.

2) Impulse Response



- This shows that this model's impulse response will go to **negative infinity.** So this proves that the system described by this model **isn't BIBO stable**.

Let's change the value of V to see the effect of this parameter on the model.

- *MATLAB codes of this part are in model1_stable.mlx file.*

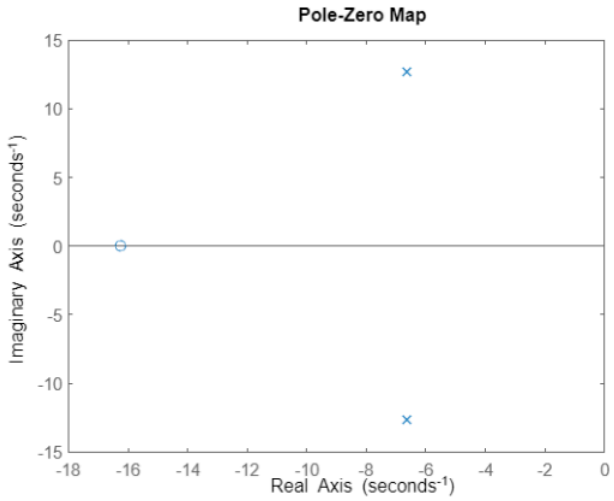
```
V = 5.7;  % m/s
```

The transfer function of this model will become:

$$\frac{\varphi(s)}{\delta(s)} = \frac{1.921*s + 31.25}{s^2+13.31*s+205.2}$$

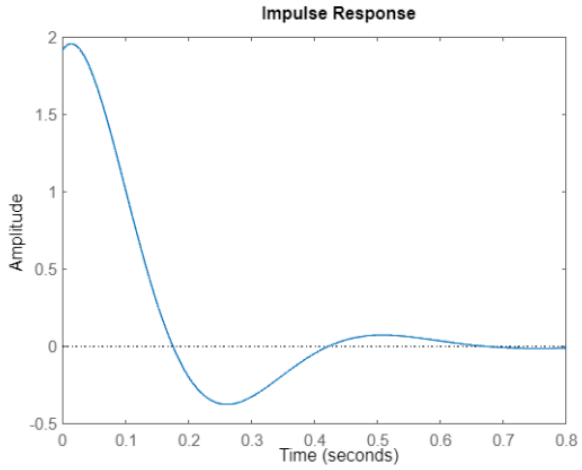
There's **no change** in the denominator of the transfer function! So I expect a similar result.

1) **Pole Zero map**



- This shows that this model doesn't have any pole at the right half plane**.** So this proves that this model is **asymptotically stable**.

2) **Impulse Response**



- This shows that this model's impulse response is limited and goes to zero at $t = \infty$**.** So this proves that the system described by this model is BIBO stable.

- Based on the model#1 pole-zero map analyses, Model#1 takes the importance of the parameter V into account very lll. But this model is very simple and it have **some limitations**:

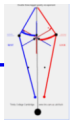
1) This model neglects the Gyroscopic Effect.

This **isn't very bad**, because as discussed in the Ref#3 (~put the title of that in the presentation), bicycle's stability **doesn't rely on gyroscopic effect very much** and it's mostly the steering and the feedback betlen that and the frame that keep the bicycle stable.

- Although, based on this research, Gyroscopic Effect isn't very important, especially at loIr speeds.

Because the Angular Momentum of the Bicycle Wheels isn't very high.
Low speed + Wheels doesn't have much of mass

But at higher speeds, it **can't be negligible**.

UNIVERSITY OF CAMBRIDGE DEPARTMENT OF ENGINEERING Trinity College Clock 

Bicycles are not held up by the gyroscopic effect

including a bike with a reverse-spinning wheel

Dr Hugh Hunt
@hughhunt

For fun stuff on spinning things go to
Dynamics movies page


2) It also doesn't take into account the interaction of different parts of the bicycle with each other. ⇒ Lower Precision

2.2.3 - Model#2: Whipple Model

About this model:

- The dynamics equations for this model follow from **linear and angular momentum balance applied to each part**, along with the assumption that the kinematic constraint forces follow the rules of action and reaction and do not network.
- It neglects the motion of the rider relative to the frame, structural compliances and dampers, joint friction and tyre models with compliance and slip.
- This model also takes **"Gyro Effect"** into account!.

The equations of this model are as follows:

$$M\frac{\partial^2 q}{\partial t^2} + CV\frac{\partial q}{\partial t} + (K_1 + K_2{}^* V^2)q = f$$

in which:

- $q = [\varphi\ \delta]^T$
- $f = [0\ T]^T$

- M - Symmetric mass matrix of the bicycle model
- CV - a damping-like matrix
- K1 and K2 - Stiffness matrices

- **I define the matrices as follows:**

$$M = \begin{bmatrix} m1 & m2 \\ m3 & m4 \end{bmatrix}$$

$$C = \begin{bmatrix} c1 & c2 \\ c3 & c4 \end{bmatrix}$$

$$K_1 = \begin{bmatrix} K_{11} & K_{12} \\ K_{13} & K_{14} \end{bmatrix}$$

$$K_2 = \begin{bmatrix} K_{21} & K_{22} \\ K_{23} & K_{24} \end{bmatrix}$$

- The elements of the matrices depend on the **geometry** and **mass distribution** of the bicycle.
- I'll choose this model as the main model to do State Space Analysis (rest of the project).

3 - State Space Analysis

Here, I'll derive the Whipple Model's State Space realization and analyze it further.

Then I'll design a State feedback controller using the **pole placement technique** for an **unstable system described by this model** and **the observer for state estimation**.

3.1 - State Space Realization

I define the state vector as follows:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \phi \\ \delta \\ \dot{\phi} \\ \dot{\delta} \end{pmatrix} \implies \dot{x} = \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{pmatrix} = \begin{pmatrix} \dot{\phi} \\ \dot{\delta} \\ \ddot{\phi} \\ \ddot{\delta} \end{pmatrix}$$

Trivially, I have:

$$\dot{x_1} = x_3, \dot{x_2} = x_4$$

Now I have to find $\dot{x_3}, \dot{x_4}$

If I multiply and expand the main equations from matrix form, I'll have:

$$m_1 \cdot \ddot{\varphi} + m_2 \cdot \ddot{\delta} + c_1 V \cdot \dot{\varphi} + c_2 V \cdot \dot{\varphi} + \left(k_{11} + k_{21} V^2\right)\varphi + \left(k_{12} + k_{22} V^2\right)\delta = 0$$

$$m_3 \cdot \ddot{\varphi} + m_4 \cdot \ddot{\delta} + c_3 V \cdot \dot{\varphi} + c_4 V \cdot \dot{\varphi} + \left(k_{13} + k_{23} V^2\right)\varphi + \left(k_{14} + k_{24} V^2\right)\delta = T$$

I have to find $\ddot{\varphi} \text{ and } \ddot{\delta}$,so I consider these two as two unknowns and others as known and then try to solve the 2 variables 2 equations system.

After solving it and simplifications, I'll have:

$$\begin{aligned} \ddot{\varphi} &= \dot{\varphi}[A_1 A_2 c_1 \cdot V - A_1 c_3 \cdot V] \\ &+ \varphi[A_1 A_2 a_1 - A_1 a_3] \\ &+ \dot{\delta}[A_1 A_2 c_2 \cdot V - A_1 c_4 \cdot V] \\ &+ \delta[A_1 A_2 a_2 - A_1 a_4] \\ &+ A_1 \cdot T \end{aligned}$$

and I define:

$$A_1 = \frac{m_2}{m_2 * m_3 - m_1 * m_4}$$

$$A_2 = \frac{m_4}{m_2}$$

$$A_3 = \frac{m_1}{m_2}$$

$$a_1 = (k_{11} + k_{21} V^2)$$

$$a_2 = (k_{12} + k_{22} V^2)$$

$$a_3 = (k_{13} + k_{23} V^2)$$

$$a_4 = (k_{14} + k_{24} V^2)$$

substituting:

$$x_1 = \varphi,\ x_2 = \delta,\ x_3 = \dot{\varphi},\ x_4 = \dot{\delta}$$

then I'll have:

$$\begin{aligned}
\dot{x_3} &= x_1[A_1A_2a_1 - Aa_3] \\
&+x_2[A_1A_2a_2 - A_1a_4] \\
&+x_3[A_1A_2c_1 \cdot V - A_1c_3 \cdot V] \\
&+\ x_4[A_1A_2c_2 \cdot V - A_1c_4 \cdot V] \\
&+A_1 \cdot T
\end{aligned}$$

and:

$$\begin{aligned}
\dot{x_4} &= x_1\left[-A_1A_2A_3a_1 + A_1A_3a_3 - \frac{a_1}{m_2}\right] \\
&+x_2\left[-A_1A_2A_3a_2 + A_1A_3a_4 - \frac{a_2}{m_2}\right] \\
&+x_3\left[-A_1A_2A_3c_1 \cdot V + A_1A_3c_3 \cdot V - \frac{c_1 \cdot V}{m_2}\right] \\
&+\ x_4\left[-A_1A_2A_3c_2 \cdot V + A_1A_3c_4 \cdot V - \frac{c_2 \cdot V}{m_2}\right] \\
&-A_1A_3 \cdot T
\end{aligned}$$

So the State Space Realization of this system becomes:

$$A = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
A_1A_2a_1 - Aa_3 & A_1A_2a_2 - A_1a_4 & A_1A_2c_1 \cdot V - A_1c_3 \cdot V & A_1A_2c_2 \cdot V - A_1c_4 \cdot V \\
-A_1A_2A_3a_1 + A_1A_3a_3 - \frac{a_1}{m_2} & -A_1A_2A_3a_2 + A_1A_3a_4 - \frac{a_2}{m_2} & -A_1A_2A_3c_1 \cdot V + A_1A_3c_3 \cdot V - \frac{c_1 \cdot V}{m_2} & -A_1A_2A_3c_2 \cdot V + A_1A_3c_4 \cdot V - \frac{c_2 \cdot V}{m_2}
\end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ A_1 \\ -A_1A_3 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \quad D = 0$$

In MATLAB, i define the parameters as follows:

```
% M matrix -> it's symmetric
m1 = 96.8;     m2 = -3.57;    m3 = -3.57;    m4 = 0.258;

% C matrix
c1 = 0;        c2 = -50.8;    c3 = 0.436;    c4 = 2.20;

% K0 matrix
k11 = -901.0;  k12 = 35.17;   k13 = 35.17;   k14 = -12.03;

% K2 matrix
k21 = 0;       k22 = -87.06;  k23 = 0;       k24 = 3.50;
```

State Space code in MATLAB:

```
a1 = k11+k21*V^2;     a2 = k12+k22*V^2;     a3 = k13+k23*V^2;     a4 = k14+k24*V^2;

A1 = m2/(m2*m3-m1*m4);        A2 = m4/m2;    A3 = m1/m2;
```

```
A = [
    0                                0                              1                              0;
    0                                0                              0                              1;
    A1*A2*a1-A1*a3                   A1*A2*a2-A1*a4                 A1*A2*c1*V-A1*c3*V               A1*A2*c2*V-A1*c4*V;
    -A1*A2*A3*a1+A1*A3*a3-a1/m2      -A1*A2*A3*a2+A1*A3*a4-a2/m2    -A1*A2*A3*c1*V+A1*A3*c3*V-c1*V/m2    -A1*A2*A3*c2*V+A1*A3*c4*V-c2*V/m2;
];
```

```
B = [
    0;
    0;
    A1;
    -A3*A1
];

C = [1 0 0 0];
D = 0;

sys_ss = ss(A, B,C,D);
```

3.2 - Model#2 Stability Analysis in State Space

- Using the model's state space realization that I derived in the last section, i'm going to analyze the stability of Model#2.

3.2.1 - Stable Model

- *MATLAB codes of this part are in model2_stable.mlx.*

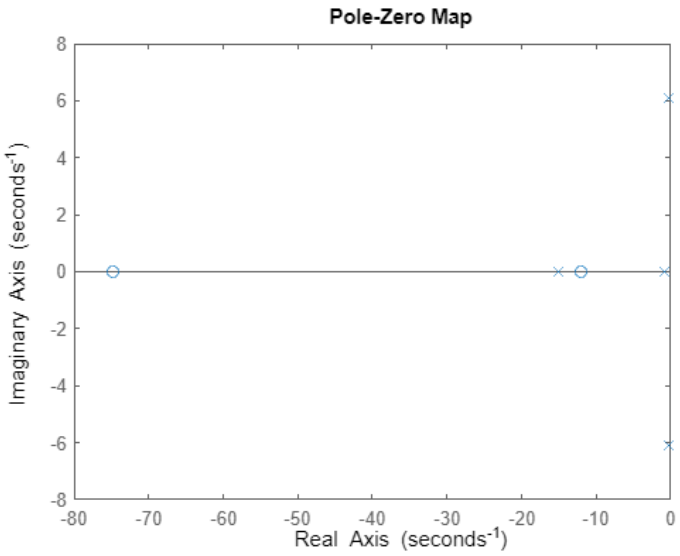
- It turns out that at the bicycle at $V \geq 6\,m/s$ is **stable**, so I choose $V = 6.1\,m/s$ for velocity.

```
V = 6.1;
sys_ss = ss(A, B,C,D);
```

```
A =
          x1       x2       x3       x4
  x1       0        0        1        0
  x2       0        0        0        1
  x3   8.741    33.09  -0.7764     2.62
  x4  -15.36   -0.227   -21.05   -15.76

B =
          u1
  x1       0
  x2       0
  x3  0.2919
  x4   7.915

C =
      x1  x2  x3  x4
  y1   1   0   0   0

D =
      u1
  y1   0
```

1) Pole Zero map

```
>> pzmap(sys_ss)
```



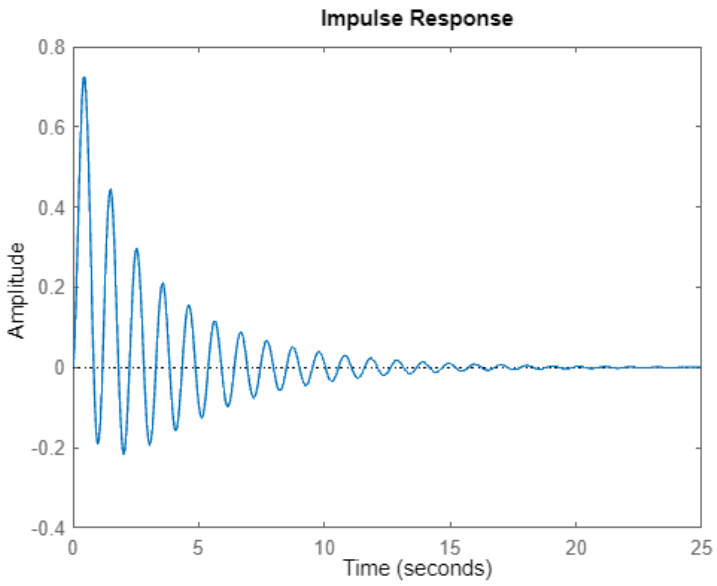

```
>> eig(A)
ans =
  -0.9063 + 0.0000i
  -0.2564 + 6.0736i
  -0.2564 - 6.0736i
 -15.1212 + 0.0000i
```

- This shows that all of the poles of the system are at the left side of the jw axis.

⇒ system (bicycle) is **globally asymptotically stable.**

2) Impulse Response

```
>> impulse(sys_ss)
```



3) Transfer Function

Since finding the transfer function of this system from its equations isn't easy, so I decided to derive it using MATLAB `ss2tf` function.

```
[tf_num, tf_denum] = ss2tf(A,B,C,D)
sys_tf = tf(tf_num, tf_denum)
```

The transfer function will become:

```
 sys_tf =

           0.2919 s^2 + 25.34 s + 262
   ---------------------------------------------
   s^4 + 16.54 s^3 + 58.88 s^2 + 599.3 s + 506.5

 Continuous-time transfer function.
```

```
>> pole(sys_tf)
ans =
 -15.1212 + 0.0000i
  -0.2564 + 6.0736i
  -0.2564 - 6.0736i
  -0.9063 + 0.0000i
```

- This also proves the result of the part#1.

3.2.2 - Unstable Model

- *MATLAB codes of this part are in model2_unstable__controller_observer.mlx*

- As mentioned before, the bicycle at $V \leq 6\ m/s$ is **unstable**, so I choose $V = 5.1\ m/s$ for velocity.

```
V = 3.5;
```

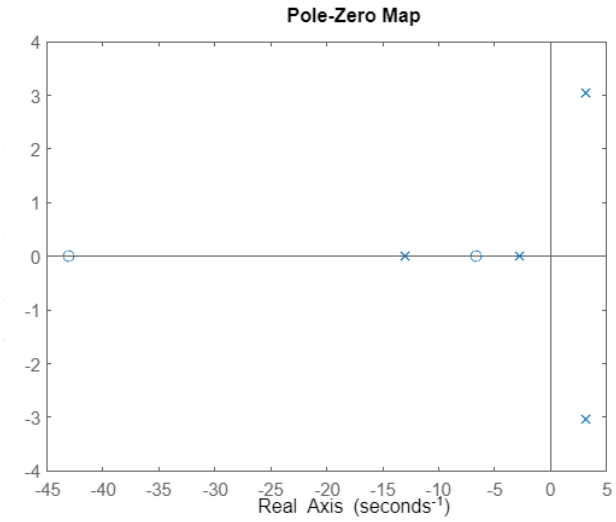
The State Space in MATLAB will become:

```
sys_ss = ss(A,B,C,D)
```

```
A =
          x1       x2       x3       x4
  x1       0        0        1        0
  x2       0        0        0        1
  x3   8.741    23.97  -0.6491     2.19
  x4  -15.36    25.41    -17.6   -13.18

B =
          u1
  x1       0
  x2       0
  x3  0.2919
  x4   7.915

C =
      x1  x2  x3  x4
  y1   1   0   0   0

D =
      u1
  y1   0
```

1) Pole Zero map

```
>> pzmap(sys_ss)
```



also

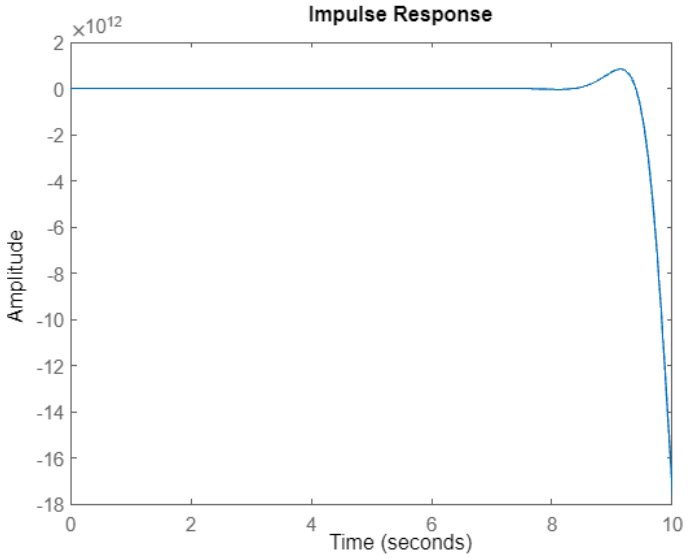
```
>> eig(A)
ans =
   3.1367 + 3.0396i
   3.1367 - 3.0396i
  -2.8048 + 0.0000i
 -12.9589 + 0.0000i
```

- This shows that **two complex poles of the system** are at the right side of the jw axis.

⇒ The system (bicycle) is **unstable.**

2) Impulse Response

```
>> impulse(sys_ss)
```



- Impulse response of the system will go to infinity. This shows that the system isn't BIBO stable.

3) Transfer Function

```
[tf_num, tf_denum] = ss2tf(A,B,C,D)
sys_tf = tf(tf_num, tf_denum)
```

The transfer function will become:

```
  sys_tf =

           0.2919 s^2 + 21.18 s + 182.3
    ----------------------------------------------
    s^4 + 13.83 s^3 + 12.95 s^2 + 323.8 s + 590.3

  Continuous-time transfer function.


>> pole(sys_tf)
ans =
 -14.3039 + 0.0000i
   1.1109 + 4.7325i
   1.1109 - 4.7325i
  -1.7465 + 0.0000i
```

- This also proves the result of the part#1. ⇒ The system is **unstable**.

3.3 - Controllability and Observability Analysis

In this section, using MATLAB, I'll try to analyze the Controllability and Observability properties of the state space representation that I derived in the last sections.

- Both of these properties are **very important** for **designing State Feedback for control** and the **Observer for state estimation**.

```
conty = ctrb(sys_ss.A, sys_ss.B); # controllability matrix
obsy = obsv(sys_ss.A, sys_ss.C); # observability matrix

>> rank(conty)
ans = 4
```

⇒ Rank of the controllability matrix is **equal to the number of states (4)** ⇒ Controllability matrix is **Full rank**. ⇒ State space realization is **fully Controllable**.

```
>> rank(obsy)
ans = 4
```

⇒ Rank of the observability matrix is **equal to the number of states (4)** ⇒ Observability matrix is **Full rank**. ⇒ State space realization is **fully Observable**.

⇒ Also, since this realization is **both Controllable and Observable,** it's also **Minimal**!

⇒ Every other 4th order realization is also minimal

3.4 - System Implementation in Simulink

Since I'm going to use Simulink to show and assess the results of the following sections, I'll explain my implementation of the model in Simulink in this section.

For implementing / realizing this model (model#2) in simulink, I did this:

1) Deriving the CCF state representation.
2) Implementing / Realizing it in Simulink.

1) Derive the CCF state space representation.

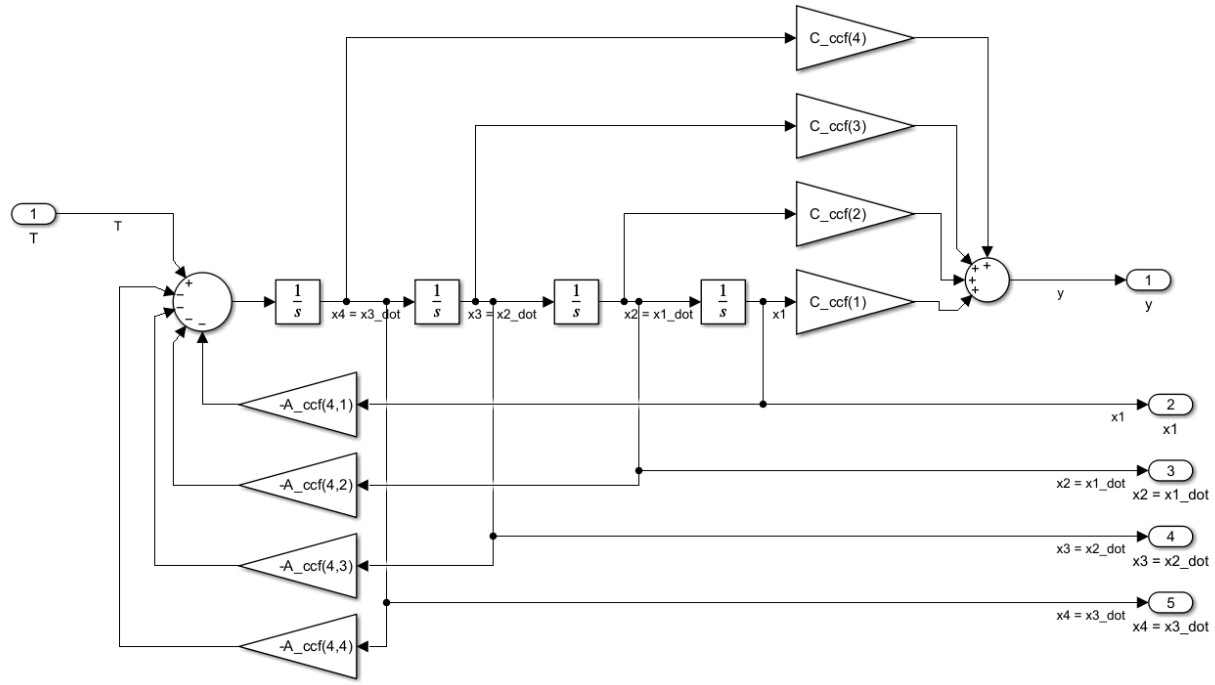
For deriving the CCF state space representation, I first derived the transfer function of the system using MATLAB, and then took the required values for the CCF from that.

```
[tf_num, tf_denum] = ss2tf(A,B,C,D);

A_ccf = [
    0                   1               0               0;
    0                   0               1               0;
    0                   0               0               1;
    -tf_denum(5)    -tf_denum(4)    -tf_denum(3)    -tf_denum(2)
];
B_ccf = [
    0;
    0;
    0;
    1
];
C_ccf = [tf_num(5)  tf_num(4)   tf_num(3)   tf_num(2)];
D_ccf = 0;
```

- Now I have the CCF state space representation.

2) Implementing / Realizing it in Simulink.

Based on what I learned from **chapter#3**, I can realize the CCF ss representation in the real world (here, in Simulink) by having its ss representation.



This is the final implementation of the CCF ss representation in Simulink. In this:

- T is the input.
- y is the output.

- **I’re going to use this representation in the following sections for doing things like, designing State FB controller, designing Observer and so on…**

3.5 - State Feedback Control Using Pole Placement Method

In this section, I'll try to design a State Feedback Controller in MATLAB using **Pole Placement Technique.**

3.5.1 - Designing the Controller

I choose the **unstable model** state space from realizing the last section (3.2.2) and try to design a controller for that.

- *MATLAB codes of this part are in model2_unstable__controller_observer.mlx*
- Based on the section 3.3 results, the state space realization is **Fully Controllable**. So it's possible to design a **State Feedback controller** for that using the **Pole Placement Technique**. So let's do it!

As I mentioned in section 3.2.2, the system is unstable at some velocity.
But it **can be made stable** using **State Feedback control** at that velocity. Because it's **Fully Controllable**.

Let's define the new poles for the system with controller:

```
new_poles = [
  -2 -5+6i -5-6i -10
];
K = place(A,B,new_poles)
```

- System's State Space with State FB will be this:

$$\begin{cases} \dot{x}(t) = (A - BK)x(t) + BFv(t) \\ y(t) = (C - DK)x(t) + DFv(t) \end{cases}$$

Since F is chosen as identity matrix, then the matrices in MATLAB will be:

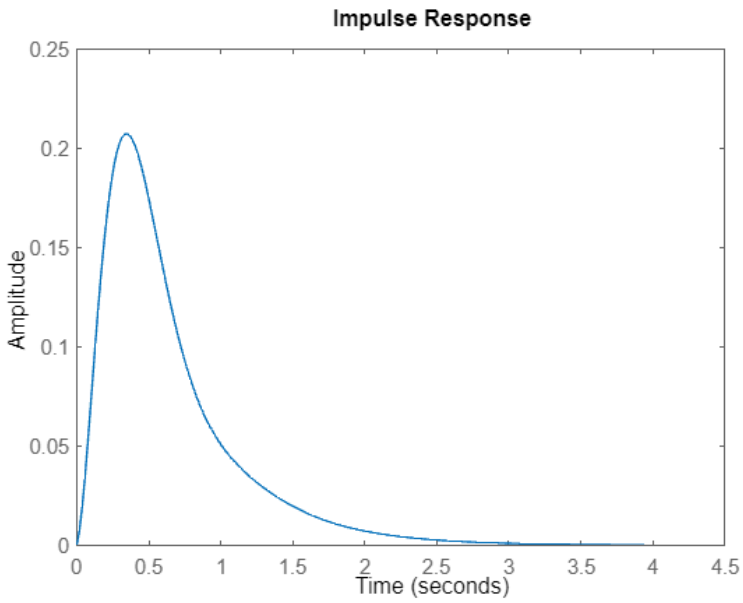
```
A_pp = A-B*K;
B_pp = B;
C_pp = C;
D_pp = D;

sys_ss_pp = ss(A_pp, B_pp, C_pp, D_pp);
sys_ss_pp
```

Now let's analyze the stability of the system + controller:

```
>> eig(A_pp)
ans =
 -10.0000 + 0.0000i
  -5.0000 + 6.0000i
  -5.0000 - 6.0000i
  -2.0000 + 0.0000i
```

and also:

```
>> impulse(sys_ss_pp)
```



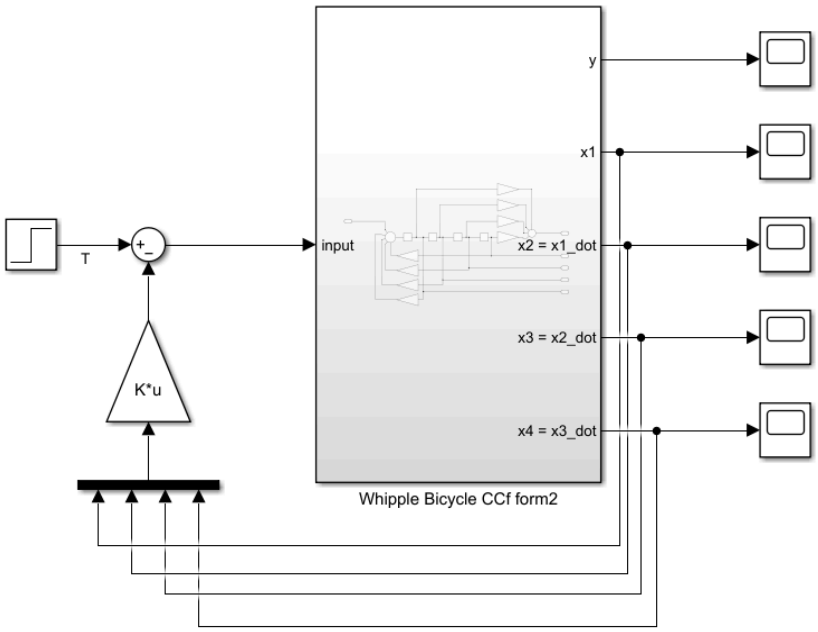
- This also proves that the **system + State FB** controller is **BIBO stable**.

3.5.2 - Implementation in simulink

Since our implementation is based on CCF ss representation, I're going to also find the K matrix for that kind of ss representation.

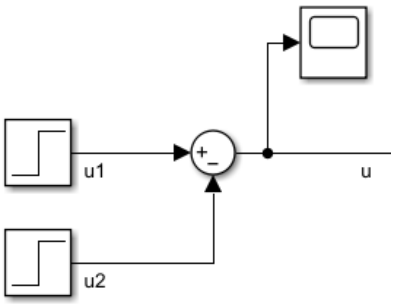
```
new_poles = [
 -2 -5+6i -5-6i -10
];
K = place(A_ccf,B_ccf,new_poles);
```

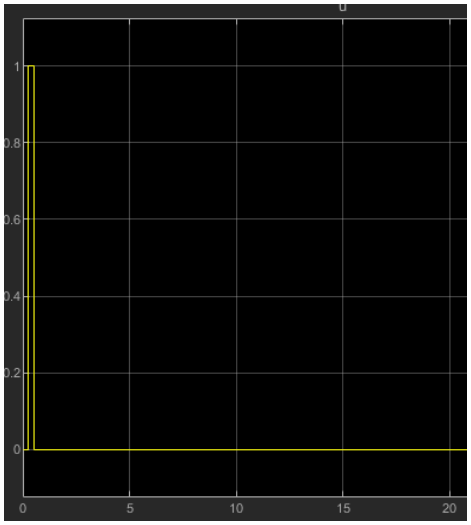
If I assume I got all of the states from the system, then the implementation will be something like this:

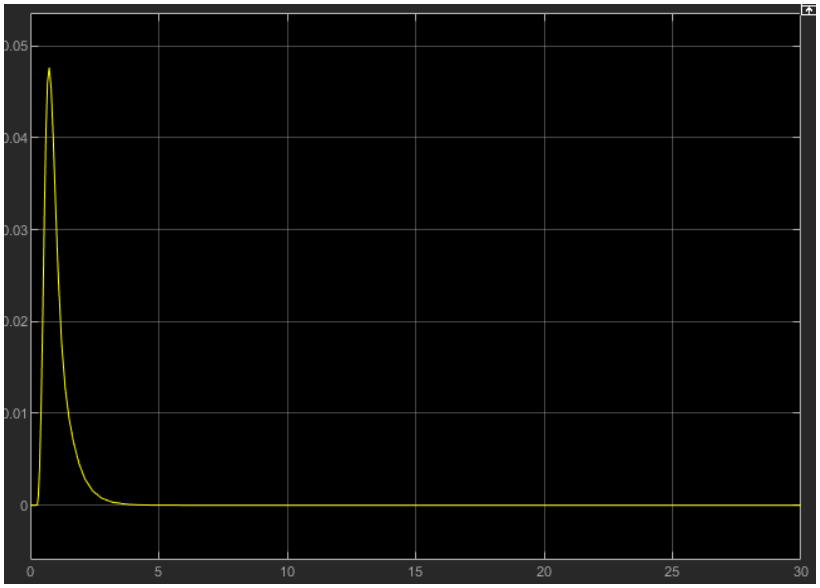


- **Now let's analyze the stability of this system:**

Impulse (kind of) signal as input:







impulse response of the system + FB

- This result shows that **the system (bicycle) + controller** is **Asymptotically Stable**.

3.6 - Observer for State Estimation

In this section, I'll try to design an Observer for State Estimation in MATLAB**.**
The type of observer will be the **Luenberger Observer.**

3.6.1 - Designing the Observer

I choose the **stable model** state space from realizing the last section (3.2.1) and try to design an observer for that.

- *MATLAB codes of this part are in model2_unstable__controller_observer.mlx*
- Based on the section 3.3 results, the state space realization is **fully Observable**. So it's possible to design a **Luenberger Observer** for that.
- Since the model is assumed to be **continuous**, the observer also will be **continuous**.

Let's define the poles for the Observer.:

```
observer_poles = [-12, -12, -12, -12];
```

Now let's find the **Observer Gain (L)** using the `acker` command from MATLAB**.**

```
L_t = acker(A', C', observer_poles);
L = L_t'
>> L =
   34.4424
   64.9835
  388.2318
 -239.6876
```

- Since the observer is **a dynamical system**, it's possible to derive a State Space representation for that like the system itself.

The State Space Dynamical Equations for the Observer are:

$$\begin{cases} \dot{\hat{x}} = (A - LC)\hat{x} + \begin{bmatrix} B & L \end{bmatrix} \begin{bmatrix} u \\ Y \end{bmatrix} \\ Y = C\hat{x} + D \begin{bmatrix} u \\ Y \end{bmatrix} \end{cases}$$

In which:

- **C** is an n by n (n = number of states) Identity matrix. Because I want all the states to show up in the output of the Observer.
- **D** is a zero matrix with the size the same as the B matrix.

Now let's calculate them in MATLAB:

```
A_obs = A-L*C
>> A_obs =
  -34.4424         0    1.0000         0
  -64.9835         0         0    1.0000
 -379.4906   23.1436   -0.6364    2.1474
  224.3242   27.7241  -17.2553  -12.9212

B_obs = [B L]
>> B_obs =
         0   34.4424
         0   64.9835
    0.2919  388.2318
    7.9153 -239.6876

C_obs = eye(length(A_obs))
>> C_obs =
     1     0     0     0
     0     1     0     0
     0     0     1     0
```

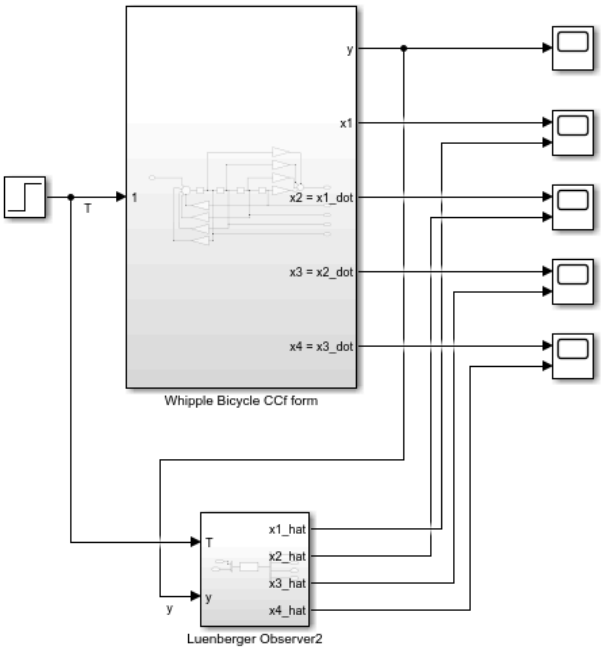
```
     0     0     0     1

D_obs = zeros(size(B_obs))
>> D_obs =
     0     0
     0     0
     0     0
     0     0
```

3.6.2 - Implementation in Simulink

Since our implementation is based on CCF ss representation, I're going to also find the Observer for that.

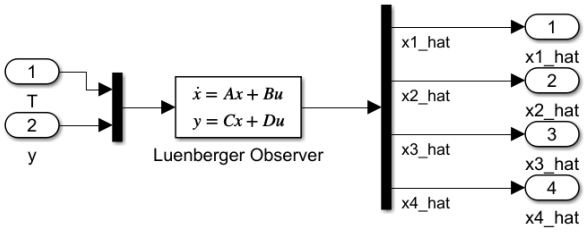
```
obsv_poles = [-20, -20, -20, -20];
L_t = acker(A_ccf', C_ccf', obsv_poles );
L = L_t';

A_obs = A_ccf-L*C_ccf;
B_obs = [B_ccf L];
C_obs = eye(length(A_obs));
D_obs = zeros(size(B_obs));
```

Implementation in Simulink and Error betlen the estimated states with the real ones:



The whole system + Observer

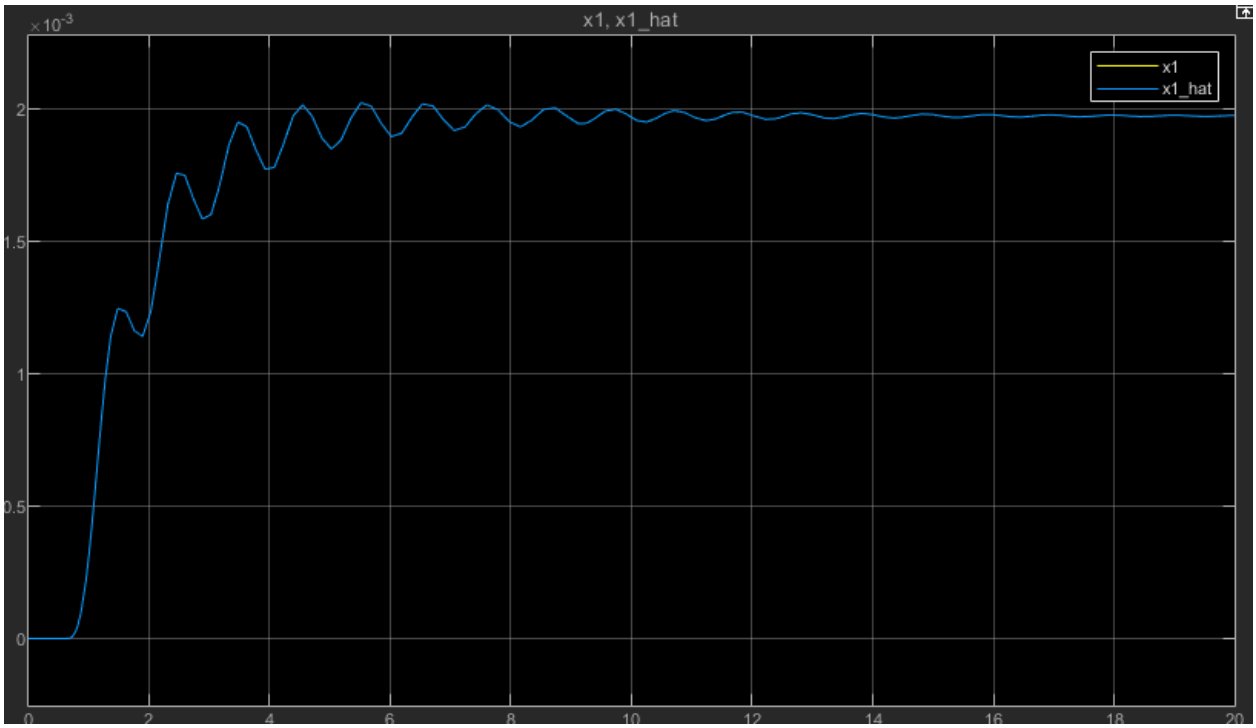
- Here, CCF implementation of the model and the Observer both added as Subsystems.



Inside of the Luenberger Observer subsystem

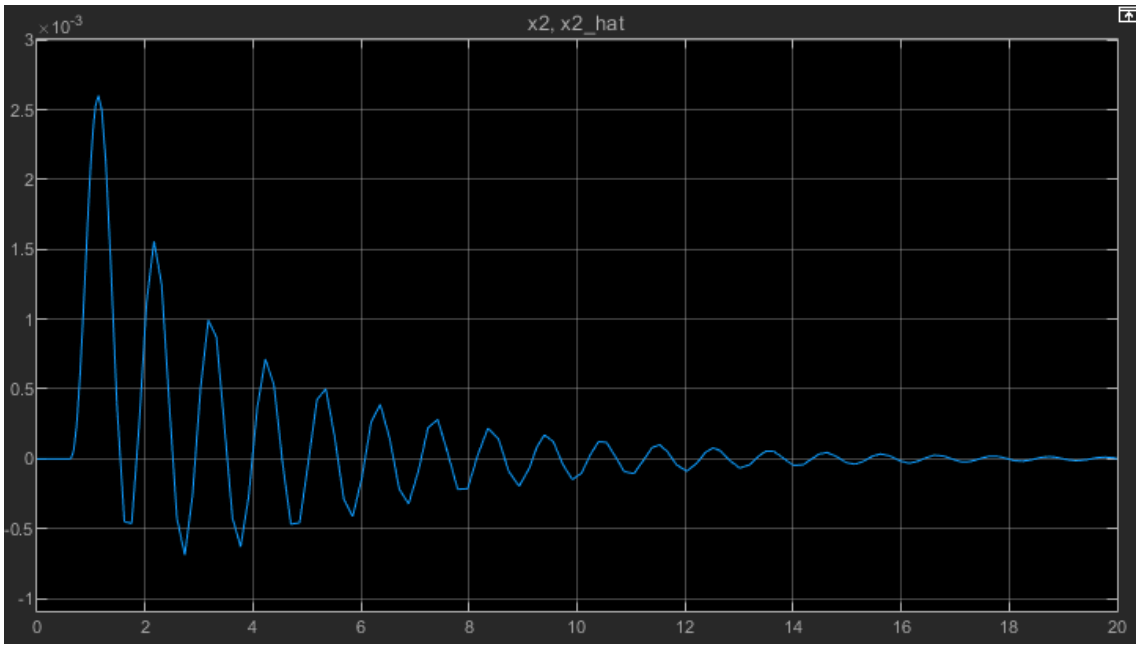
For comparison betlen the real states and the estimated ones from the Observer, I plotted both of them in the same scope:

- x_1 - real and the estimated states:



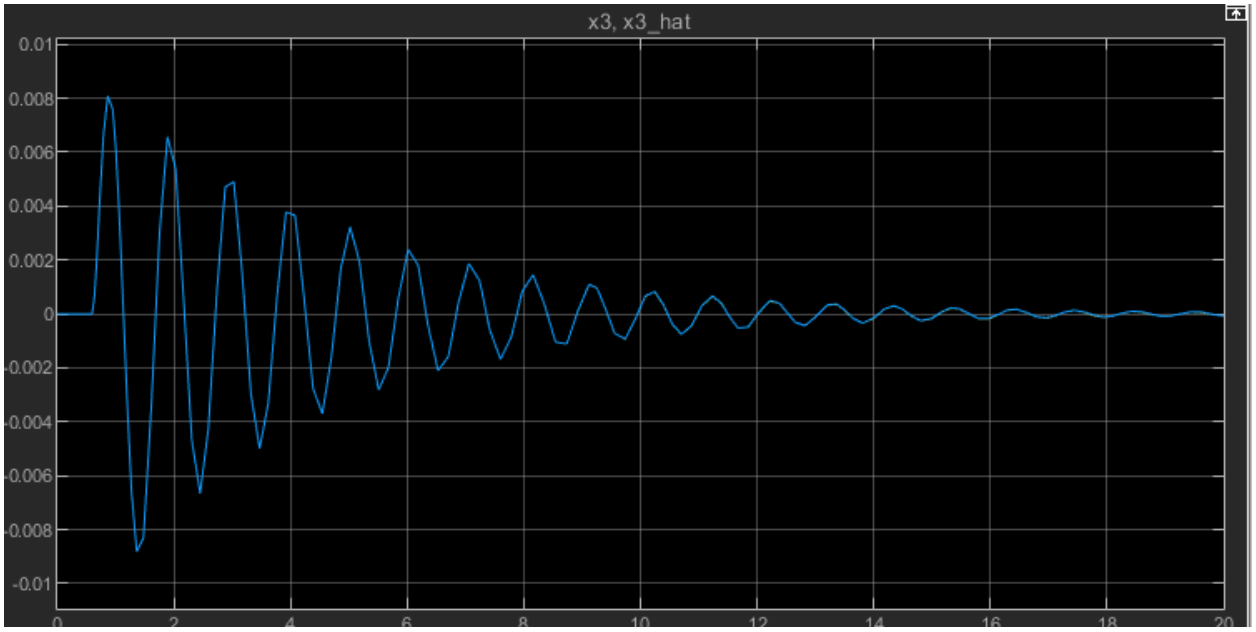
⇒ Since both states are overlapping each other verylll, I can say Observer estimated x_1 properly.

- x_2 - real and the estimated states:



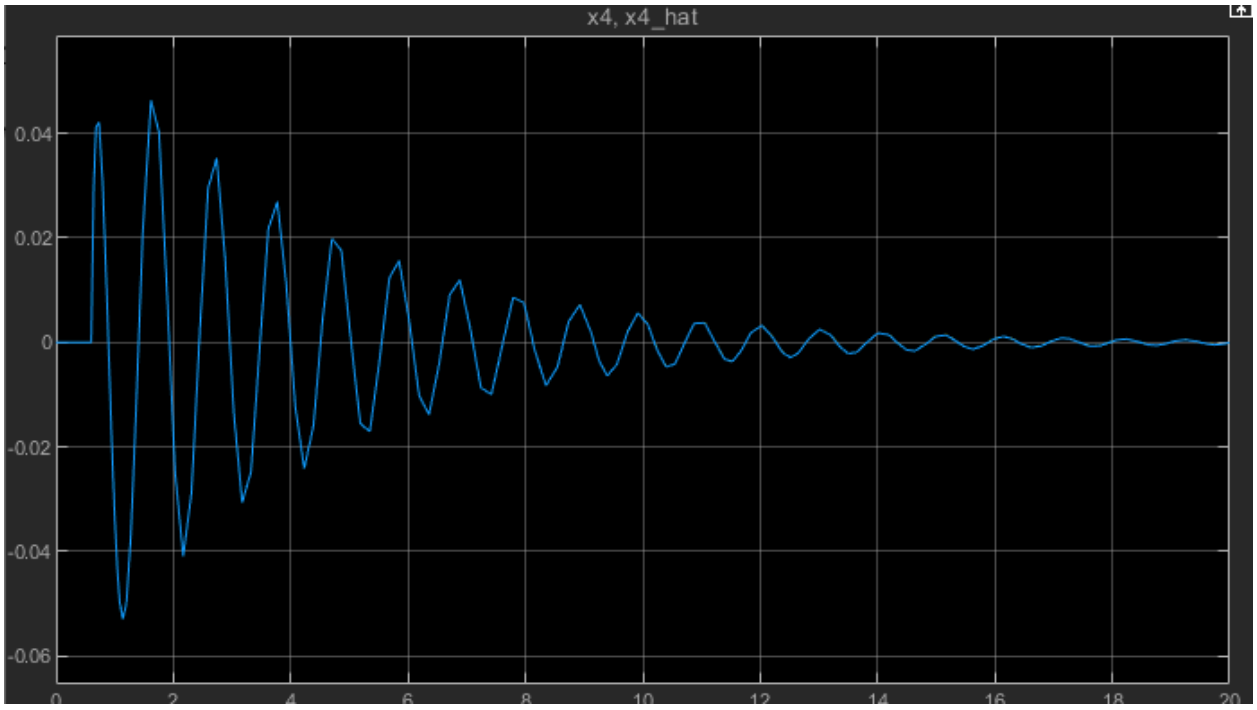
⇒ Since both states are overlapping each other verylll, I can say Observer estimated x_2 properly.

- x_3 - real and the estimated states:



⇒ Since both states are overlapping each other verylll, I can say Observer estimated x_3 properly.

- x_4 - real and the estimated states:



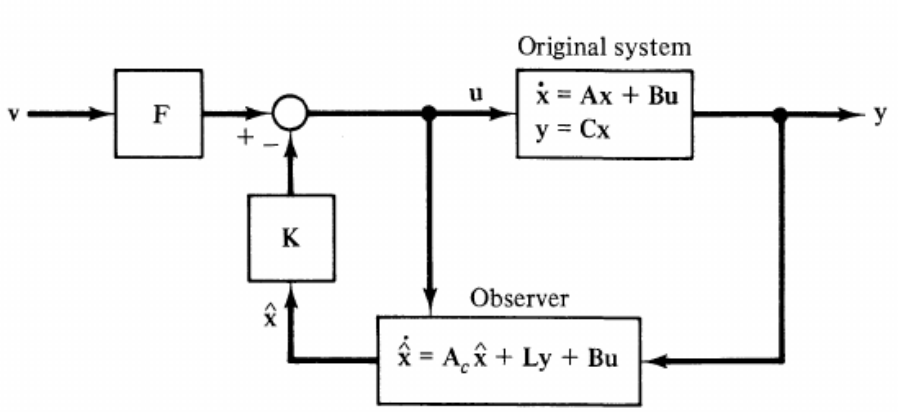
⇒ Since both states are overlapping each other verylll, I can say Observer estimated x_4 properly.

3.7 - State Feedback Control Using Pole Placement Method + Luenberger Observer

3.7.1 - Designing the State Feedback Control + Luenberger Observer

A state feedback controller requires all of the states of the system.
But in reality, **not all states are directly measurable**. This is **where the observer steps in**. The observer utilizes input, output and available sensor data to **estimate the hidden states**.

In this section, I'll add both the State Feedback from section 3.6 and the Observer 3.7 together to make a complete State Feedback controller for my unstable system.

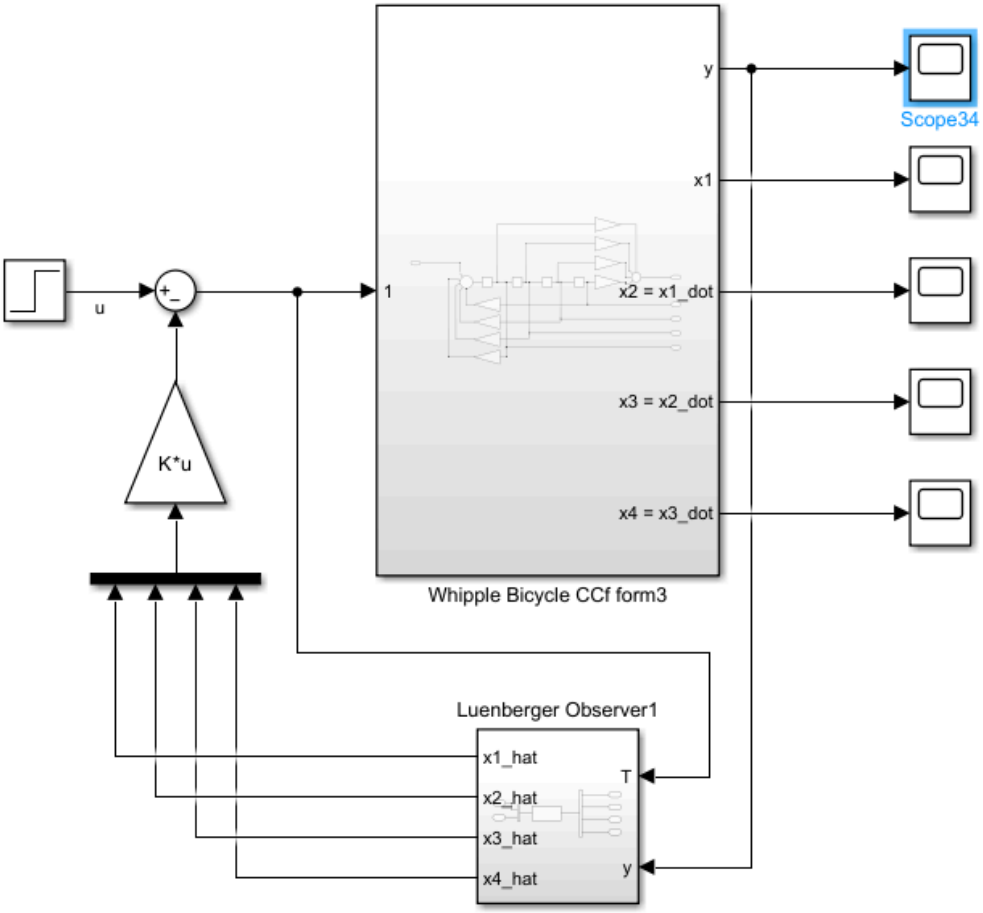


- For designing both of the Controllers (finding the State Feedback gain or K matrix using Pole Placement) and the Observers, based on **"Separation Principle for Feedback Controller"**, which I read about it in **chapter 6** of the course, I have to at first **design both** of the State Feedback Controllers and the Observer **separately**, then couple them together by **feeding the output of the Observer**, which is the system **estimated states** into the State Feedback Controller.

3.7.2 - Implementation in Simulink

This part is easy, since I already have the controller and the observer (from the sections 3.5 and 3.6).
I only have to couple them together.

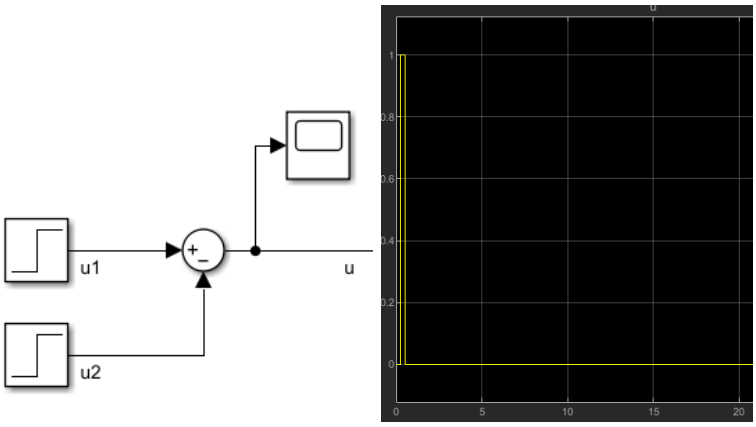
Implementation in Simulink:



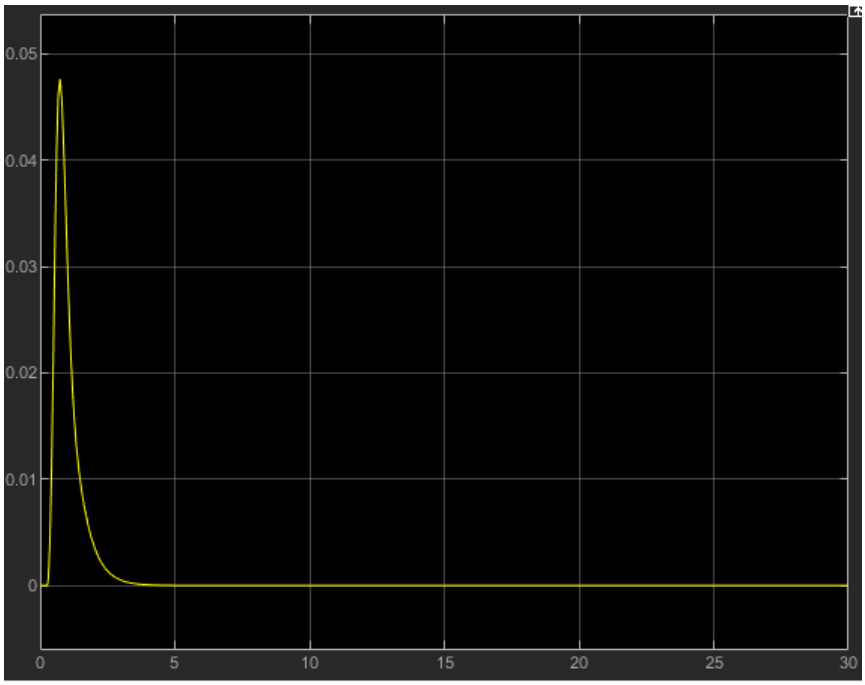
The whole system + Controller (State FB Controller + Observer)

Now let's see the output of the system to impulse input signal:

- I generate the impulse input signal by using the combination of two step signals:



impulse (kind of) signal generation



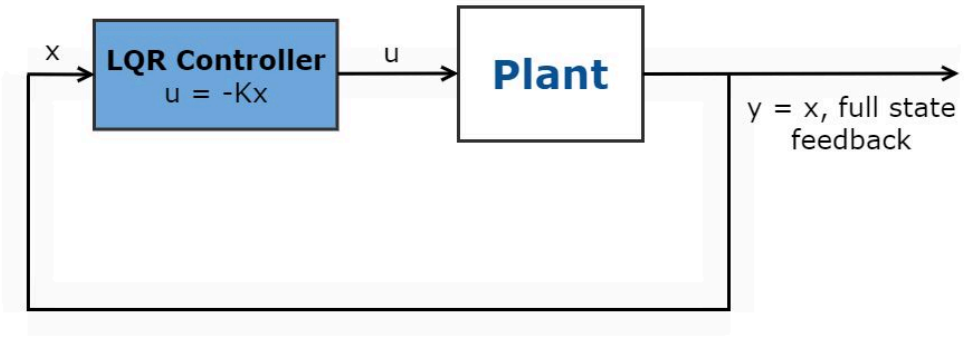
impulse response of the system

- Since the **impulse response of the system** is **finite**,

I can conclude that the system is **Stable** using the **State Feedback Controller (+ Observer).**

4 - Xtra - LQR Controller

4.1 - Description



The Linear Quadratic Regulator (LQR) is a control strategy used to design **Feedback Controllers** for linear systems.

It aims to **minimize a cost function** that represents the **system's performance**.

The LQR controller is chosen for **its ability to improve system performance** by **decreasing settling time**, **rise time**, **overshoot**, and **error**. It is easy to design and increases the accuracy of the state variables by estimating the state.

The LQR controller computes the **optimal control input** given the **system's state** and **desired state**, and it is particularly useful for controlling motor speed and position.

LQR controller cost function is defined as follows:

$$J = \int_0^\infty \left(x^T Q x + u^T R u\right) dt$$

- This cont function does have an analytical solution. That's why this kind of controller is so popular.

The LQR control algorithm is widely used in **various fields**, including **robotics**, **mechatronics**, and **system identification**
It is based on the **principle of finding the optimal feedback gain matrix** to **minimize the cost function associated with the system's behavior**.
The LQR controller is a poIrful tool for achieving **stable** and **efficient** control of linear dynamic systems.

4.2 - Designing the LQR Controller

The way that I design and LQR controller, is by **solving the algebraic Riccati equation**.
Then I'll get a K matrix that I're going to use as the **State Feedback gain**.

I don't do that by hand by ourselves, I simply let MATLAB do this thing for us. I use the `lqr` command from MATLAB.

```
[K,S,P] = lqr(sys,Q,R)
```

I choose the **unstable model state space** from the last section **(3.2.2)** and try to design a LQR controller for that.

1) Choosing the Q and R matrices.

```
Q = [
    1/0.01  0   0   0;
    0   1/0.04  0   0;
    0   0   0   0;
    0   0   0   0;
];
R = 1/4;
```

2) Using the MATLAB `lqr` command to find the K matrix.

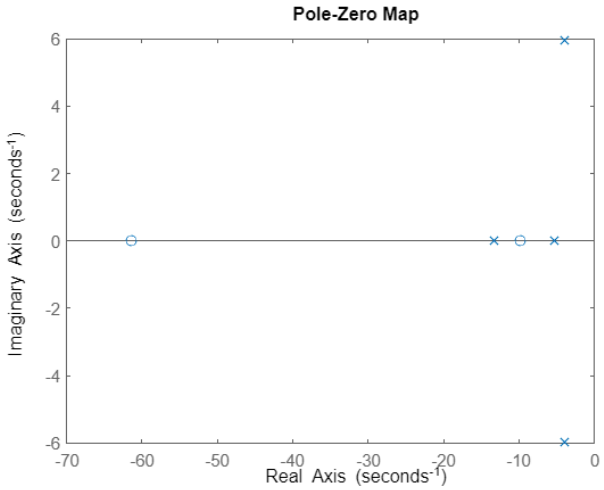
```
[K, S,P] = lqr(sys_ss, Q, R)
>> K =
   25.8018   20.1905    4.4580    1.4741
```

Now let's analyze the stability properties of this system:

```
sys_lqr = ss(A-B*K, B,C,D);
```

1) Poles Zero Analysis

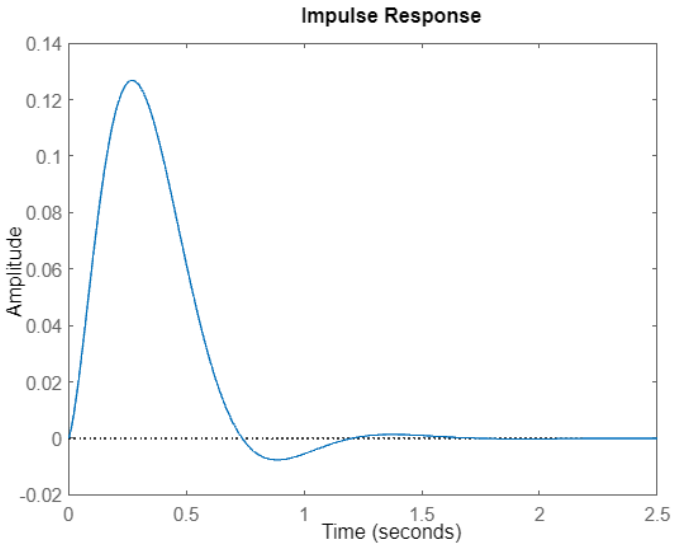
```
>> pzmap(sys_lqr)
```




```
>> pole(sys_lqr)
ans =
 -13.3297 + 0.0000i
  -3.9217 + 5.9548i
  -3.9217 - 5.9548i
  -5.3538 + 0.0000i
```

⇒ Based on these two, I can conclude that the system using the LQR controller is **asymptotically stable**.

2) Impulse Response



- Since the impulse response is finite and doesn't go to infinity, this proves that the **system + State FB** controller (using LQR) is **BIBO stable**.

4.3 - Implementation in Simulink

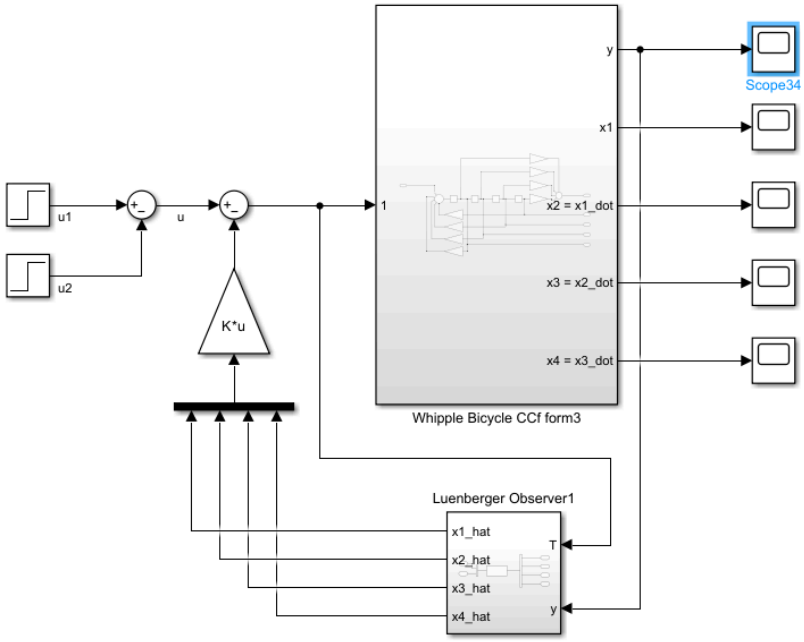
Since our implementation is based on **CCF ss representation**, I'm going to also find the K matrix for that kind of ss representation.

```
%% K matrix using LQR
Q = [
    1/0.01  0    0    0;
    0    1/0.04  0    0;
    0    0    0    0;
    0    0    0    0;
];
R = 1/4;

[K, S,P] = lqr(A_ccf, B_ccf, Q, R);

%% observer on the CCF form
observer_poles = [-26, -26, -26, -26]; % two times faster than the fastest system pole after
using LQR State FB

L_t = acker(A_ccf', C_ccf', observer_poles);
L = L_t';
```

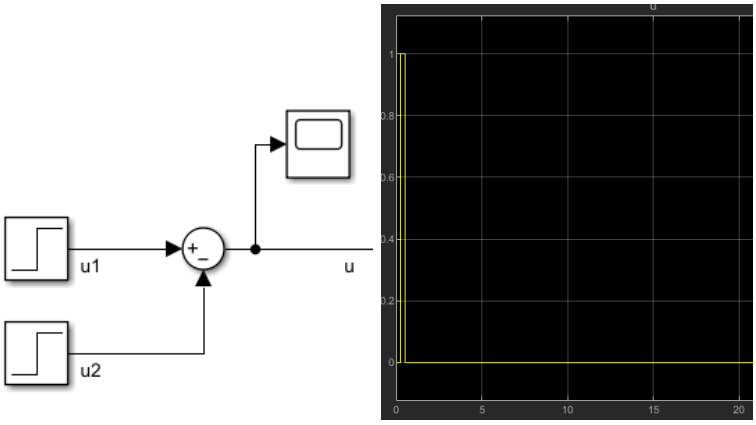
- Now I find the Observer Matrices:

```
A_obs = A_ccf-L*C_ccf;
B_obs = [B_ccf L]; % input is [u Y]'
C_obs = eye(length(A_obs)); % since I want all the states
D_obs = zeros(size(B_obs));
```

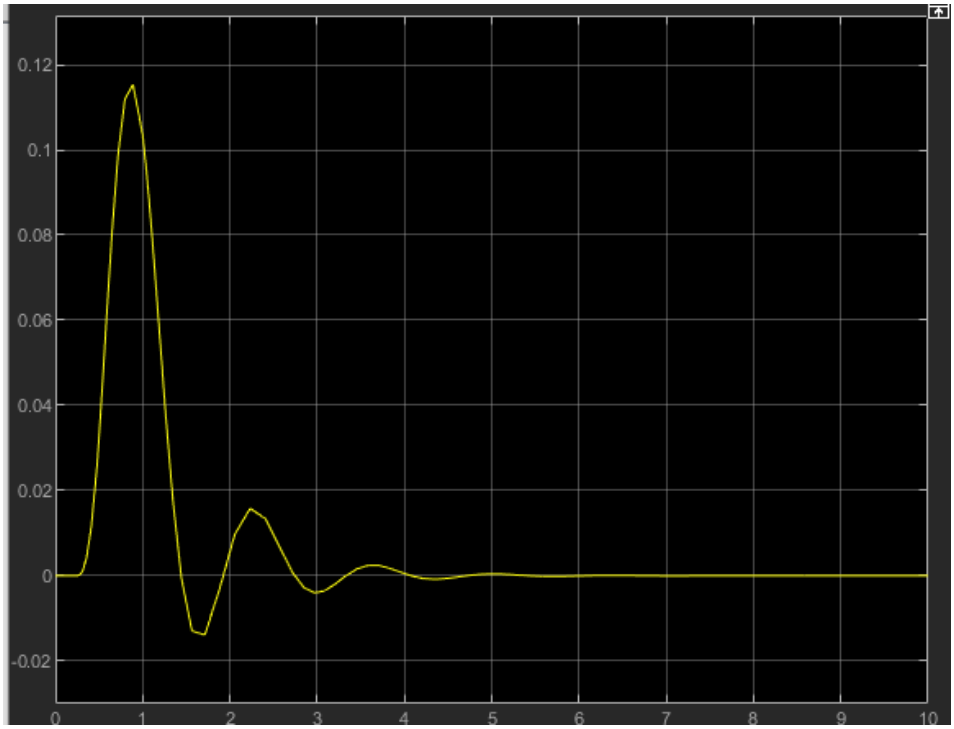
- The System + Controller (+ observer) in this section, will be exactly like the State Feedback from the section 3.7.2.



the whole system + LQR controller (+ Observer)



impulse (kind of) signal generation



impulse response of the system

- Since the **impulse response of the system** is **finite**,

I can conclude that the system is **Stable** using the **State Feedback LQR Controller + Observer)**

5 - Files

- **matlab live scripts**

file	filepath	description
get_bicycle_fb_tf.m	modern_cont_final_project/	get transfer function of the **model#1**
get_bicycle_no_fb_tf.m	modern_cont_final_project/	get transfer function of the **model#0**
model0.mlx	modern_cont_final_project/	**model#0** analysis
model1_stable.mlx	modern_cont_final_project/	**model#1** analysis parameters are chosen in a way that the system is **stable**
model1_unstable__controller.mlx	modern_cont_final_project/	**model#1** analysis + controller (State FB using Pole Placement) parameters are chosen in a way that the system is un**stable**
model2_ccf_transform.mlx	modern_cont_final_project/	live script that contains the code for finding the CCF state space representation of the **model#2**
model2_stable.mlx	modern_cont_final_project/	**model#**2 analysis parameters are chosen in a way that the system is **stable**
model2_unstable__controller_observer.mlx	modern_cont_final_project/	**model#2** analysis + controller (State FB using Pole Placement + observer) parameters are chosen in a way that the system is un**stable**
model2_unstable__lqr.mlx	**modern_cont_final_project/**	**model#2** analysis + controller (State FB using LQR + observer) parameters are chosen in a way that the system is **unstable**

- **simulink**

file	filepath	notes
model1_init_params_stable.m	modern_cont_final_project/simulink/	matlab script that contains initial parameters for the **model#1 + controller (state FB using Pole Placement + Observer)** simulink will use that as **Init Fcn** to load the parameters in its environment parameters are chosen in a way that the system is **stable**
model1_init_params_unstable.m	modern_cont_final_project/simulink/	matlab script that contains initial parameters for the **model#1 + controller (state FB using Pole Placement + Observer)** simulink will use that as **Init Fcn** to load the parameters in its environment parameters are chosen in a way that the system is **unstable**
model1_controller_observer.slx	modern_cont_final_project/simulink/	Simulink model for **stable model#1 system** (CCF representation) + Controller + observer
model1_unstable_controller_observer.slx	modern_cont_final_project/simulink/	Simulink model for **unstable model#1 system** (CCF representation) + Controller + observer
model2_init_params_stable.m	modern_cont_final_project/simulink/	MATLAB script that contains initial parameters for the **model#2 + controller (state FB using Pole Placement + Observer)** simulink will use that as **Init Fcn** to load the parameters in its environment parameters are chosen in a way that the system is **stable**
model2_init_params_unstable.m	modern_cont_final_project/simulink/	MATLAB script that contains initial parameters for the **model#2 + controller (state FB using Pole Placement + Observer)** simulink will use that as **Init Fcn** to load the parameters in its environment parameters are chosen in a way that the system is **unstable**

model2_init_params_unstable__lqr_controller.m	modern_cont_final_project/simulink/	MATLAB script that contains initial parameters for the **model#2 + controller (state FB using LQR + Observer)** simulink will use that as **Init Fcn** to load the parameters in its environment parameters are chosen in a way that the system is **unstable**
model2_controller_observer.slx	modern_cont_final_project/simulink/	Simulink model for **stable model#2 system (CCF representation) + Controller (State FB using Pole Placement)** + observer
model2_unstable__lqr_controller_observer.slx	**modern_cont_final_project/simulink/**	Simulink model for un**stable model#2 system (CCF representation) + Controller (State FB using LQR)** + observer
model2_unstable_controller_observer.slx	modern_cont_final_project/simulink/	Simulink model for un**stable model#2 system (CCF representation) + Controller (State FB using Pole Placement)** + observer

6 - Conclusion

This project served as a thrilling exploration of bicycle dynamics through the lens of modern control theory. By embarking on this journey, we successfully achieved the objective of **analyzing bicycle stability with different control strategies**.
Our investigation unfolded by traversing through various models and control techniques, documented within the following key sections:

1) **Parameters and Models**: This section meticulously laid the groundwork, establishing the essential parameters and constructing various bicycle models. We began with the fundamental "Model#0," followed by an insightful stability analysis. Subsequently, we introduced "Model#1" with feedback and conducted its stability analysis. Finally, we ventured into the realm of the classic "Whipple Model" and investigated its stability characteristics.

2) **State Space Analysis:** Diving deeper into the system's dynamics, we delved into state space representations. Here, we achieved a crucial milestone by identifying the "right" realization and performing comprehensive stability analysis, differentiating between stable and unstable models. Additionally, we assessed the system's controllability and observability, providing valuable insights into its controllability potential.

3) **System Implementation in Simulink:** Transforming theory into practice, we utilized Simulink as our platform for system implementation. This section covered several key milestones:

4) **State Feedback Control using Pole Placement:** We designed and implemented a state feedback controller using the pole placement method, gaining hands-on experience with shaping the system's response.

5) **Observer for State Estimation:** Recognizing the importance of state estimation, we designed and implemented an observer to compensate for unmeasured states, enhancing control accuracy.

6) **State Feedback Control + Luenberger Observer:** Combining the strengths of both techniques, we implemented a comprehensive control system incorporating state feedback and state estimation for improved performance.

7) **Xtra - LQR Controller:** As an enriching extension, we ventured into the realm of LQR control. This section detailed the design and implementation of an LQR controller, offering an alternative approach to achieving optimal control performance.

By meticulously progressing through these stages, we gained a profound understanding of how modern control techniques can influence and stabilize bicycle dynamics. This project served as a testament to the power of mathematical modeling, control design, and simulation in unraveling the complexities of a seemingly simple system like a bicycle.

However, our journey doesn't end here. Future endeavors could explore advanced control strategies, incorporate rider dynamics, or delve into real-world implementations. Regardless of the chosen path, this project has undeniably ignited a passion for understanding and controlling dynamic systems, paving the way for future explorations in this captivating field.

7 - References

1. Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review | Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences
2. Dynamics and Optimal Control of Road Vehicles - D. J. N. Limebeer and Matteo Massaro
3. Hugh Hunt - Cambridge University - Are Gyroscopic Effects Significant When Riding A Bicycle ?
4. Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review | Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences
5. Lateral dynamics of a bicycle with passive rider model