

# **Project Report**

## **Title:**

Control of an eight thruster Remotely operated Underwater Vehicle (ROV) model using PID and LQR controllers.

## **Abstract:**

To make two different control system models in Simulink on PID and LQR controllers respectively and to get the results for positional and velocity parameters of the ROV based on the desired reference inputs. To tune parameters related to both the controllers to achieve better results and finally test the model physically underwater.

## **A) PID Controller:**

### **Literature Review:**

The basic mathematical model of an ROV is given by:

$$\dot{\eta} = J(\eta)v$$

$$M\dot{v} + C(v)v + D(v)v + g(\eta) = \tau$$

- where M = Mass matrix (with added mass effect taken into account)  
C = Coriolis force matrix  
D = Damping forces matrix  
g = Restoring forces matrix  
 $\tau$  = Forces & moments matrix

### **1) Added Mass Effect:**

The added mass of an object is the effect in which some mass of fluid surrounding the object under observation is accelerated/decelerated along with it.

### **2) Coriolis Effect:**

The Coriolis force is a fictitious force that comes into play whenever we are trying to explain the forces on an object with respect to a rotating frame.

### 3) Ziegler-Nichols Method for tuning PID's:

This method can be used to tune our PID both in the case where we have a working model for our plant or even when we don't. The first step to this method is measuring two parameters:  $K_U$  which is the gain at which the system becomes marginally stable and  $T_U$  which is the period of oscillation at marginal system response. These values are found by taking  $K_I$  and  $K_D$  values to be zero for that input and changing  $K_P$  until marginal stability is achieved.

After these parameters are evaluated controller gains can simply be calculated from the below table:

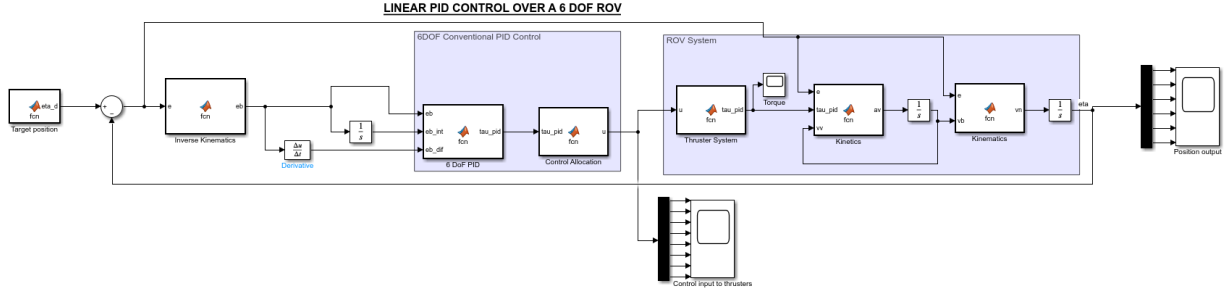
Control Type	$K_P$	$T_i$	$T_d$	$K_I = K_P/T_i$	$K_D = T_d K_P$
PID (classic)	$0.6 K_U$	$T_U/2$	$T_U/8$	$1.2 K_U/T_U$	$0.075 K_U T_U$
P	$0.5 K_U$	-	-	-	-
PI	$0.45 K_U$	$T_U/1.2$	-	$0.54 K_U/T_U$	-
PD	$0.8 K_U$	-	$T_U/8$	-	$0.1 K_U T_U$
Pessen Integration	$0.7 K_U$	$2T_U/5$	$3 T_U/20$	$1.75 K_U/T_U$	$0.105 K_U T_U$
Some Overshoot	$K_U/3$	$T_U/2$	$T_U/3$	$(2/3) K_U/T_U$	$(1/9) K_U T_U$
No Overshoot	$0.2 K_U$	$T_U/2$	$T_U/3$	$(2/5) K_U/T_U$	$(1/15) K_U T_U$

### 4) Controller Models:

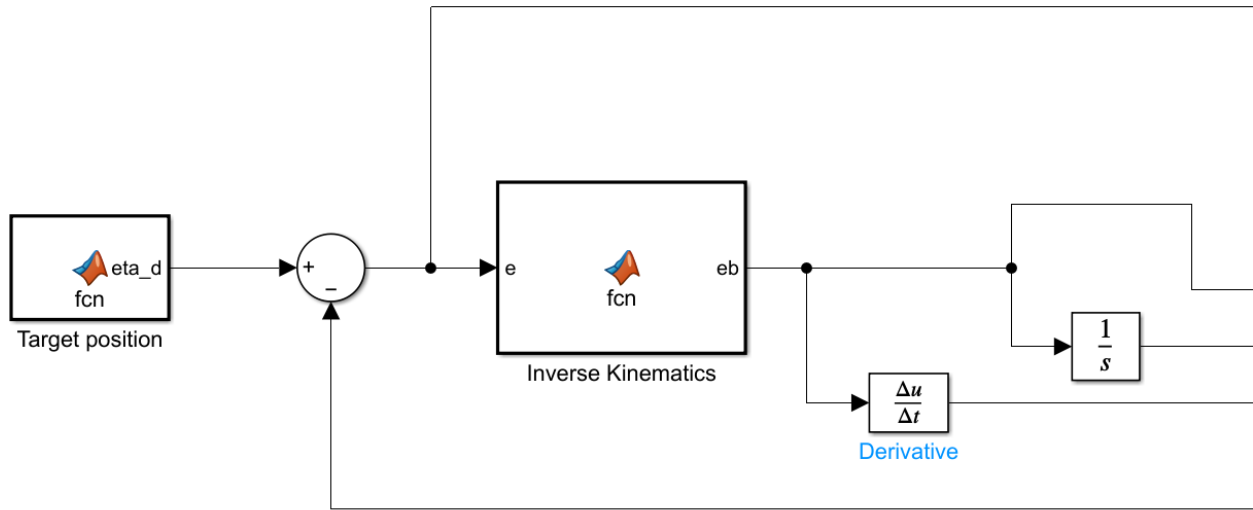
The two controller models that we used are:

#### A. Linear Model:

The schematic of the simulink model created using the linear PID control looks like this:



It consists of different functionalities in each block of the design:



The PID controller uses an error signal, called the tracking error, generated from the difference between the desired position and the current position of the rover.

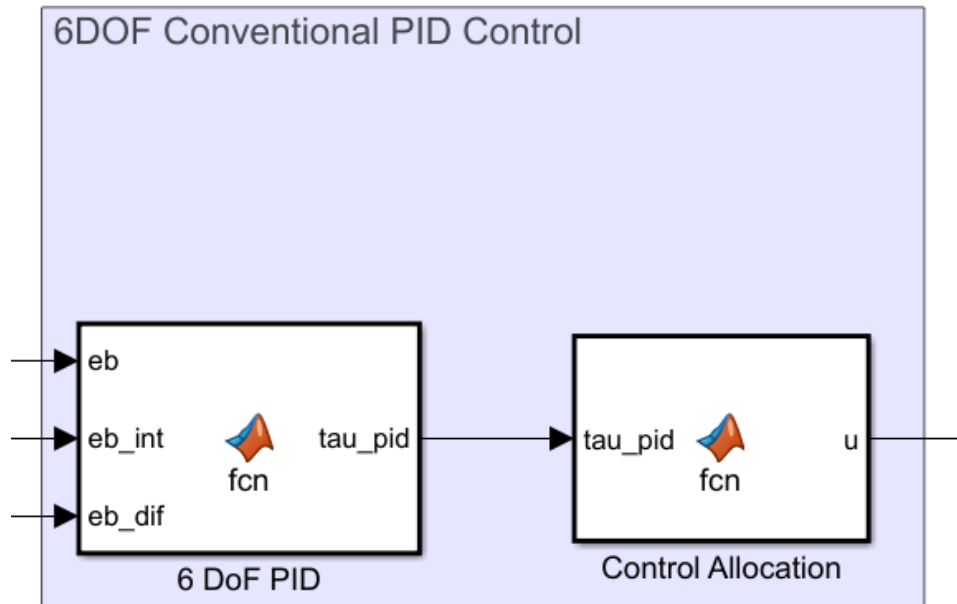
$$e = \eta_d - \eta$$

This error signal, in the world frame, is converted to the error signal in body frame  $e^b$  using the following equation:

$$e^b = J^T(\eta) e$$

where the transformation matrix from the vehicle body frame to the world reference frame using Euler angle transformation is given by:

$$J_{\Theta}(\eta) = \begin{bmatrix} R_b^n(\Theta) & 0_{3 \times 3} \\ 0_{3 \times 3} & T_{\Theta}(\Theta) \end{bmatrix}$$



Using the error signal in the body frame  $eb$ , the torque generated by the PID can be calculated using the equation:

$$\tau_{PID} = K_P e^b(t) + K_I \int_0^t e^b(t') dt' + K_D \frac{de^b(t)}{dt}$$

In order to generalise the required control forces, control allocation calculates the control input signal  $u$  to apply to the thrusters. The control forces due to the control inputs applied to the thrusters can be expressed as:

$$\tau = T(\alpha)F = T(\alpha)Ku$$

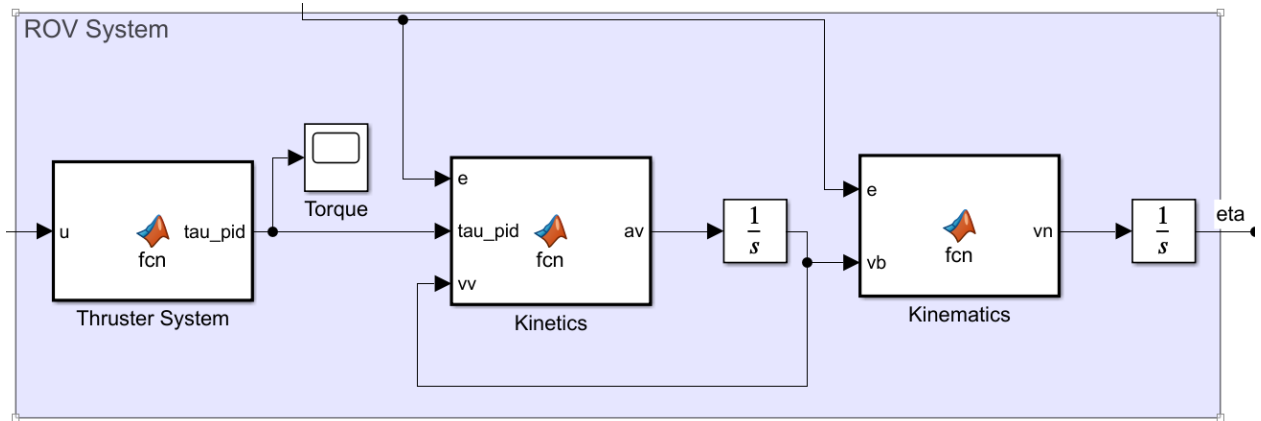
As a result, the control input vector can be derived as:

$$u = K^{-1}T^{-1}\tau$$

For the linear model, the following values of  $K_P$ ,  $K_I$ , and  $K_D$  have been obtained using the Ziegler-Nichols method:

6-DoF PID	Surge	Sway	Heave	Roll	Pitch	Yaw
$K_P$	3	3	3	4	4	2

$K_I$	0.2	0.2	0.2	0.3	0.3	0.1
$K_D$	2.5	2.5	0.5	0.5	1	0.5



Using the control input vector, the thruster system generates the control forces in 6 DoFs with the help of the above mentioned equation:

$$\tau = T(\alpha)F = T(\alpha)Ku$$

Since the 8 thrusters of the ROV produce a maximum thrust of 40N at operating voltage of 16V, the thrust coefficients are approximated to 40. Thus the thrust coefficient matrix  $K$  is taken as

$$K = \text{diag}[40, 40, 40, 40, 40, 40, 40, 40]$$

After obtaining the torque vector, the kinetics is used to determine the acceleration in the body frame for the given forces using the state equation:

$$M\dot{v} + C(v)v + D(v)v + g(\eta) = \tau$$

The kinematics block is then used to define the vehicle velocity in the world frame  $v''$ . The position of the vehicle is then determined in the integrator and, using inverse kinematics, is converted to the body frame before being supplied back into the controller.

## B. Non-Linear Model:

Because of disturbances underwater like current speed, there will be non-linearities introduced into the system. This makes the linear-PID controller model inappropriate to use as there will be

a lot of deviations from the desired o/p and noise in the system. So, we need to include the system dynamics to get the control input to the thrusters.

In the nonlinear model-based PID control system design, the dynamic model of the ROV is utilized to produce a 6-DoF predictive force and the model-based PID is used to provide a corrective force in 6 DoFs to adjust the error in the model. This is advantageous in that the model error and nonlinearities tend to be smaller than the dynamics themselves.

In the predictive force generation, a virtual reference trajectory strategy is introduced for the design of trajectory tracking. With the use of a scalar measure of tracking in Fossen (Fossen, 1994), a virtual reference  $x_r$  can be defined that satisfies:

$$\dot{x}_r = \dot{x}_d + \lambda e_b$$

where  $\lambda > 0$  is the control bandwidth that describes the amount of tracking error to the overall tracking performance, and  $e_b$

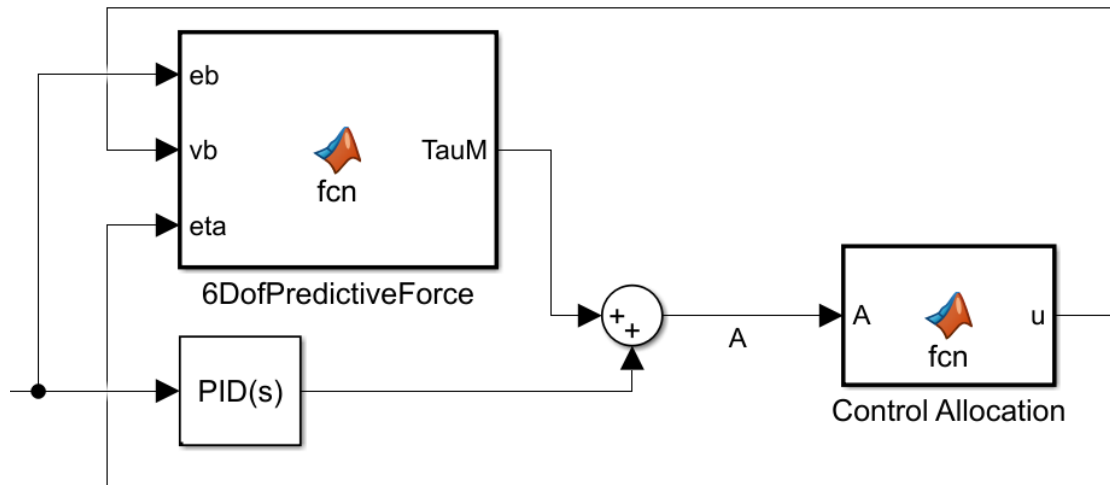
is the tracking error in the body frame.

Since the velocity  $v$  is the time derivative of the position (i.e.  $v = \dot{\eta}$ ), for a defined virtual reference position  $\eta_r$ , the following is satisfied:

$$v_r = \dot{v}_d + \lambda e_b$$

So,  $\lambda$  is used to tune the 6-DoF predictive force.

This is shown in the following block:



Where 'A' is the new controller output.

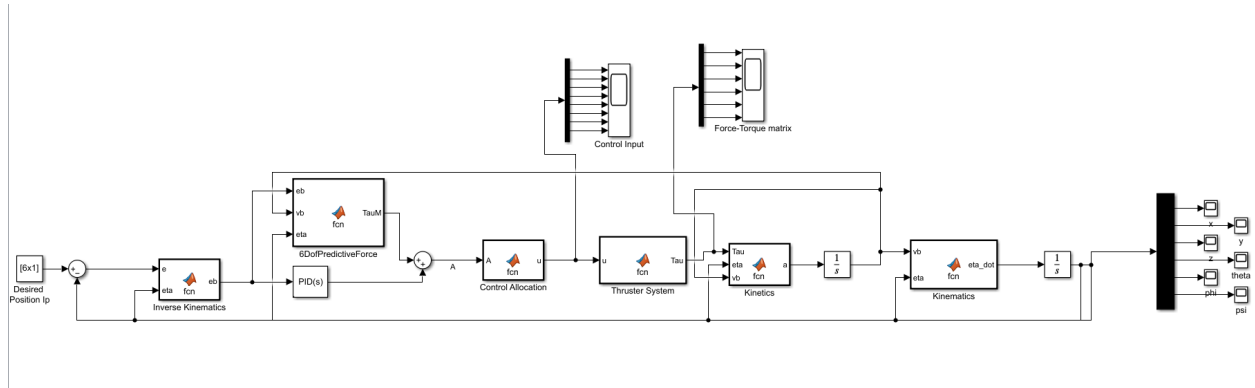
The PID controller gains ( $K_p$ ,  $K_i$  and  $K_d$ ) for the non-linear model are found out by Ziegler Nichols method and were found out to be as follows:

6-DoF PID	Surge	Sway	Heave	Roll	Pitch	Yaw
$K_P$	1	1	1.2	0.3	0.3	0.3
$K_I$	0.001	0.001	0.001	0.001	0.001	0.001
$K_D$	0	0	0	0.3	0.3	0.3

Finally, the control law for the nonlinear model-based PID controller is computed given by:

$$\tau = M(\dot{v}_d + \lambda(v_d - v)) + C_{RB}(v)v + C_A(v_w)v_w + D(v_w)(v_d + \lambda e^b - v_c) + g(\eta) + K_P e^b(t) + K_I \int_0^t e^b(t') dt' + K_D \frac{de^b(t)}{dt}$$

The final model for this system is shown below:



Here, we took desired position input as [1; 1; 2; 0; 0; 0].

## Implementation:

We implemented the models for both the controllers above and got the following results when we give a desired positional input:

### 1) Linear Controller Model:

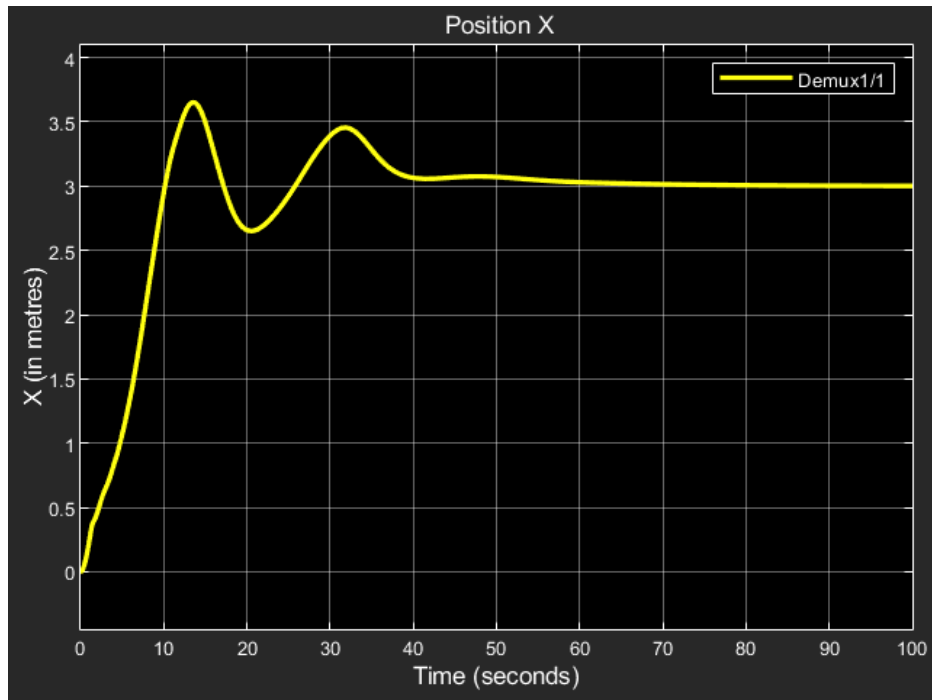
Using the above method, we have obtained the following results for a desired position input:  $\eta_d = [3; 4; 1; 1.57; 0; 0]$ .

The output of the position and orientation control is obtained as follows:

#### Position X :

*Desired output: 3m*

*Obtained result:*



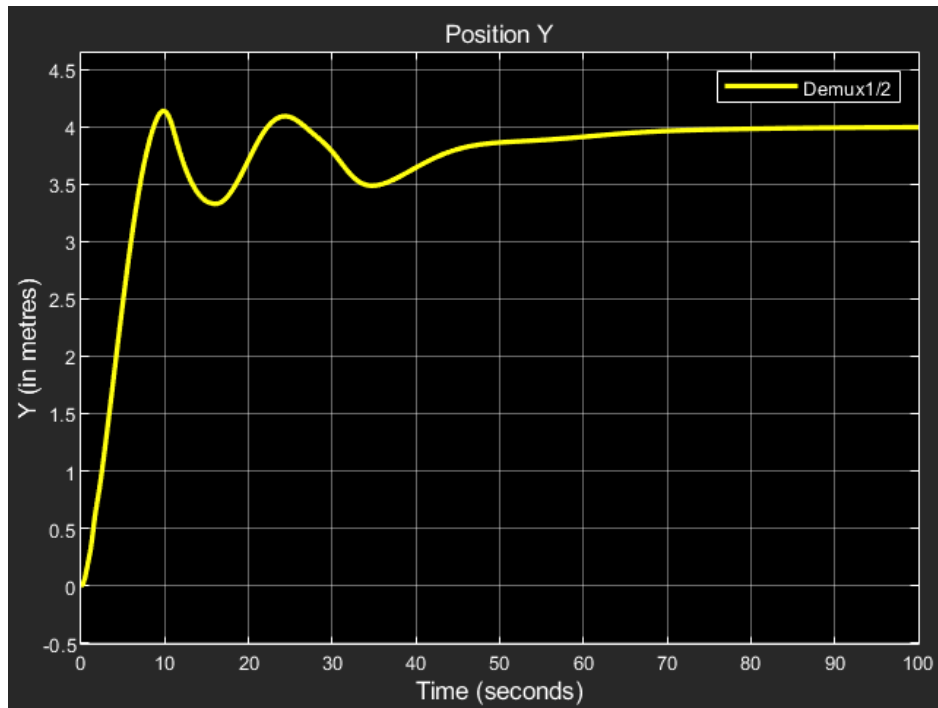
The steady state response final value = 3.

### **Position Y :**

*Desired output: 4m*

*Obtained result:*



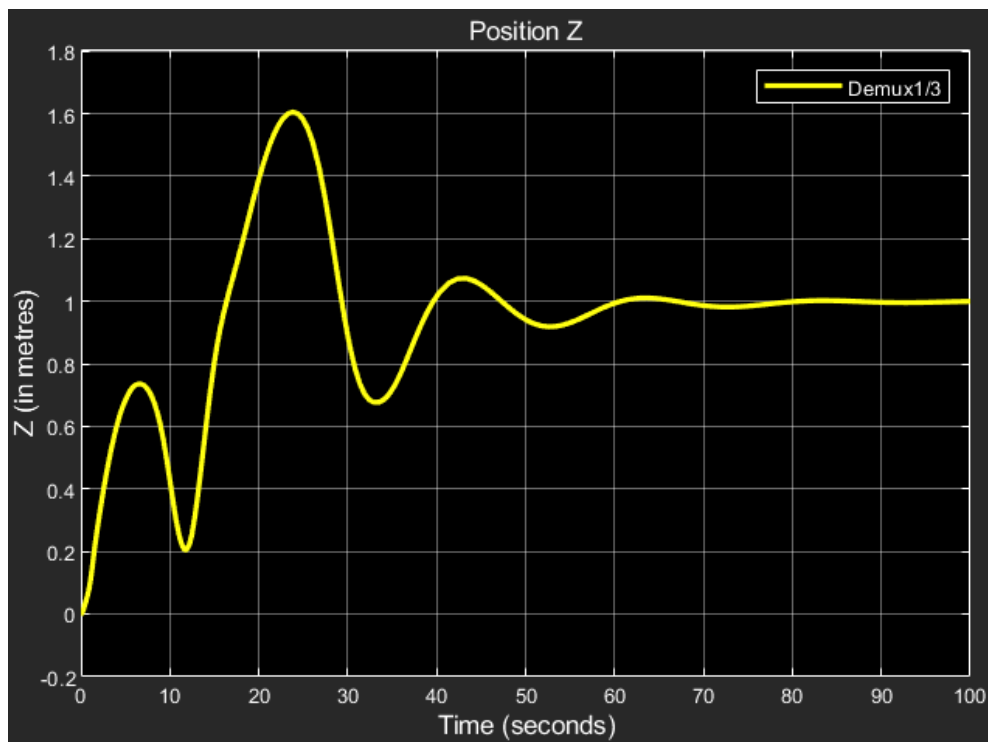


The steady state response final value = 4.

### Position Z :

*Desired output: 1m*

*Obtained result:*

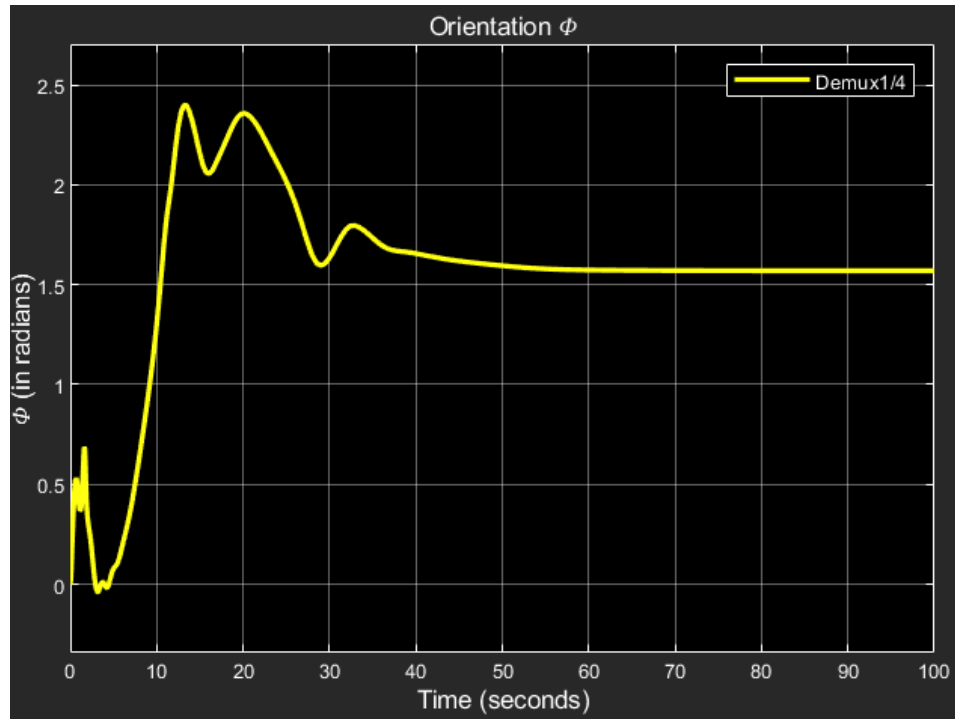


The steady state response final value = 1.

### Orientation $\Phi$ :

*Desired output:* 1.57 radians

*Obtained result:*

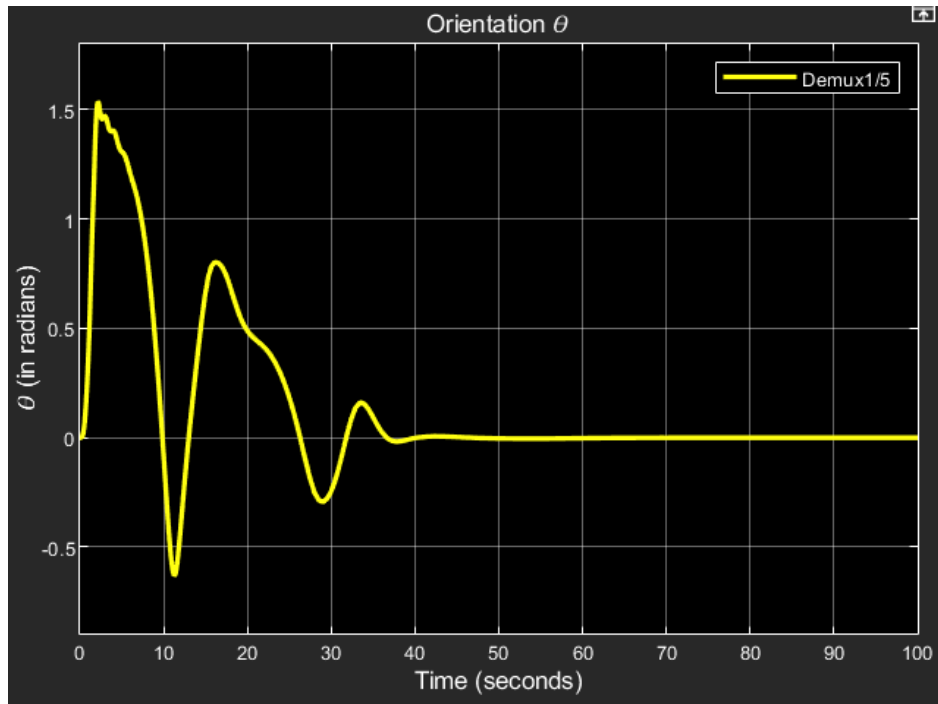


The steady state response final value = 1.57.

### Orientation $\Theta$ :

*Desired output:* 0 radians

*Obtained result:*

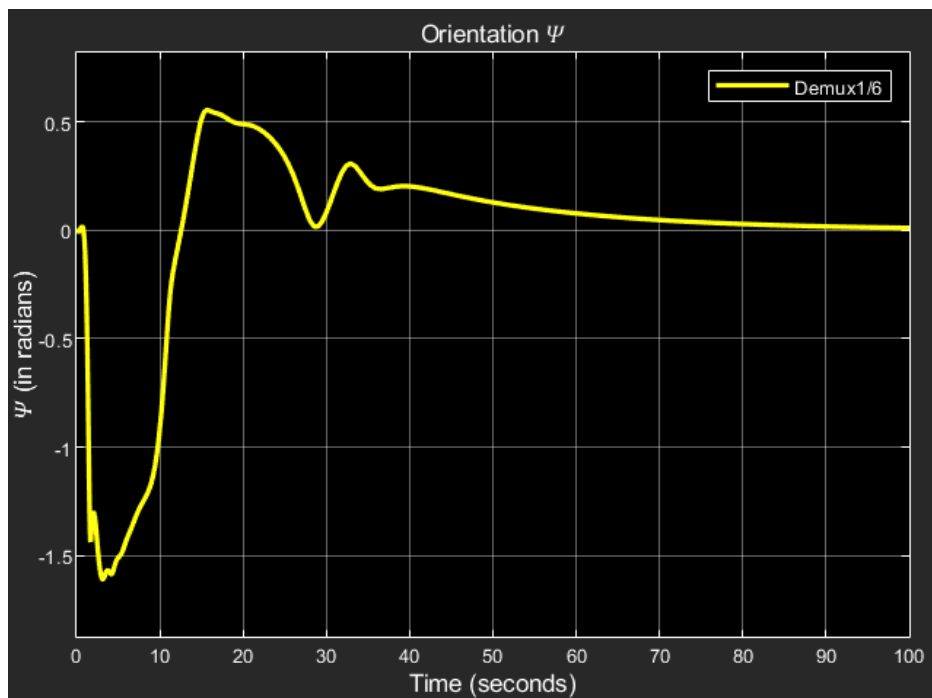


The steady state response final value = 0.

**Orientation  $\psi$  :**

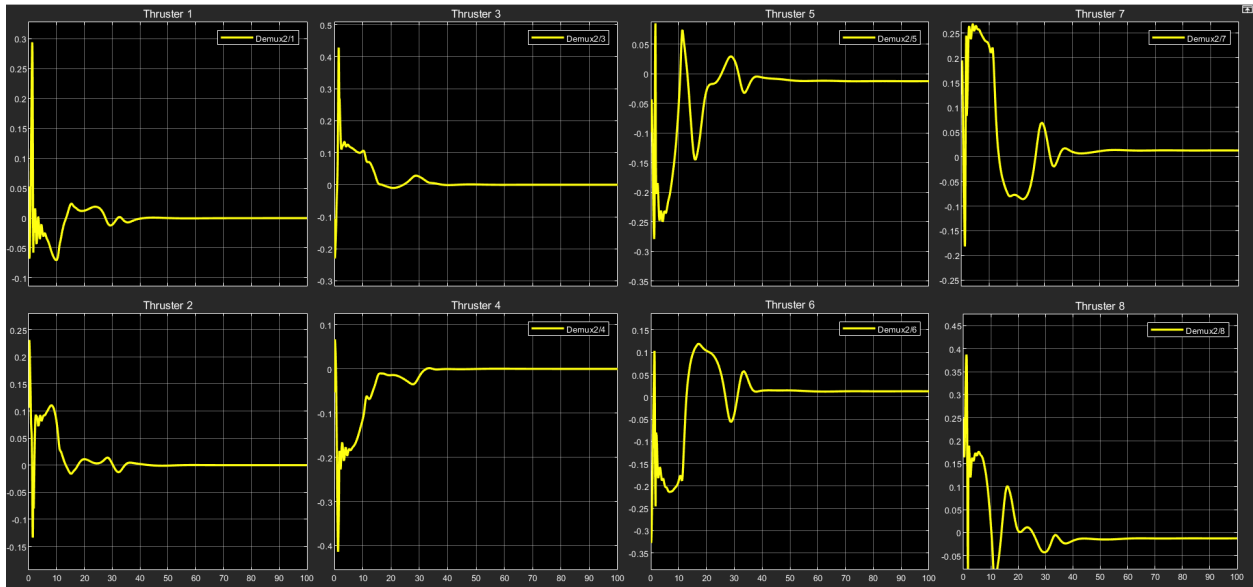
*Desired output:* 0 radians

*Obtained result:*



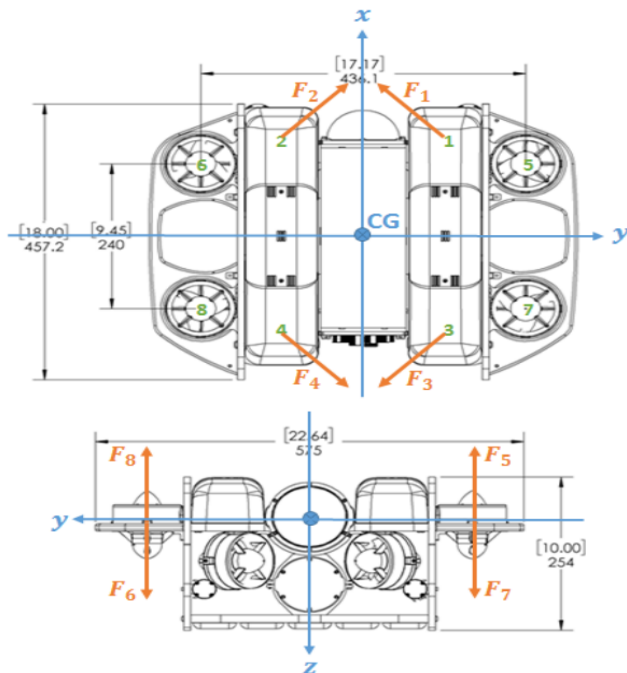
The steady state response final value = 3.

The control inputs to the thrusters for the same are represented in the plots below:

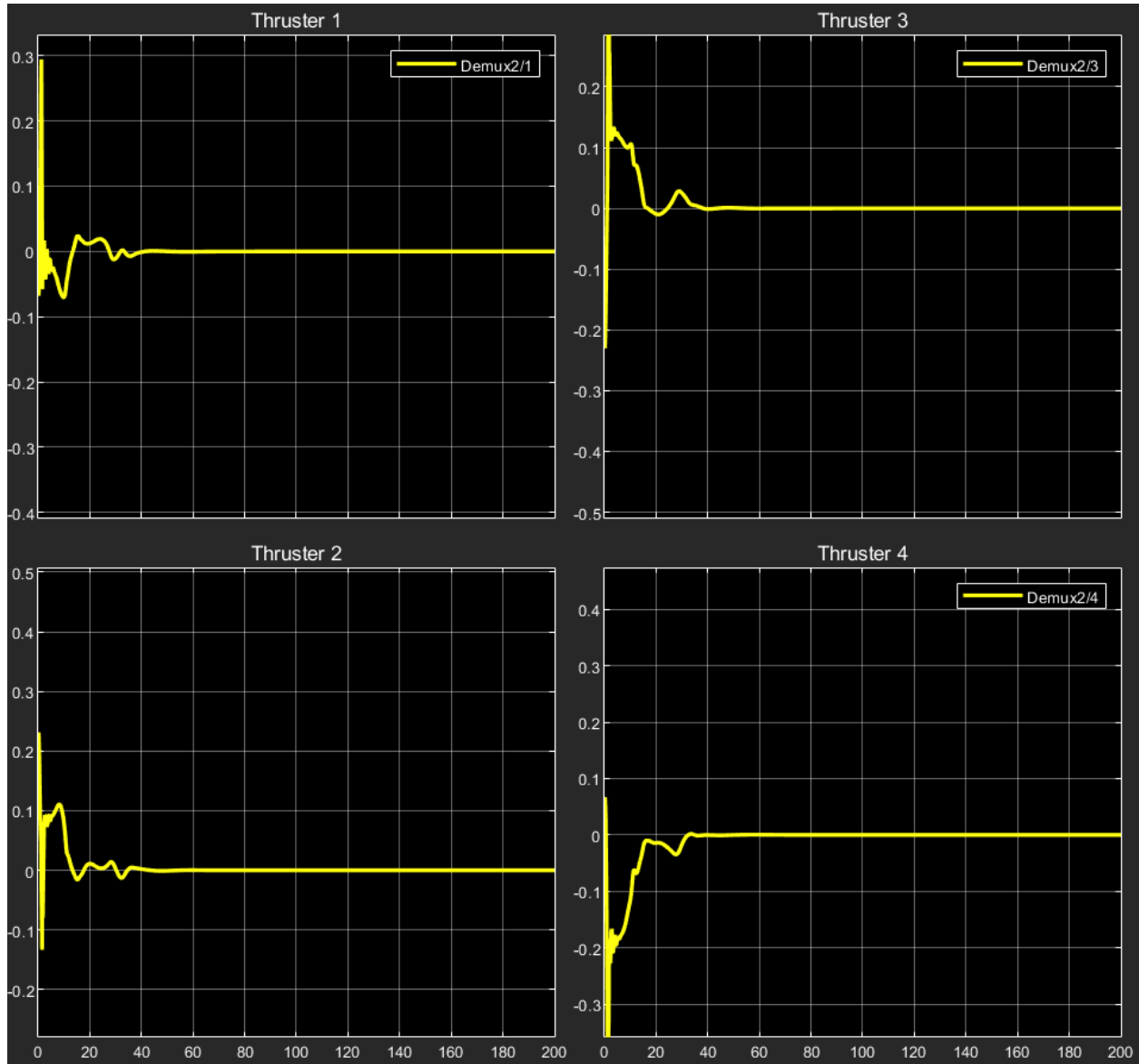


As it is evident from the plots, each thruster is instructed to provide the required thrust every instant for a finite period of time after which the control input eventually settles down to a final value.

The model of the rover under study somewhat looks like this:



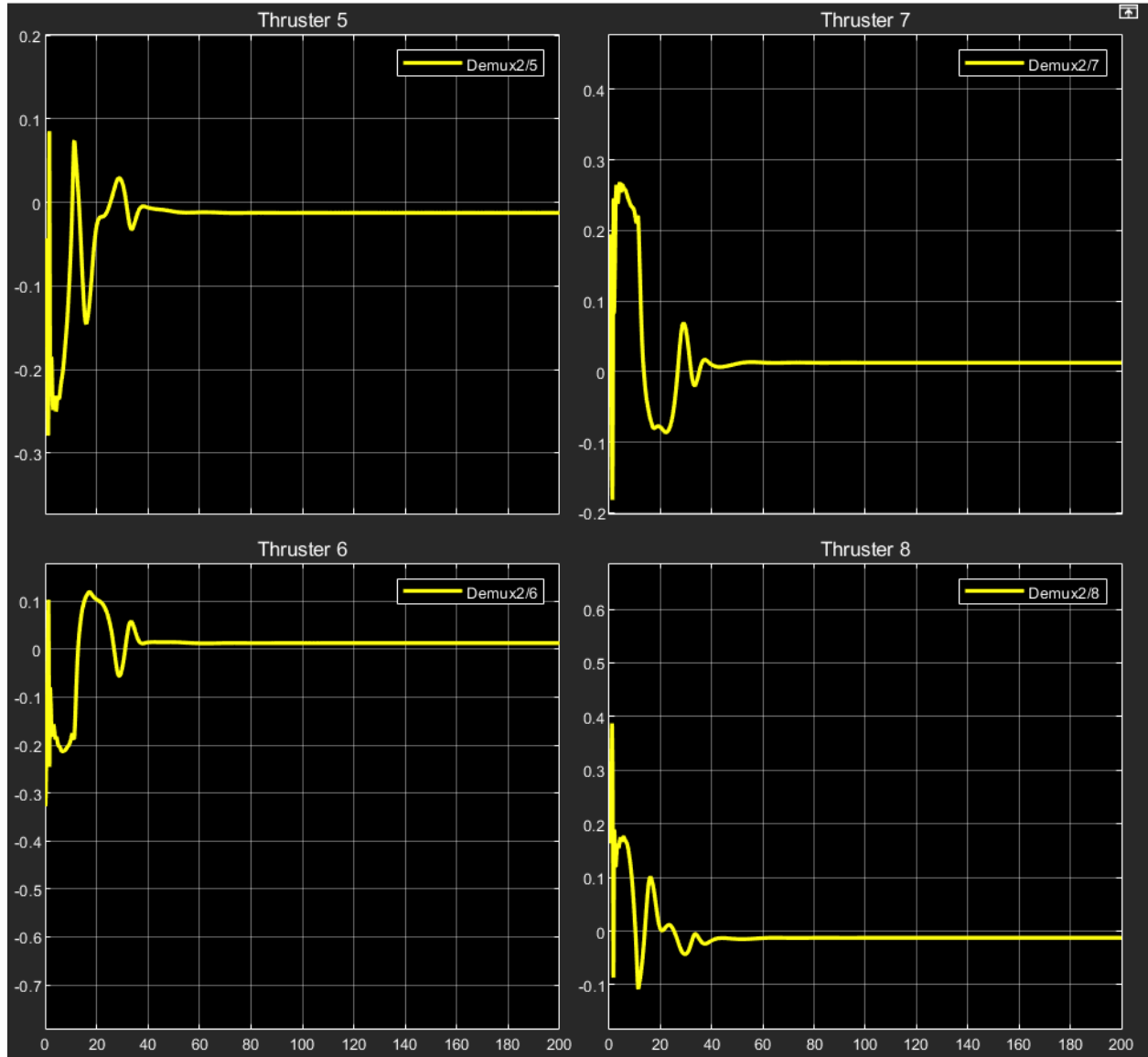
Here, the thrusters 1,2,3,4 are used in the position control in the X,Y directions while 5,6,7,8 are used for the Z-directional control. If we see the plots for the thruster control inputs for 1,2,3,4:



We can observe the steady state final value to be zero for all the thrusters 1,2,3,4.

**Reason:** The above thrusters are used for controlling the position in X,Y directions and their respective control inputs drive them to work in a synchronised manner so as to reach the respective position. Once the rover has reached the required X,Y, coordinates of the position with the required orientation, there is no need for them to provide any more thrust given the lack of any external force acting in the X,Y directions.

The interesting part comes when we observe the control input plots for the thrusters 5,6,7,8:



As it is observed, each of the control inputs have a non-zero steady state value; positive for thrusters 6,7 and negative for thrusters 5,8.

**Reason:** Thrusters 5,6,7,8 are used for positional control in the Z direction. When the rover moves to the desired Z coordinate position in the required orientation, the job of the thrusters is not done since the position has to be retained while neutralising an external force. This external force is the buoyancy acting upon the rover since it is designed to be positively buoyant. The values of forcing acting due to weight and buoyancy for the rover under study are:

$$W = 112.8 \text{ N}$$

$B = 114.8 \text{ N}$  ; this implies that the net external force acting on the rover is 2N upwards.

When we look at thrusters 5,8 ; they provide a thrust vertically upwards when rotated clockwise while thrusters 6,7 produce a thrust vertically downwards for the same. This means that for thrusters 5,8 positive thrust is upwards while negative thrust is downwards. This is opposite in the case of thrusters 6,7 From the plots of control inputs to the above thrusters, the steady state values are as follows:

For thrusters 5,8:  $-1.25 \times 10^{-2}$  (approx.)

For thrusters 6,7:  $+1.25 \times 10^{-2}$  (approx.)

Thrust produced by thrusters 5,8 is negative, which implies that thrust is produced in the vertically downwards direction.

Thrust produced by thrusters 6,7 is positive, which again implies that thrust is produced in the vertically downwards direction.

Total thrust produced  $T = K \cdot u$  where  $K = 40$  for all the thrusters.

Total thrust produced in steady state:  $4 \times 1.25 \times 10^{-2} \times 40 \text{ N} = 2 \text{ N}$  downwards.

Since the external force in Z direction has been neutralised, the rover is stabilised once it reaches the desired position.

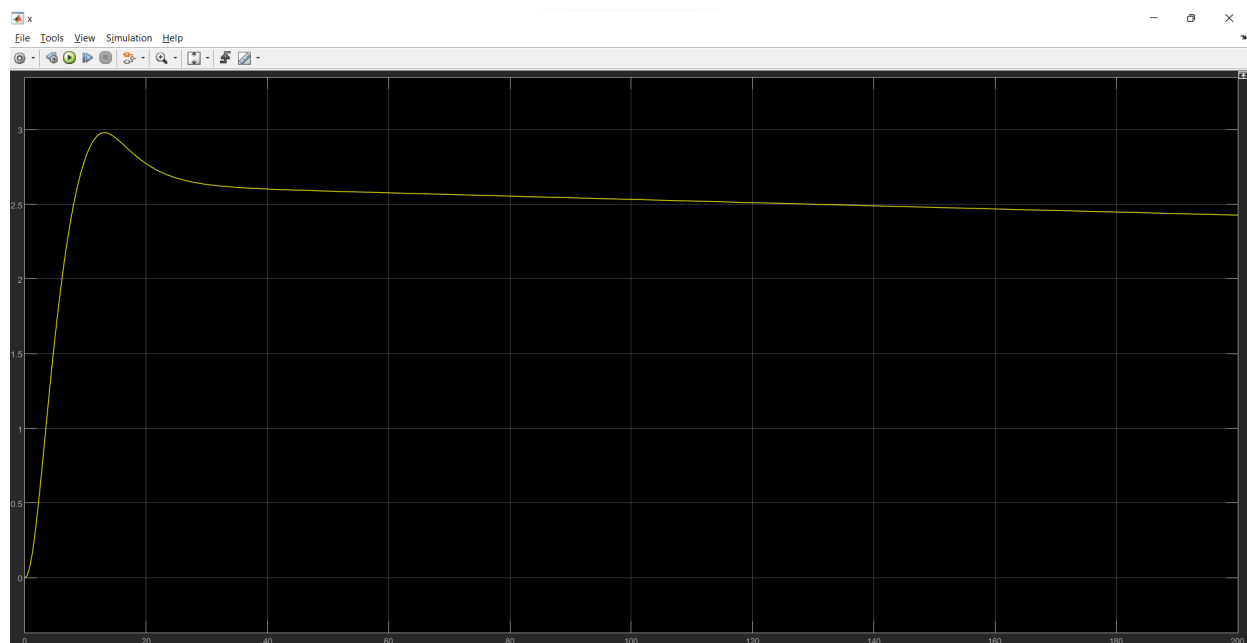
This is how the positional control of the underwater rover has been established using the linear PID control method.

## 2) Non-Linear Controller Model:

For desired positional input as  $[1; 1; 2; 0; 0; 0]$ , we got the following results for positional coordinates in world frame:

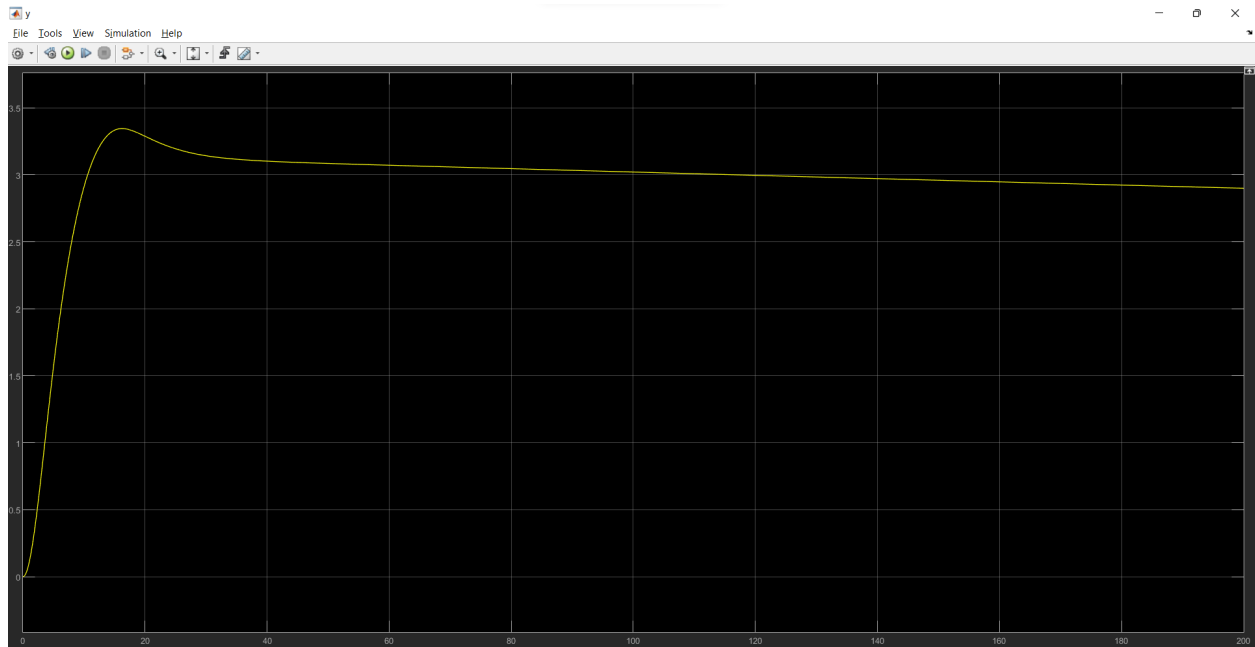
### Position X :

Desired output: 1 m



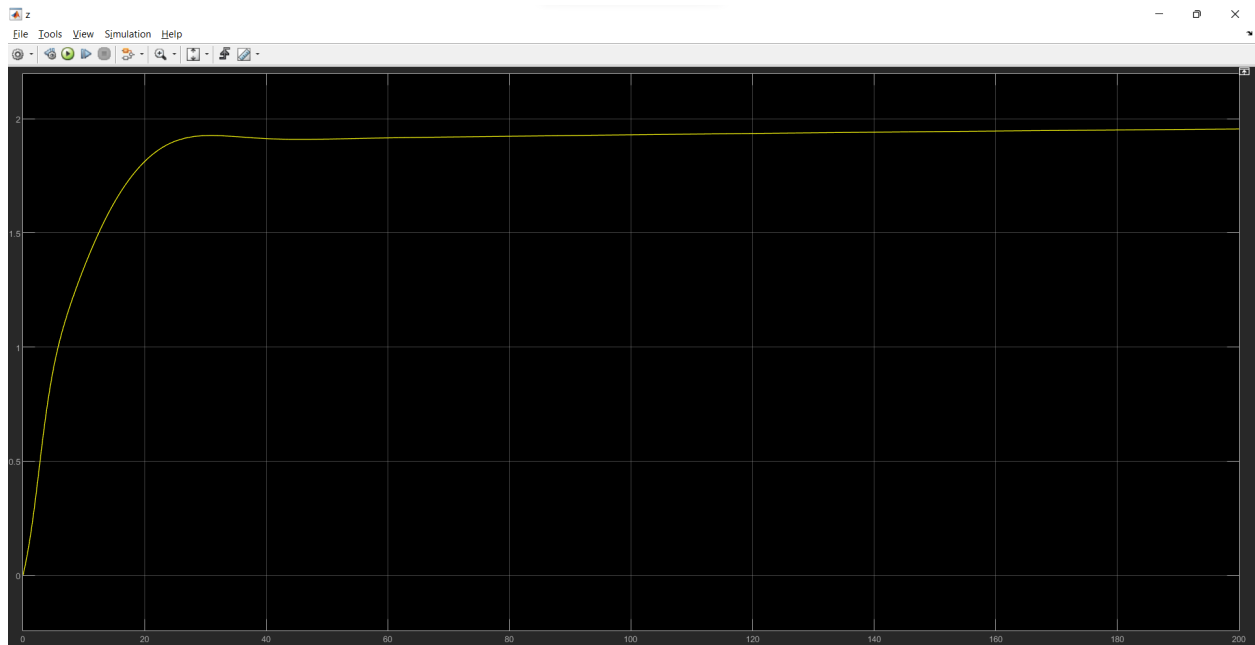
## Position Y :

Desired output: 1 m



## Position Z :

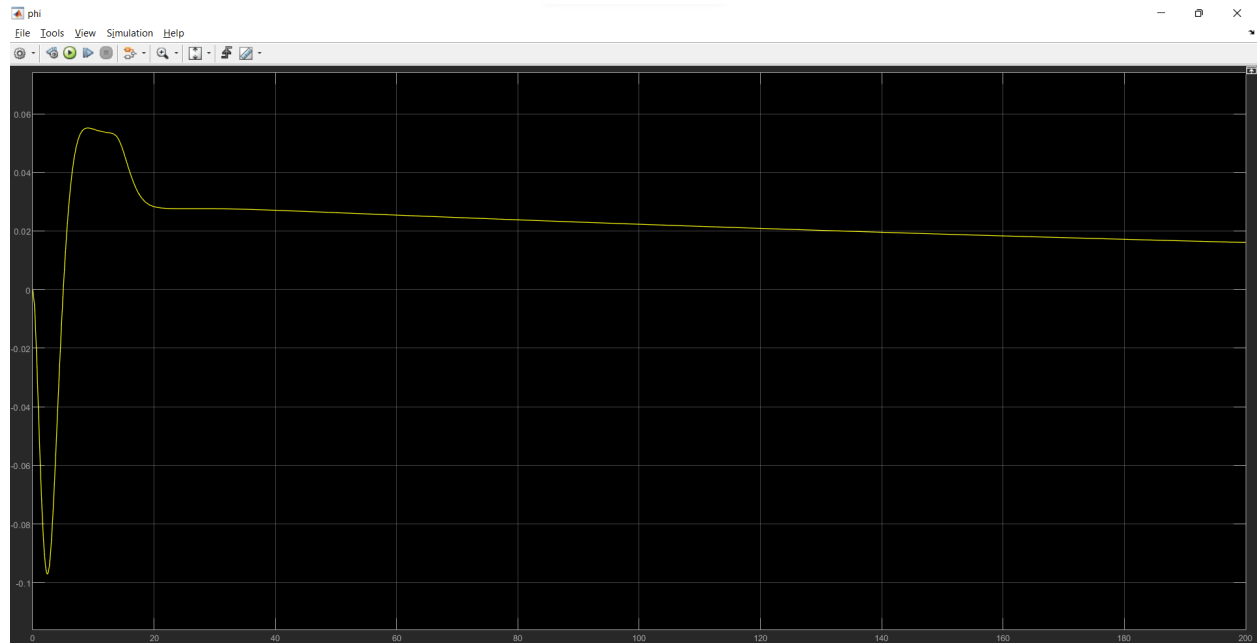
Desired output: 2 m





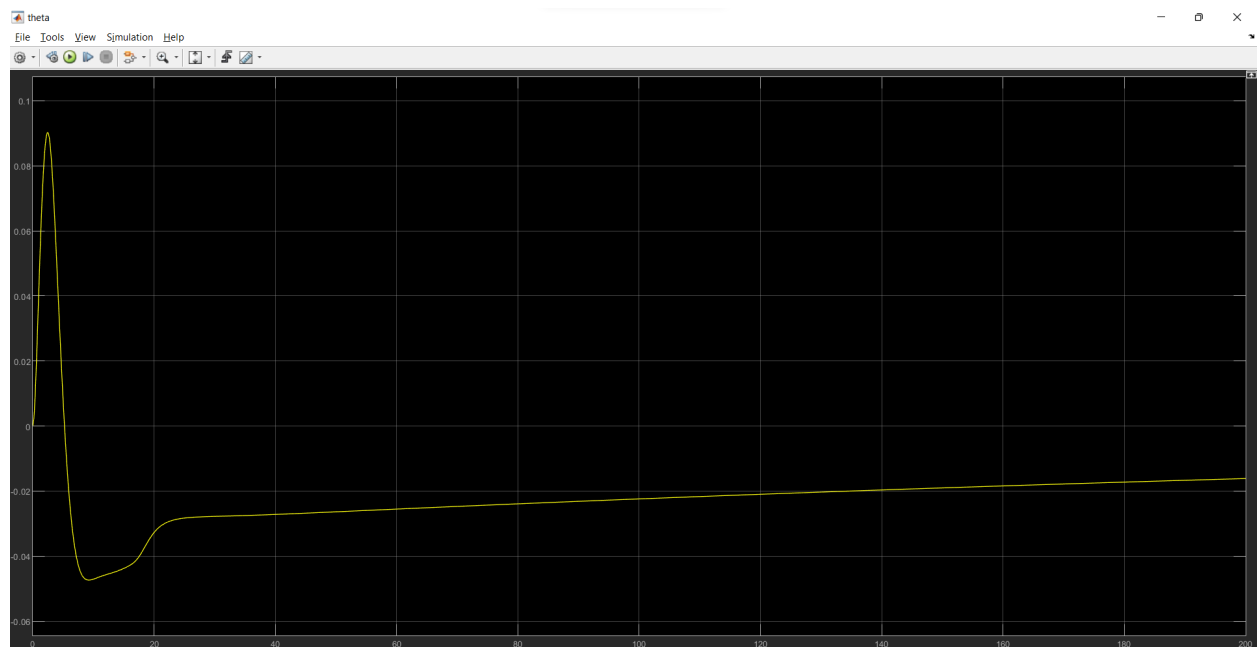
## Orientation $\Phi$ :

Desired output: 0 rad



## Orientation $\theta$ :

Desired output: 0 rad

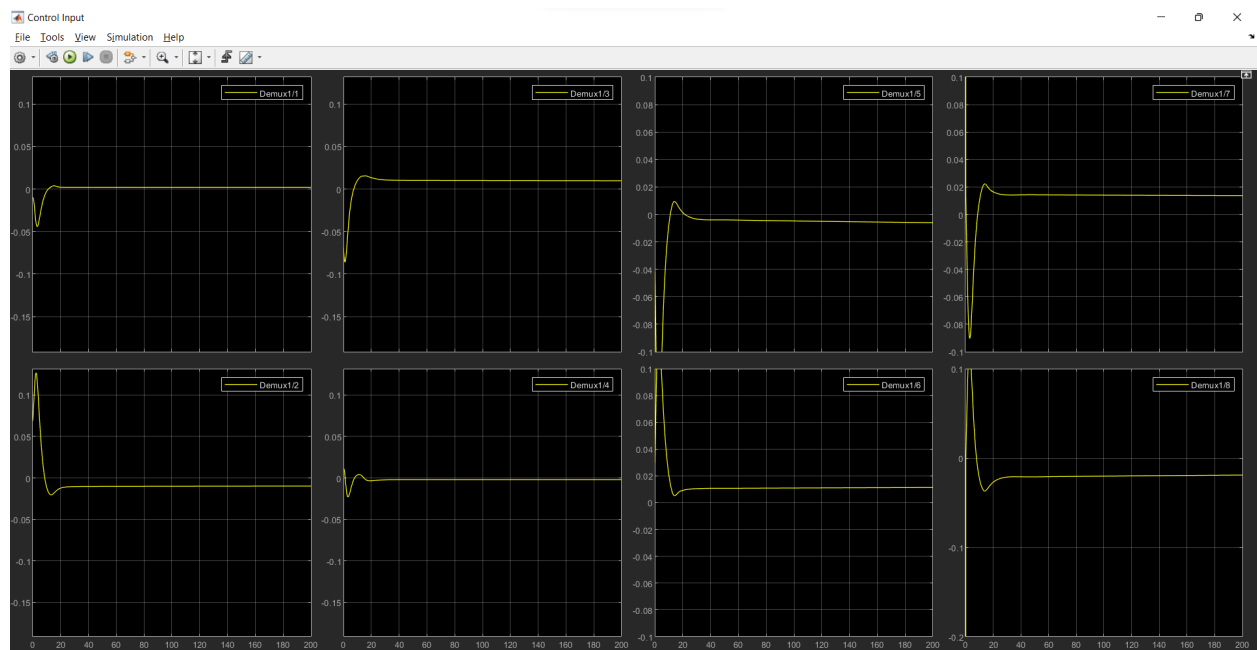


## Orientation $\psi$ :

Desired output: 0 rad



The controller input plots (controller input for each thruster will be as follows):



So, we can see that the thrust control input for thrusters 5,6,7 and 8 will be:  $-1.375e^{-02}$ ,  $1.375e^{-02}$ ,  $1.375e^{-02}$ ,  $-1.375e^{-02}$  respectively.

Now, since propeller pair of 5 and 8 will be opposite to that of the pair 6 and 7, we get the total thrust on the ROV in Z-direction as:

$1.375e^{-02} * 4 * 40 = 2N$  (approx.), where 40 is the gain.  
So, the force in the vertical direction is balanced.

## B) LQR Controller:

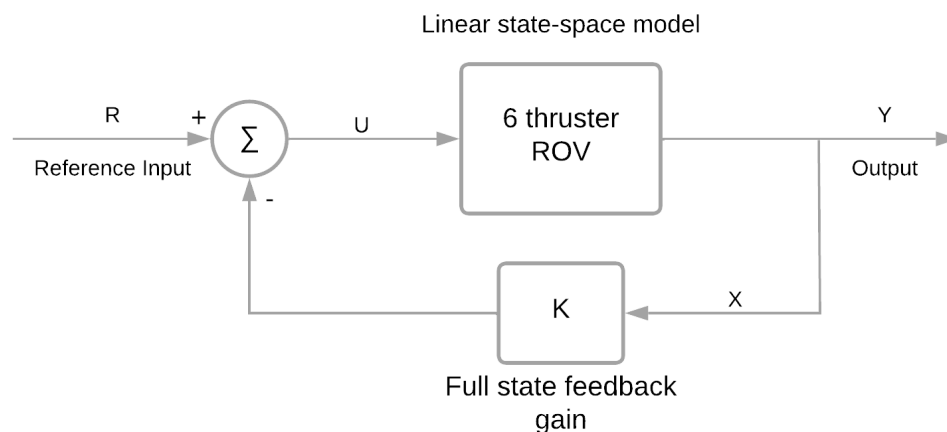
### Literature Review:

The PID controller has been one of the most commonly used controllers for a really long time. There have been numerous PID tuning techniques, such as the Ziegler-Nichols method but are insufficient for high-performance control applications. The Linear Quadratic Regulator (LQR) is an optimal control method based on full state feedback. It aims to minimise the quadratic cost function and is then applied to linear systems, hence the name Linear Quadratic Regulator.

### Why LQR controller?

The use of LQR over PID control comes from the higher robustness of the former in terms of tuning the parameters with varying conditions. PID control uses the error in the input parameter of the closed loop system and tunes the parameters to reduce the error to zero. LQR, on the other hand, uses the state space model of the system and takes complete state feedback:  $-Kx$ , to calculate the error. LQR uses the method of cost function to calculate the control input cost vs the importance of achieving desired states.

### State-Space Model with Full State Feedback Gain:



$$\begin{aligned}\dot{X} &= AX + BU \\ y &= CX + DU\end{aligned}$$

### **Cost Function:**

The cost function defined by the system equations is minimised by the LQR controller using an optimal control algorithm. The cost function involves the system's state parameters and input (control) parameters, along with the Q and R matrices. For the optimal LQR solution, the overall cost function must be as low as possible. The weights given to the state and control parameters are represented by the Q and R matrices, which act as knobs whose values can be varied to adjust the total value of the cost function.

The system must have a linearized state-space model to solve the LQR optimization problem. The cost function to be optimised is given by

$$J = \int (X^T Q X + U^T R U) dt$$

### **Algebraic Riccati Equation and Its Solution(S matrix):**

The Q and R matrices are used to solve the Algebraic Riccati Equation (ARE) to compute the full state feedback matrix.

$$A^T S + SA - SBR^{-1}B^T S + Q = 0$$

On solving the above equation, we obtain the matrix  $S$ .

### **Feedback Gain (K) and Eigen Values from S matrix:**

The matrix  $S$  obtained from the above ARE is used to find the full state feedback gain matrix  $K$  using the relation,

$$K = R^{-1}B^T S$$

The control matrix  $U$  is then given by

$$U = -KX$$

## Linearisation of a state-space model:

The linearization of a state-space model is needed while using the LQR technique since it works on linear systems. In our case, the state space model is in the form:  $\dot{x} = f(x)$ ; where  $f(x)$  is a nonlinear function of  $x$ . In such cases, to linearise the equation, we use the concept of linearising about a fixed point. The steps for the same are as follows:

1. Find the fixed points  $\bar{x}$ ; where  $f(\bar{x}) = 0$ .
2. Linearise about an  $\bar{x}$ , by calculating the Jacobian of dynamics at the fixed point  $\bar{x}$ ; where the latter can be represented as:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^T f_1 \\ \vdots \\ \nabla^T f_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

where each term is a partial derivative of the dynamics with respect to a variable.

This step is executed since the dynamics of a nonlinear system behave linearly at the fixed point, or, in a small neighborhood around the fixed point.

Changing the frame of reference to one with  $\bar{x}$  as the origin:

$$\begin{aligned} \dot{x} - \dot{\bar{x}} &= f(x) \\ &= f(\bar{x}) + J \cdot (x - \bar{x}) + J^2 \cdot (x - \bar{x})^2 + \dots \end{aligned}$$

where  $J$  is calculated at  $\bar{x}$ .

The higher order terms from the third term  $J^2 \cdot (x - \bar{x})^2$  are neglected since they are really small, hence the equation reduces to:

$$\begin{aligned} \Delta \dot{x} &= 0 + J \cdot \Delta x + 0 \\ \Rightarrow \Delta \dot{x} &= J \cdot \Delta x \end{aligned}$$

This is in the form  $\dot{x} = Ax$

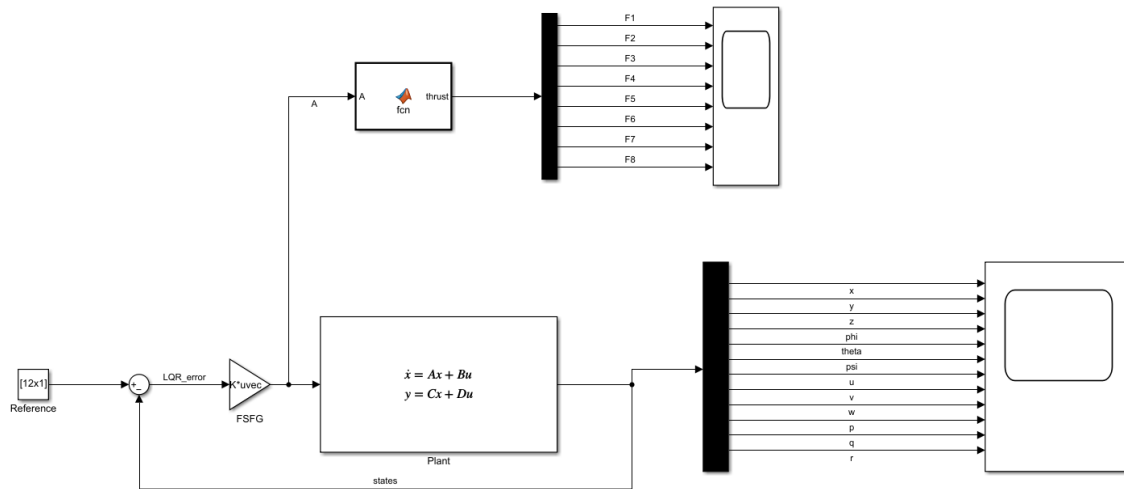
Note: The above method for linearization works only when the fixed point satisfies the condition for linearising a system given by the Hartman Grobman Theorem, which states that:

*“the behaviour of a dynamical system in a domain near a hyperbolic equilibrium point is qualitatively the same as the behaviour of its [linearisation](#) near this equilibrium point, where hyperbolicity means that no eigenvalue of the linearisation has real part equal to zero. Therefore, when dealing with such dynamical systems one can use the simpler linearisation of the system to analyse its behaviour around equilibria.”<sup>[1]</sup>*

Hence, linearization works only when the fixed point is hyperbolic or put simply, has a non-zero real part.

## Implementation:

The following is the Simulink model that we built:



To implement this, we need the following:

1) State-space model:

- a) A and B matrices: These relate the derivative of the states with the current states and the control input. But, since our model is non-linear in nature, we cannot get a direct relation consisting of A and B matrices. Also, the LQR controller works only for Linear Systems. So, we have to linearise our system around an operating point. For our system, we took the origin in the world frame (initial point of the bot) as the operating point. Also, we used the position of the bot in the world frame and velocity of the bot in its own frame as the two states. Meaning:

$$x = \begin{bmatrix} \eta \\ v \end{bmatrix}$$

(Here x is the state but not the distance in x direction)

$$\begin{aligned}\dot{\mathbf{x}} &= \begin{bmatrix} \dot{\boldsymbol{\eta}} \\ \dot{\mathbf{v}} \end{bmatrix} = f(\mathbf{x}, \mathbf{u}, \boldsymbol{\tau}_d, t) \\ &= \begin{bmatrix} J(\boldsymbol{\eta})\mathbf{v} \\ M^{-1}[Kp(A\mathbf{u}) + \boldsymbol{\tau}_d - C(\mathbf{v})\mathbf{v} - D(\mathbf{v})\mathbf{v} - g(\boldsymbol{\eta})] \end{bmatrix}\end{aligned}$$

Since,

$$\dot{\boldsymbol{\eta}} = J(\boldsymbol{\eta})\mathbf{v}$$

The above state space model is clearly non-linear. So, we have to linearise to find A and B as follows:

$$A(t) \equiv \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_0$$

$$B(t) \equiv \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \dots & \frac{\partial f_1}{\partial u_m} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \dots & \frac{\partial f_2}{\partial u_m} \\ \vdots & & & \\ \frac{\partial f_n}{\partial u_1} & \frac{\partial f_n}{\partial u_2} & \dots & \frac{\partial f_n}{\partial u_m} \end{bmatrix}_0$$

So, we obtain the A and B as follows:

A =

0	0	0	0	0	0	1.0000	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1.0000	0	0	0	0
0	0	0	0	0	0	0	0	0	1.0000	0	0	0
0	0	0	0	0	0	0	0	0	0	1.0000	0	0
0	0	0	0	0	0	0	0	0	0	0	1.0000	0
0	0	0	0	0	0	0	0	0	0	0	0	1.0000
0	0	0	0	1.1784	0	-0.3608	0	0	0	0.0090	0	0
0	0	0	-1.1784	0	0	0	-0.5569	0	-0.0090	0	0	0
0	0	0	0	0	0	0	0	-0.4504	0	0	0	0
0	0	0	-15.4440	0	0	0	-0.8005	0	-0.4505	0	0	0
0	0	0	0	-15.4440	0	0.5187	0	0	0	-0.4505	0	0
0	0	0	0	0	0	0	0	0	0	0	0	-0.4375

B =

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0.0895	0	0	0	0	-0.1287	0
0	0.0895	0	0.1287	0	0	0
0	0	0.0870	0	0	0	0
0	0.1287	0	6.4350	0	0	0
-0.1287	0	0	0	6.4350	0	0
0	0	0	0	0	0	6.2500

But we obtained the above results by taking weight W and buoyancy B as 110 N and 120 N respectively.

- b) We take the C matrix in state-space ( $y=CX+DU$ ) as  $\text{eye}(12)$  so as to send back all the states as feedback to the summing point.
- 2) Q and R matrices:
- a) Q stands for the importance of the states to reach their desired values.
  - b) R stands for the cost of the control input.
  - c) So, if we have an expensive control input compared to that of our needs to reach the states, we take the Q matrix to be dominating compared to R and vice versa.



- 3) Then, we calculate the full-state feedback matrix, the Algebraic Riccati Equation and the eigenvalues by using the following command:

$$[K,S,P] = \text{lqr}(A,B,Q,R)$$

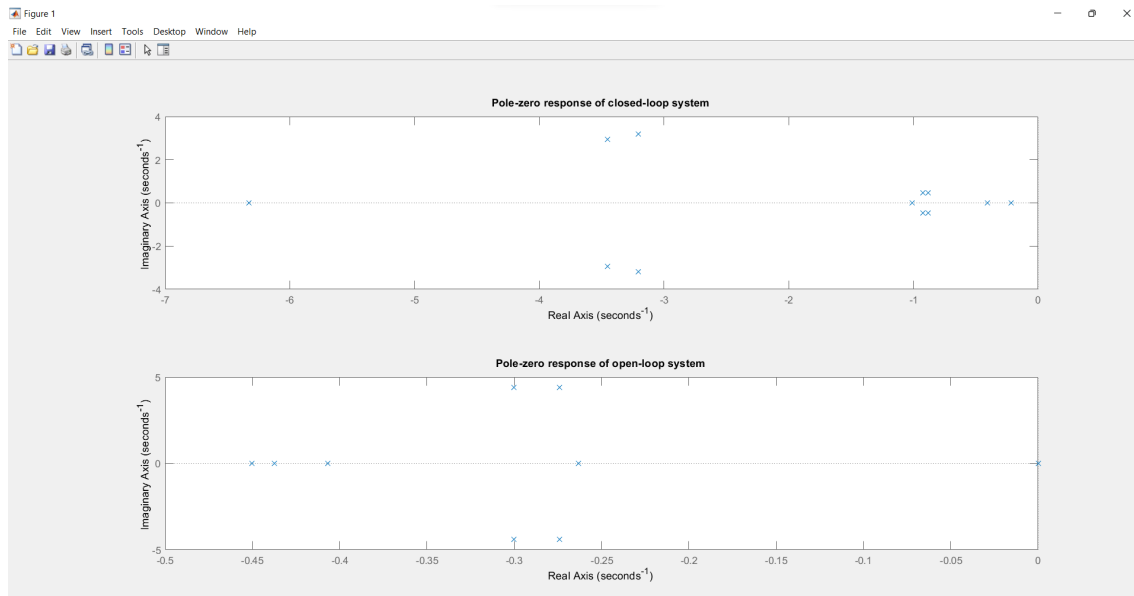
- 4) Then, we put the obtained value of the Gain matrix in the state-space model in Simulink. The following results are obtained for different R matrices (Q matrix being a 12x12 identity matrix) and desired states being [1;2;1;0;0;0;0;0;0;0;0;0]:

$$R = \begin{bmatrix} 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 1 \end{bmatrix}$$

K =

0.2257	0.0466	-0.0694	-0.0395	0.1086	-0.0778	0.3981	0.0775	-0.1335	-0.0820	-0.0690	-0.0835
-0.1061	0.3582	-0.0989	-0.1746	-0.0732	-0.0777	-0.2105	0.5498	-0.1950	-0.0917	-0.0869	-0.0844
-0.0841	0.0426	1.0110	-0.0416	-0.0681	-0.0769	-0.1525	0.0686	1.9685	-0.0807	-0.0832	-0.0822
-0.0616	-0.9758	-0.0392	0.7201	-0.0528	-0.0415	-0.1152	-1.5787	-0.0741	1.0806	-0.0440	-0.0449
0.9717	0.0209	0.0057	-0.0482	0.8252	-0.0402	1.9287	0.0292	0.0182	-0.0403	1.1059	-0.0417
-0.0315	0.0317	0.0000	-0.0998	-0.0997	1.0173	-0.0579	0.0474	0.0007	-0.0431	-0.0422	1.0986

For the above, the Poles of the closed loop and open loop transfer function are as follows:



**Poles:**

Closed loop(With the feedback):

```
>> P
```

```
P =
```

```
-0.2170 + 0.0000i  
-0.4043 + 0.0000i  
-0.8836 + 0.4709i  
-0.8836 - 0.4709i  
-0.9222 + 0.4604i  
-0.9222 - 0.4604i  
-1.0104 + 0.0000i  
-3.2047 + 3.1841i  
-3.2047 - 3.1841i  
-3.4503 + 2.9329i  
-3.4503 - 2.9329i  
-6.3274 + 0.0000i
```

Open loop(Without the feedback):

```
>> eig(A)
```

```
ans =
```

```
0.0000 + 0.0000i  
0.0000 + 0.0000i  
-0.2740 + 4.3867i  
-0.2740 - 4.3867i  
-0.2633 + 0.0000i  
-0.3003 + 4.3834i  
-0.3003 - 4.3834i  
-0.4067 + 0.0000i  
0.0000 + 0.0000i  
0.0000 + 0.0000i  
-0.4504 + 0.0000i  
-0.4375 + 0.0000i
```

Making 'R' more dominant

$$R = \begin{bmatrix} 3 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 \\ 1.1 & 3 & 1.1 & 1.1 & 1.1 & 1.1 \\ 1.1 & 1.1 & 3 & 1.1 & 1.1 & 1.1 \\ 1.1 & 1.1 & 1.1 & 3 & 1.1 & 1.1 \\ 1.1 & 1.1 & 1.1 & 1.1 & 3 & 1.1 \\ 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 3 \end{bmatrix}$$

K =

0.0519	0.0773	-0.1269	-0.0227	0.0272	-0.1151	0.0832	0.1343	-0.2570	-0.1199	-0.1145	-0.1280
-0.1688	0.2815	-0.1849	-0.0839	-0.0669	-0.1156	-0.3654	0.4644	-0.3807	-0.1283	-0.1308	-0.1308
-0.1273	0.0681	0.6515	-0.0247	-0.0599	-0.1137	-0.2502	0.1147	1.3169	-0.1175	-0.1230	-0.1257
-0.0964	-0.6480	-0.0667	0.3660	-0.0498	-0.0664	-0.1907	-1.0835	-0.1311	0.6980	-0.0657	-0.0749
0.6517	0.0248	0.0103	-0.0423	0.4201	-0.0634	1.3778	0.0337	0.0314	-0.0577	0.7219	-0.0681
-0.0370	0.0386	0.0018	-0.1623	-0.1600	0.6746	-0.0699	0.0576	0.0044	-0.0652	-0.0637	0.7528

### Making 'R' less dominant

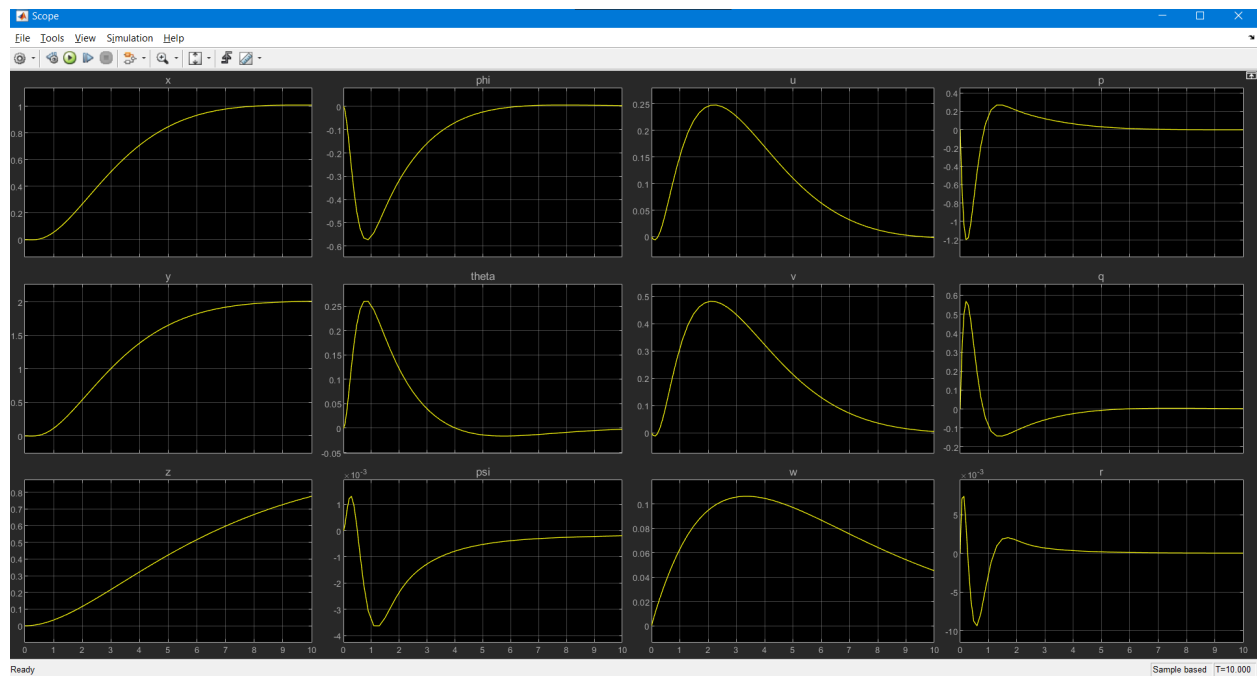
$$R = \begin{bmatrix} 0.11 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.11 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.11 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.11 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.11 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.11 \end{bmatrix}$$

K =

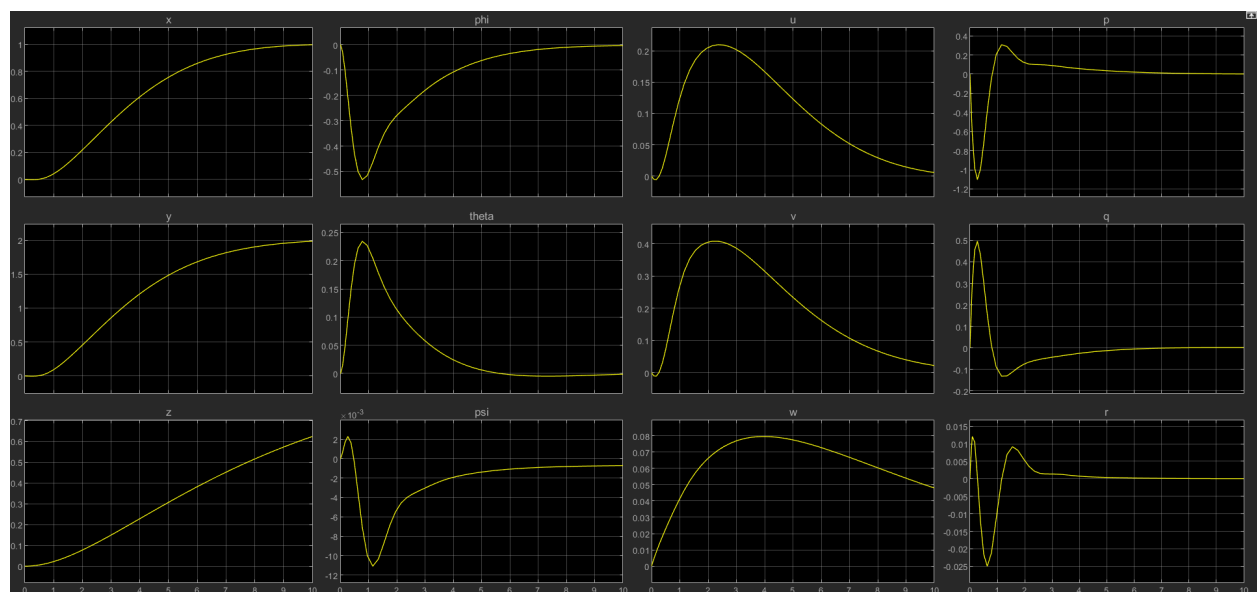
1.5573	0.0626	-0.1795	-0.1972	0.9310	-0.2184	2.4229	0.1125	-0.2899	-0.2289	-0.1793	-0.2237
-0.3114	1.8036	-0.2369	-1.2401	-0.3650	-0.2177	-0.5373	2.4914	-0.4009	-0.2641	-0.2370	-0.2240
-0.2545	0.0702	3.0405	-0.2219	-0.3307	-0.2166	-0.4097	0.1202	5.1156	-0.2275	-0.2299	-0.2209
-0.1677	-2.5885	-0.1324	3.6238	-0.2158	-0.1032	-0.3032	-3.9273	-0.2366	3.2398	-0.1149	-0.1062
2.5139	0.0842	0.0518	-0.2122	3.8528	-0.1015	4.5801	0.1225	0.1094	-0.1100	3.2729	-0.1008
-0.0853	0.0878	-0.0001	-0.2061	-0.2059	3.0582	-0.1486	0.1297	0.0016	-0.1130	-0.1095	3.1451

### **Results:**

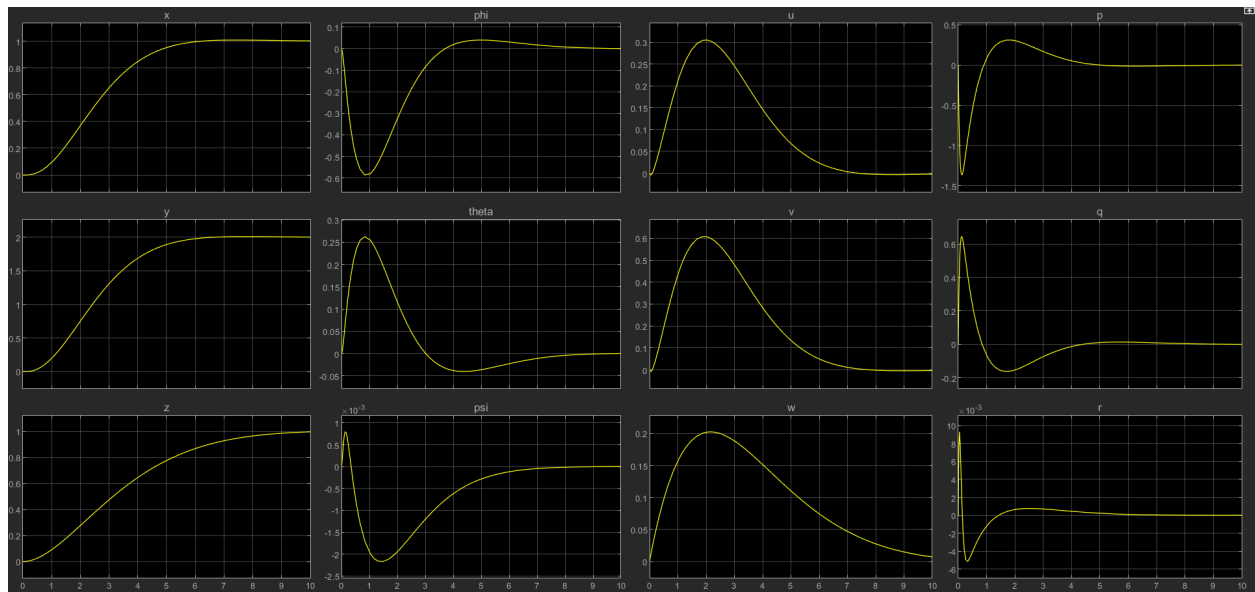
a) Observed states:



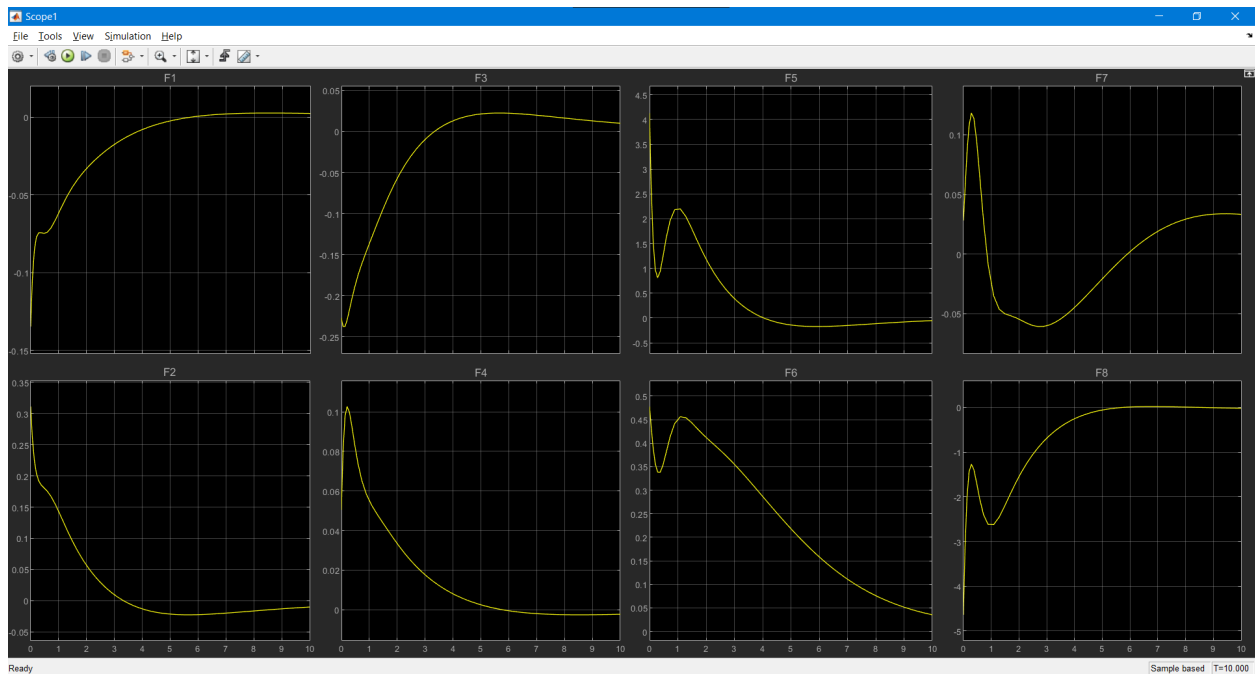
## More dominant 'R'



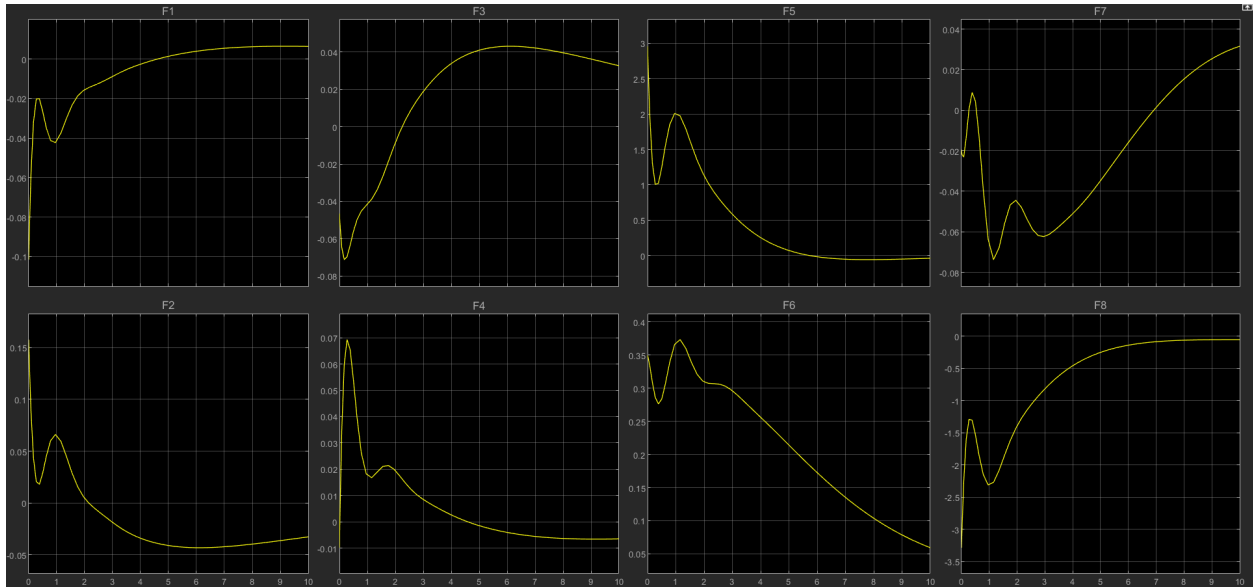
## Less dominant 'R'



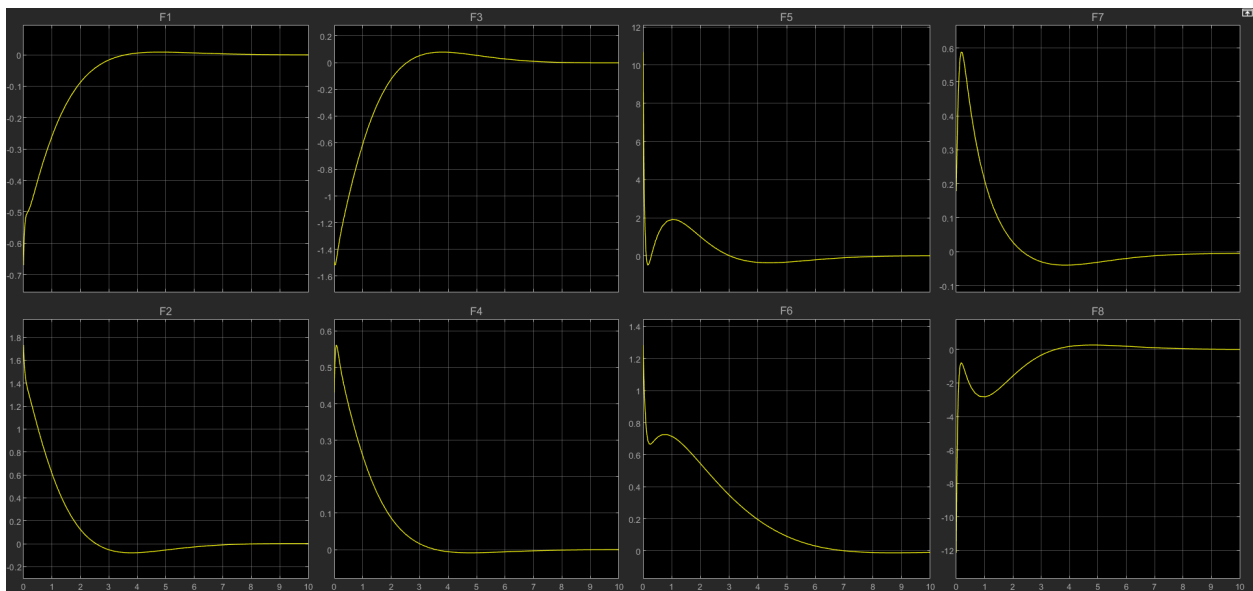
b) Thrust i/p to the thrusters:



## More dominant 'R'



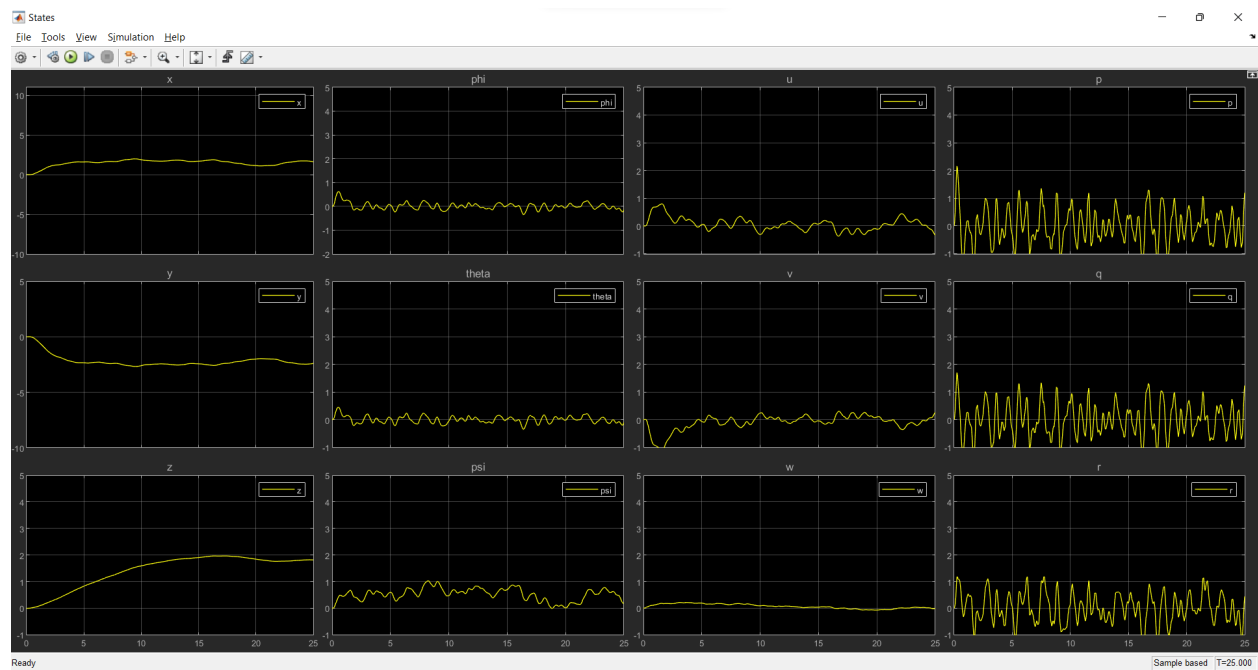
Less dominant ‘R’



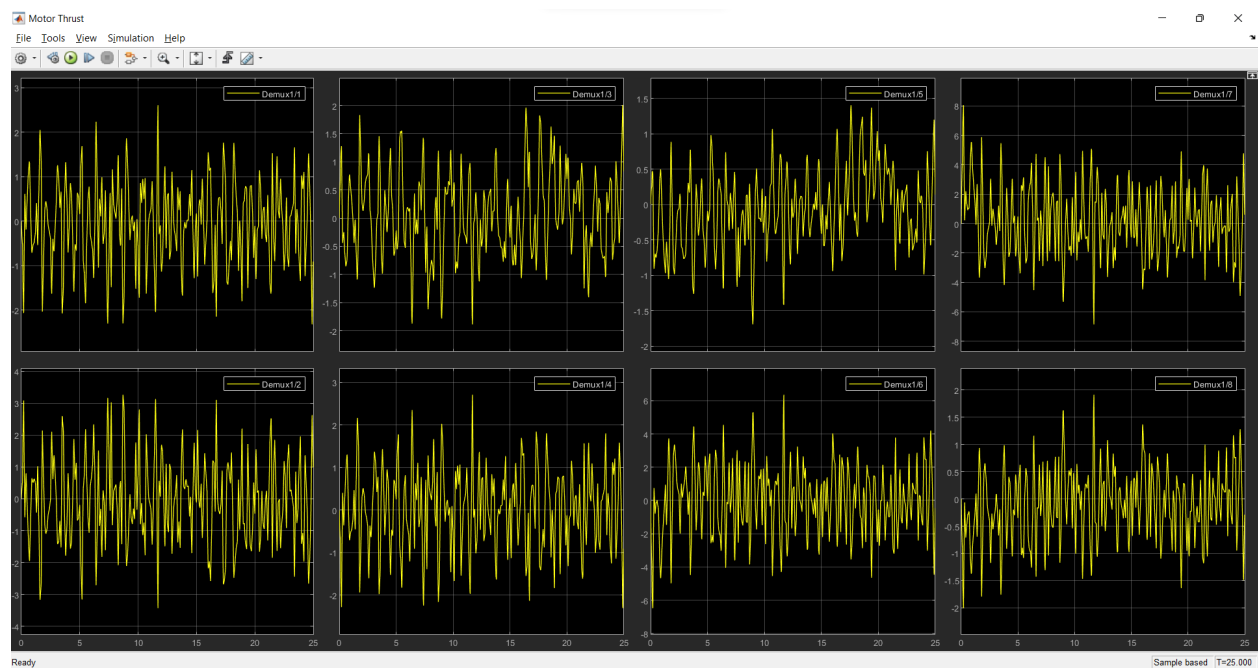
## Band Limited White Noise:

We planned on modelling a system with White Noise. So, we used the following control system:



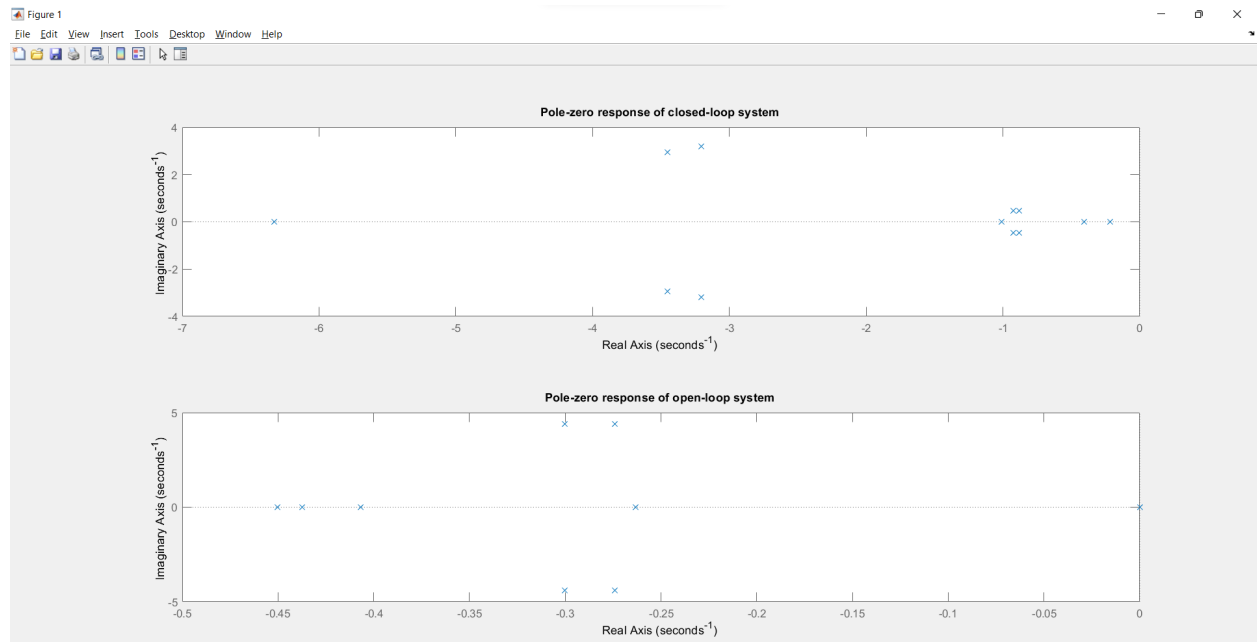


Thrust provided by each thruster:



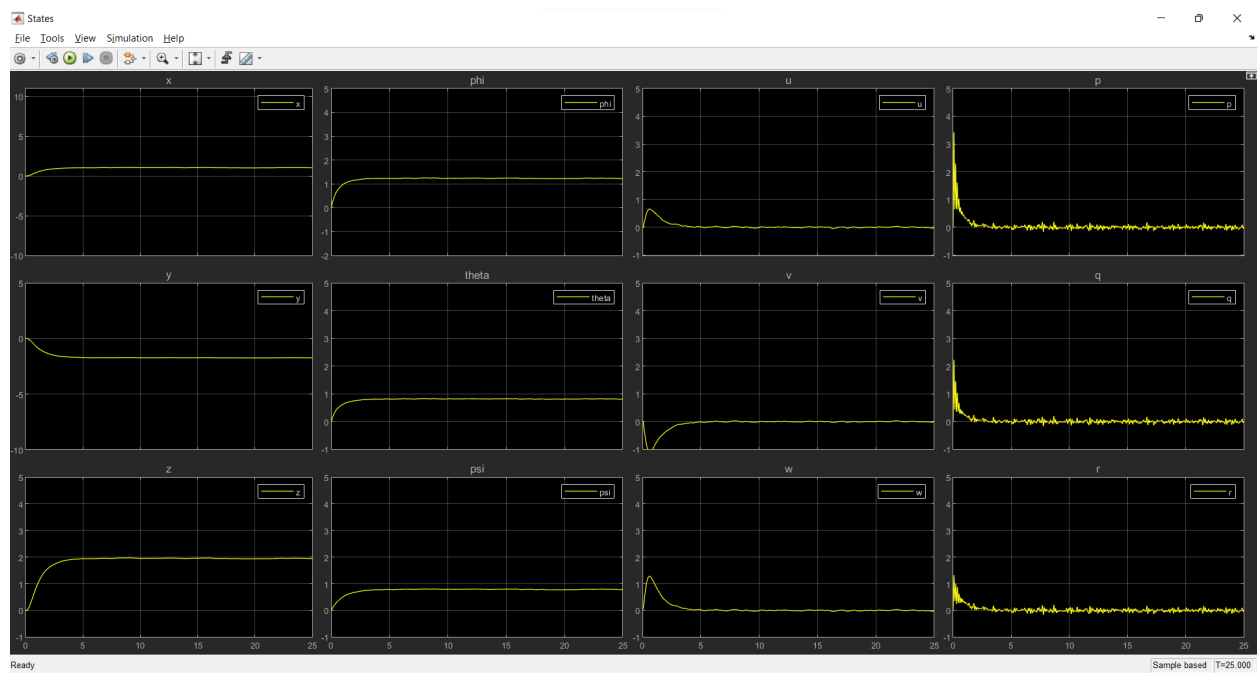
Poles of this system:



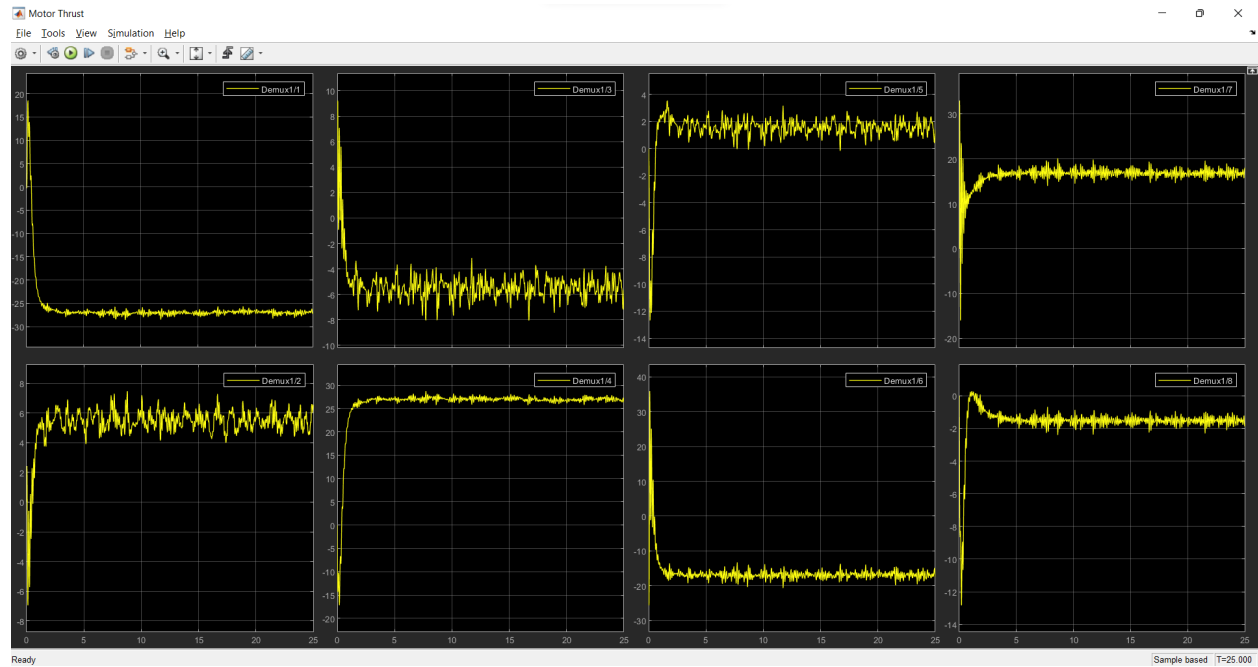


2)  $Q = 1000 \cdot \text{eye}(12)$ :

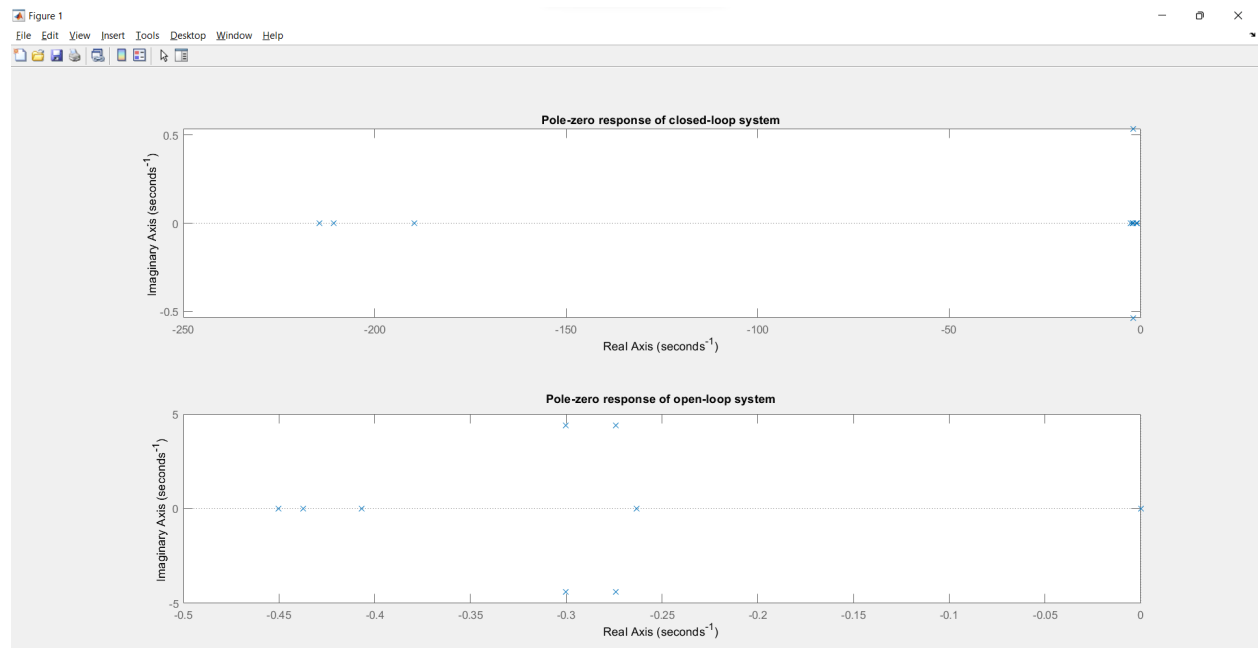
States:



Thrust provided by each thruster:



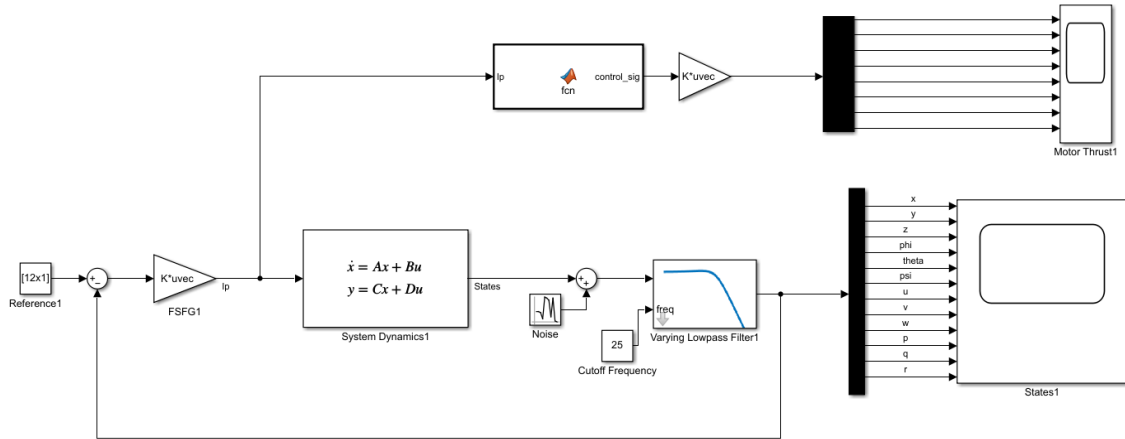
Poles of this system:



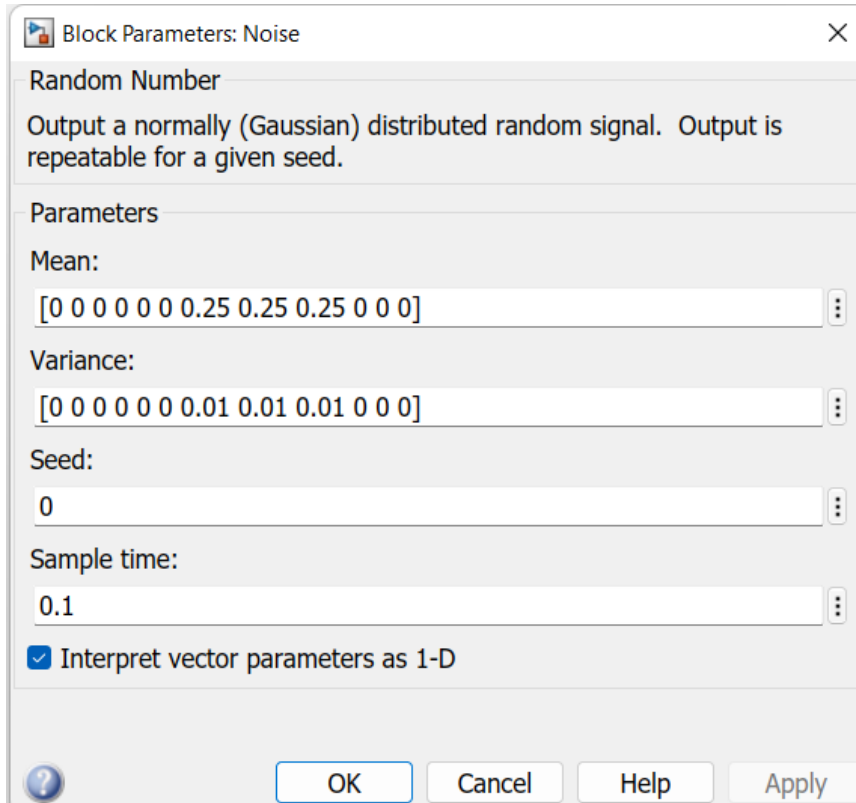
Clearly, the response of the system is faster in the second case.

**Note:** In the above two cases, the open loop and closed loop pole plots are drawn without taking the noise into account. The effect of noise is shown only in the ‘Thrust’ and ‘States’ plots.

Also, in the paper that we referred to, they considered noise caused due to the ‘current velocity’ which has a direct effect on the velocity of the rover instead of the torques. So, for that, a Random Number generator block was included in the model as follows:



The parameters of the noise block are as follows:

The image shows a MATLAB/Simulink dialog box titled "Block Parameters: Noise". It contains the following fields and options:

- Random Number**: Output a normally (Gaussian) distributed random signal. Output is repeatable for a given seed.
- Parameters**:
  - Mean:** [0 0 0 0 0 0 0.25 0.25 0.25 0 0 0]
  - Variance:** [0 0 0 0 0 0 0.01 0.01 0.01 0 0 0]
  - Seed:** 0
  - Sample time:** 0.1
  - ☒ Interpret vector parameters as 1-D
- Buttons**: ? (Help), OK, Cancel, Help, Apply

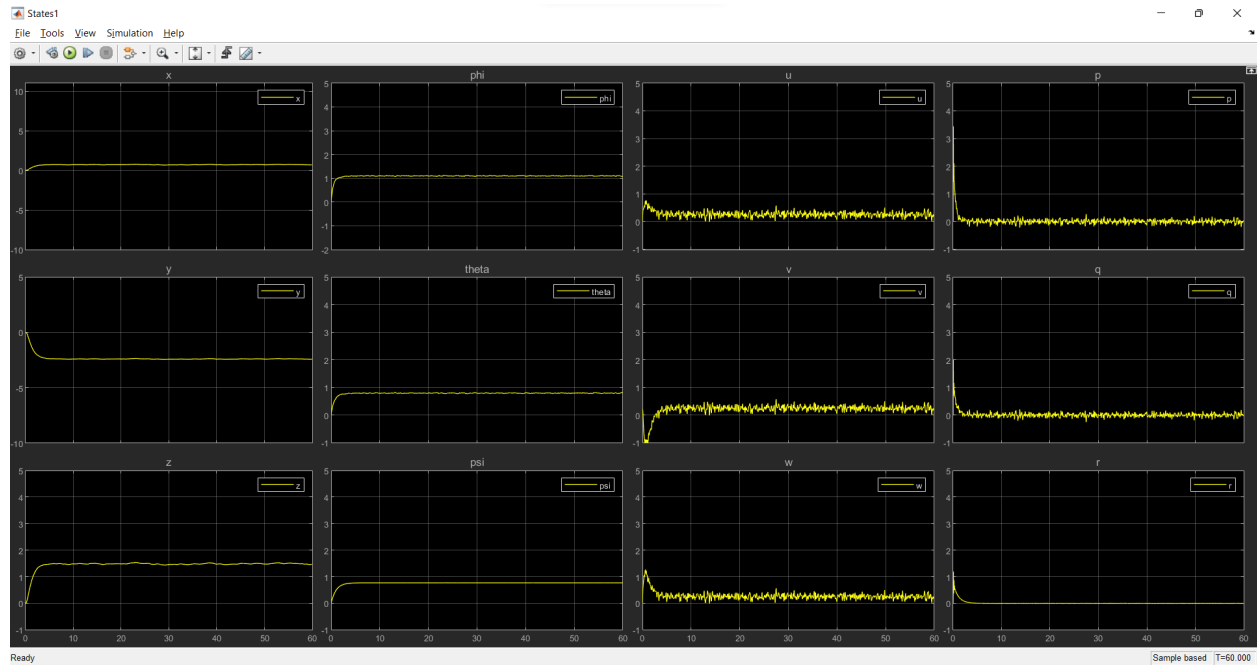
Here, noise is added considering the velocity of current as: [0.25,0.25,0.25] and taking the variance as 0.01 in all the three directions.

Taking R matrix as:  $R=1.2*\text{eye}(6)+0.8*\text{ones}(6)$

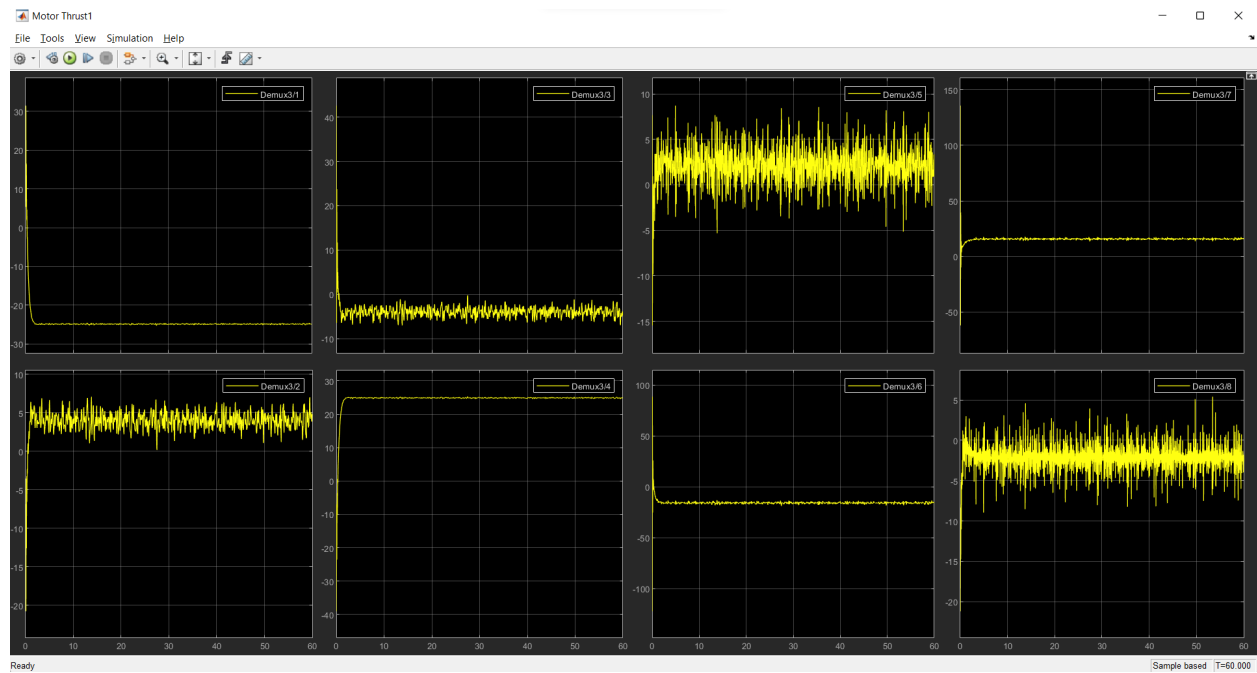
And Q matrix as:  $Q=1000*\text{eye}(12);$

The results that we obtained for 1min stop time are:

States:



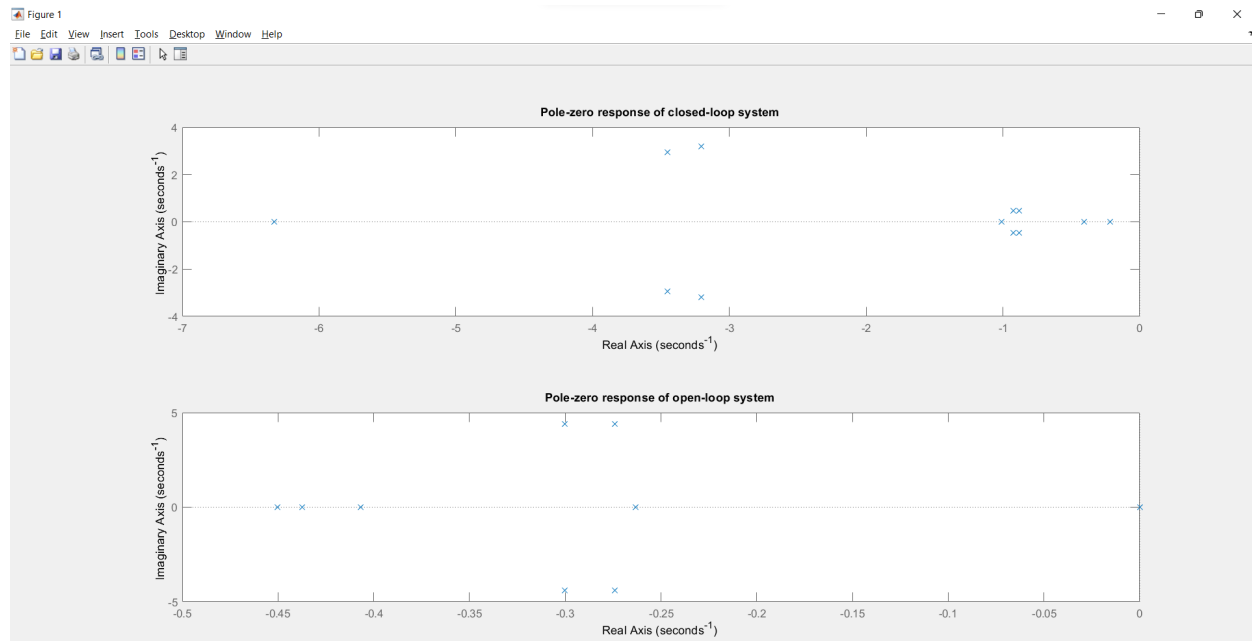
Thrust provided by each thruster:



Clearly, the thrust output involves too much vibrations.

## Analysis of Eigenvalues:

When we analyse the change in eigen values while considering the open loop and LQR controlled closed loop systems, for the same values of Q,R we see a clear change in their position in the pole zero plot.



Considering the above plot, we can see a clear shift in the eigen values in the direction of the negative real axis, ensuring more stability of the system. As we write down the eigen values or the poles of the two systems, the difference becomes evident.

For  $Q = \text{eye}(12)$  &  $R = 0.1 * \text{ones}(12) + 0.9 * \text{eye}(12)$

Open loop poles:	Closed loop poles:
0.0000 + 0.0000i	-6.3274 + 0.0000i
0.0000 + 0.0000i	-3.2047 + 3.1841i
-0.2740 + 4.3867i	-3.2047 - 3.1841i
-0.2740 - 4.3867i	-3.4503 + 2.9329i
-0.2633 + 0.0000i	-3.4503 - 2.9329i
-0.3003 + 4.3834i	-0.9222 + 0.4604i
-0.3003 - 4.3834i	-0.9222 - 0.4604i
-0.4067 + 0.0000i	-0.8836 + 0.4709i
0.0000 + 0.0000i	-0.8836 - 0.4709i
0.0000 + 0.0000i	-1.0104 + 0.0000i
-0.4504 + 0.0000i	-0.2170 + 0.0000i
-0.4375 + 0.0000i	-0.4043 + 0.0000i

As it can be observed, in the case of the open loop system, we have four eigen values or poles located at the origin which leads to the fact that the system is marginally stable. Whereas, for the

closed loop system, most of the poles have shifted towards the left side of the imaginary axis with all the poles lying on the left half of the s-plane. This gives enough proof to say that after implementing LQR control, the system has become more stable as compared to the open loop system and hence has helped in increasing performance in terms of reaching the end states.

## Conclusion:

Given the results of implementing both PID and LQR control on the ROV system, we can see a better response for an LQR control over the PID when we include disturbances in the environment. Also, the LQR control is far more robust in adapting to the change in buoyancy as compared to the PID control, which broadens the spectrum of usage in the former as compared to the latter. When we have changes in the system, using a PID requires tuning all the three gain values for all the control inputs. Tuning the gain values for multiple parameters is not an easy task and there is a very narrow range of values which give the desired results for a specific situation. On the other hand, when we want to adapt to changes while working on LQR control, our tuning is dependent only on the Q,R cost matrices which can help in changing the relative importance of the states and control inputs for a given circumstance. Here, our goal is achieved by changing the values of the matrices, so as to select the more important factor among the states and inputs, where we achieve the desired results for a wide range of values which only differ in terms of speed of transient response and steady state error. Another advantage of LQR over PID control that we have come across was the ability to easily control velocity components along with the position coordinates, which would lead to much more complexity in the case of a PID based controller. For simulating in a noisy environment, we used nonlinear PID control where a water current velocity was added as a disturbance; its value being constant with time and we attained appreciable results. Whereas for the LQR setup, we have used a gaussian white noise as a source of disturbance, which gives a better look at the real world scenario. Upon increasing the Q matrix or simply, the cost of attaining the states, we have observed better performance in attaining the states and less erratic, more uniform thrust outputs produced by the thrusters; which is a more reliable result since it is realisable in a real world environment.

However, one drawback of using the LQR control is its applicability to only linear systems, whereas most of the real world scenarios tend to have non linearity in them. However, this problem can be overcome by linearising the system near the fixed points and applying the control.

Hence, we have come to the conclusion that using an LQR control is **far more reliable** than a PID control for the positional and velocity control of a 6-DOF Autonomous Underwater Vehicle.

## Future Work:

- The model can be implemented with an Adaptive Controller based system which is expected to give better performance.
- We wish to implement our work on the actual physical system to know how it works in the real world. And by doing that, we will be able to understand how the sensor noise affects the system and how the controller will compensate for it.

## **Bibliography:**

- 1) 6-DoF Modelling and Control of a Remotely Operated Vehicle by Chu-Jou Wu, B.Eng. (Electrical) Academic Supervisor: Assoc Prof. Karl Sammut Engineering Discipline College of Science and Engineering Flinders University July, 2018.
- 2) Lum, C. (2019). Introduction to Linear Quadratic Regulator (LQR) Control. Retrieved from YouTube: <https://www.youtube.com/watch?v=wEevt2a4SKI&t=4689>