Industrial Automation Robotics A Assignment

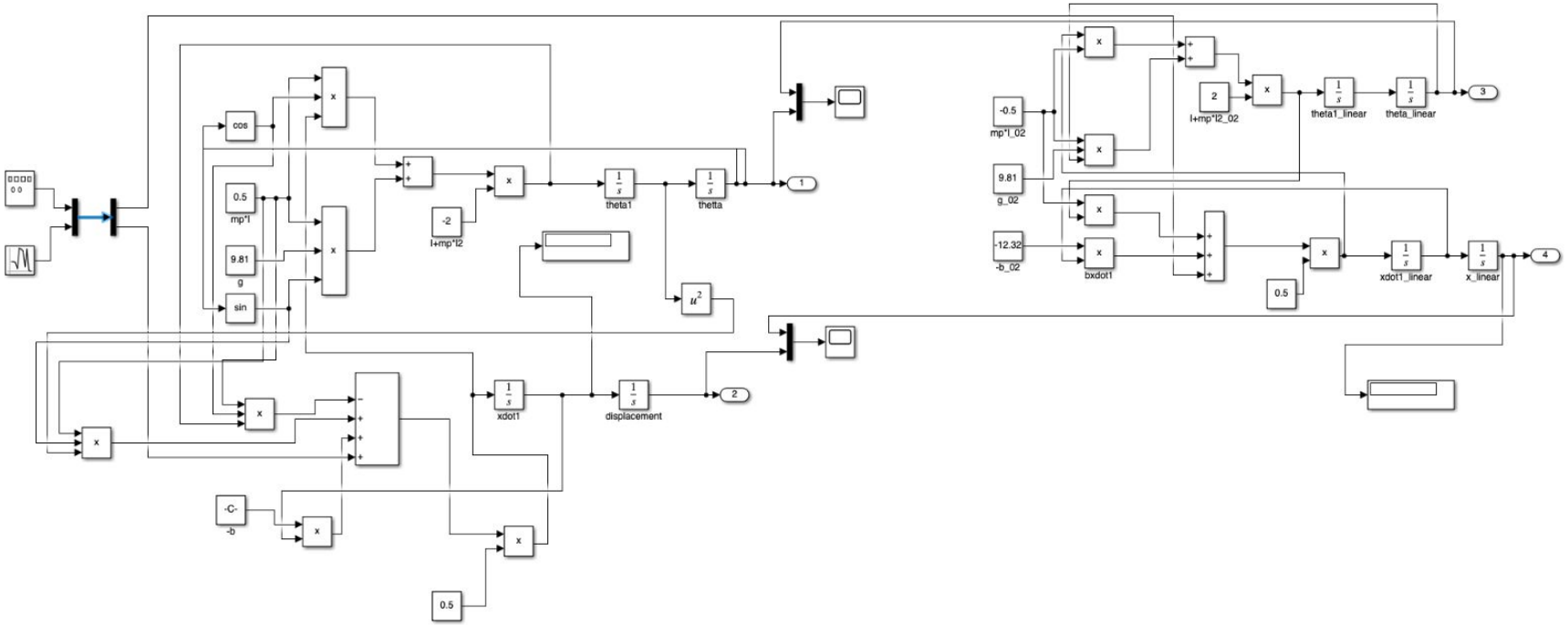
Professor: Maria Gabriella Xibilia
Student: Aruzhan Abdirashova
The University of Messina

Ex. 1: Hanging Crane

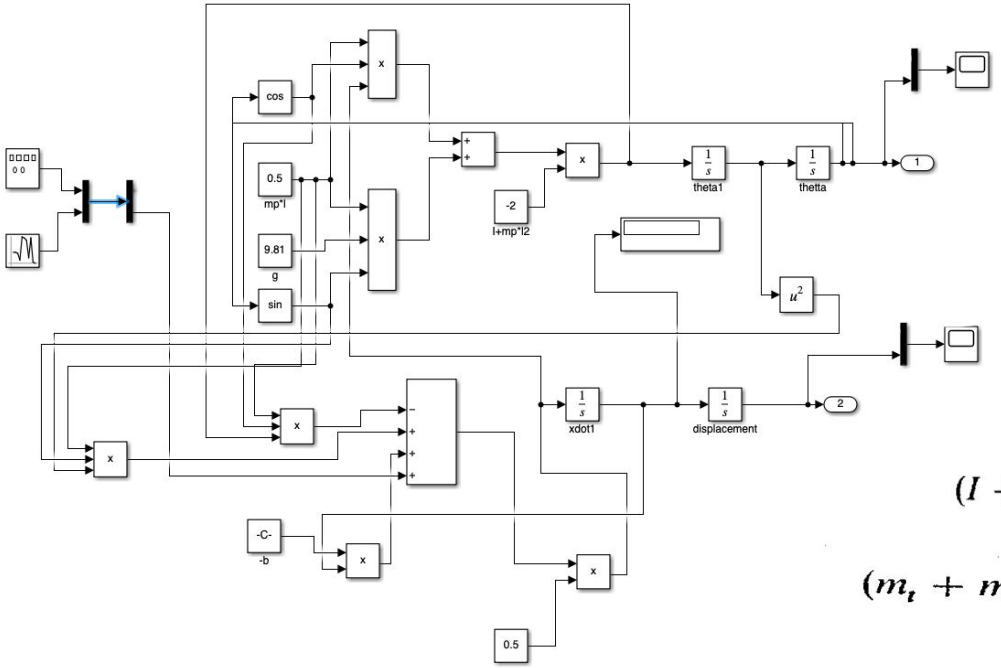
Given the parameters and both the linear and nonlinear equations for the Hanging Crane system, I was able to simulate their models in a blank Simulink model.

Initially, I attempted to use a script with the saved parameters, but encountered an issue where the input displayed as 0 in the Display block, despite showing correctly in the Gain block where I referenced them. To resolve this, I replaced the parameter calls with Constant blocks, manually calculating the variables. I first used a Signal Generator, and later added a Random Number block to introduce noise. To use both as inputs, I employed a Mux block. I utilized the same Scope for both linear and nonlinear models but used separate Scopes for linear displacement and angular position. The nonlinear model used the exact dynamic equations, while the linear model was linearized around the equilibrium point ($\theta=0$). The simulations revealed different responses from the linear and nonlinear models under the same input conditions. I also applied initial conditions for both models and parameters in the Scope and Bode plot analysis.

Ex. 1: Hanging Crane simulation in Simulink



Nonlinear model of the process



The nonlinear equations were modeled in Simulink. A Signal Generator with added noise simulated real-world conditions. Changes in mass and damping significantly affected the system's response, altering natural frequency and damping ratios. Noise introduced small oscillations, especially in θ, highlighting the model's sensitivity to disturbances.

$$(I + m_p l^2)\ddot{\theta} + m_p g l \sin\theta = -m_p l \ddot{x} \cos\theta.$$

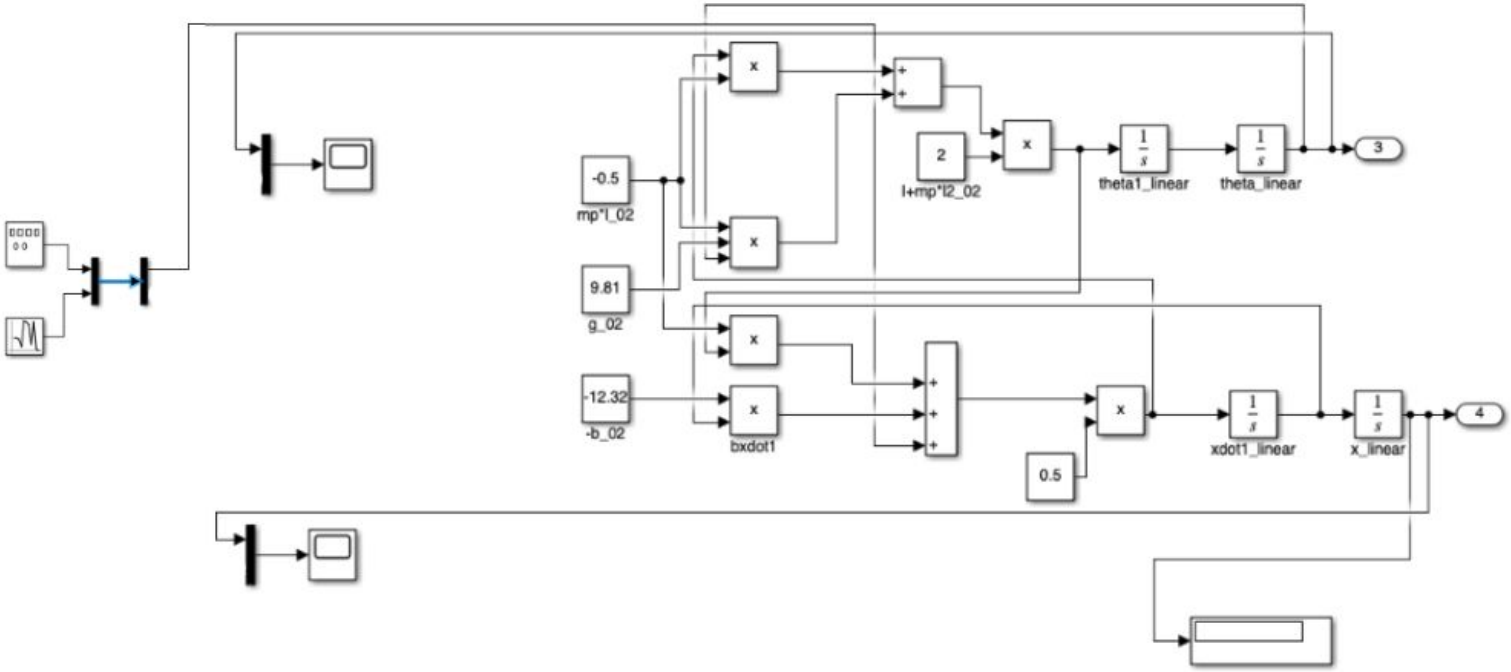
$$(m_t + m_p)\ddot{x} + b\dot{x} + m_p l\ddot{\theta} \cos\theta - m_p l\dot{\theta}^2 \sin\theta = u.$$

Linearized model

$\cos\theta \approx 1, \quad \sin\theta \approx \theta, \quad \dot{\theta}^2 \approx 0$

$$(I + m_p l^2)\ddot{\theta} + m_p g l \theta = -m_p l \ddot{x}$$

$$(m_t + m_p)\ddot{x} + b\dot{x} + m_p l \ddot{\theta} = u.$$



The system's response was compared using the same input conditions as the nonlinear model. The linear model closely matched the nonlinear model for small inputs. But for larger disturbances, it failed to capture the nonlinear dynamics. It was sensitive to noise, but not as nonlinear model.

Transfer function

Neglecting the friction term, let b = 0,

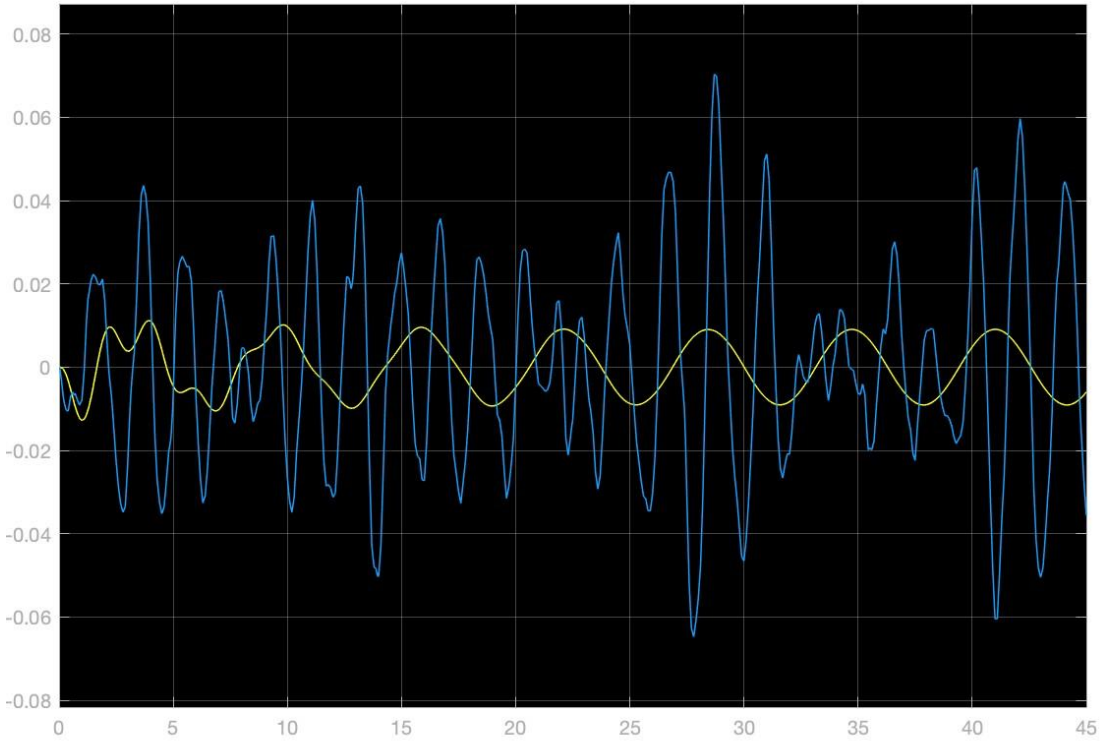
$$\frac{\Theta(s)}{U(s)} = \frac{-m_p l}{s^2\left((I + m_p l^2)(m_t + m_p) - m_p^2 l^2\right) + m_p g l (m_t + m_p)}$$

Scope for angular displacement

Angular displacement θ was analyzed for both models using a Signal Generator input. The angular responses start similarly, indicating that the linear model approximates the nonlinear behavior well for small perturbations. As the simulation progresses, the nonlinear model exhibits more complex oscillations and higher peaks, while the linear model maintains a smoother, less varied response. The nonlinear model shows greater sensitivity to disturbances, with more pronounced deviations from the expected path compared to the linear model.

BLUE LINE - NONLINEAR

YELLOW LINE - LINEAR

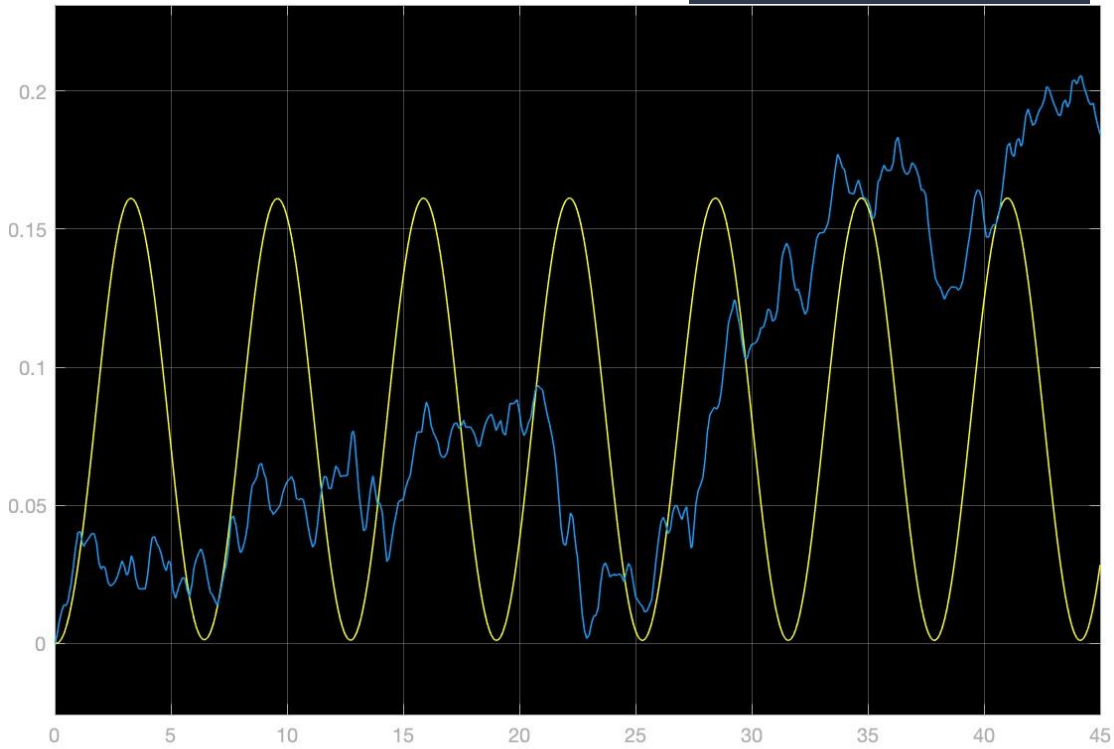


Scope for displacement

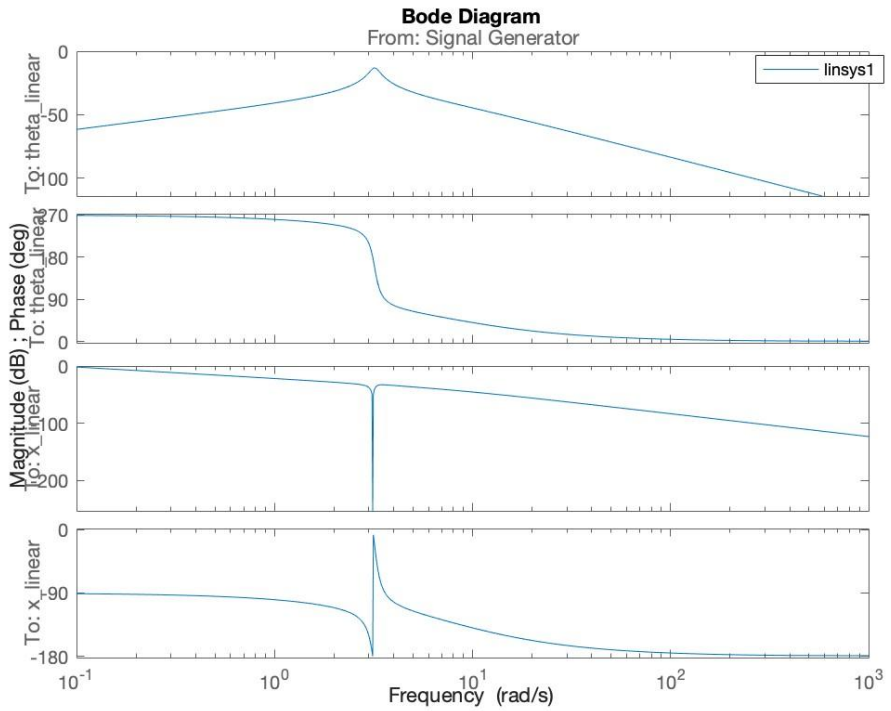
Linear displacement x was analyzed for both models using a Signal Generator input. At the beginning of the simulation, the responses of both models are relatively close. As the input amplitude increases, the nonlinear model exhibits more complex behavior, with additional oscillations and deviations not captured by the linear model. The nonlinear model is more responsive to noise, reflecting dynamics that the linear model oversimplifies.

BLUE LINE - NONLINEAR

YELLOW LINE - LINEAR



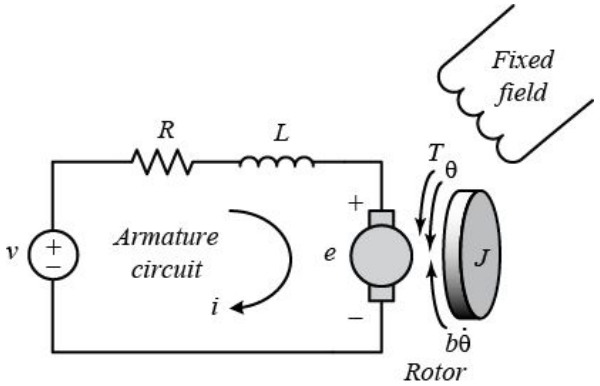
Model Linearizer plot: Bode Plot



The outputs analyzed were the angular displacement θ and the linear displacement x, and the input was solely Signal Generator.

The Bode plot shows a resonance peak in θ around 10 rad/s, indicating that the system is prone to amplified oscillations at this frequency, which could lead to instability if not properly managed. Additionally, both θ and x exhibit significant phase lag, particularly noticeable beyond 1 rad/s, where the phase drops sharply, indicating that the system's response will be increasingly delayed, making it harder to control at higher frequencies. Control strategies must carefully consider these frequency-dependent behaviors to ensure stability and responsiveness.

Ex. 2: Design a position speed controller for the DC motor using a PID and the appropriate Ziegler and Nichols method.



Voltage source (V) applied to the motor's armature is considered as the input of the system; while the rotational speed of the shaft (dΘ/dt) is the output.

The motor torque, T, is related to the armature current, i, by a constant factor K:
T=K*i
The back electromotive force (emf), e, is related to the angular velocity by:
e=K*dθ/dt

From the picture above, we can write the following equations based on the Newton's law combined with the Kirchhoff's voltage law:

$$J\ddot{\theta} + b\dot{\theta} = Ki$$

$$L\frac{\partial i}{\partial t} + Ri = V - K\dot{\theta}$$

The transfer function

Applying the Laplace transform, the above modeling equations can be expressed in terms of the Laplace variable s, where s denotes the Laplace operator..

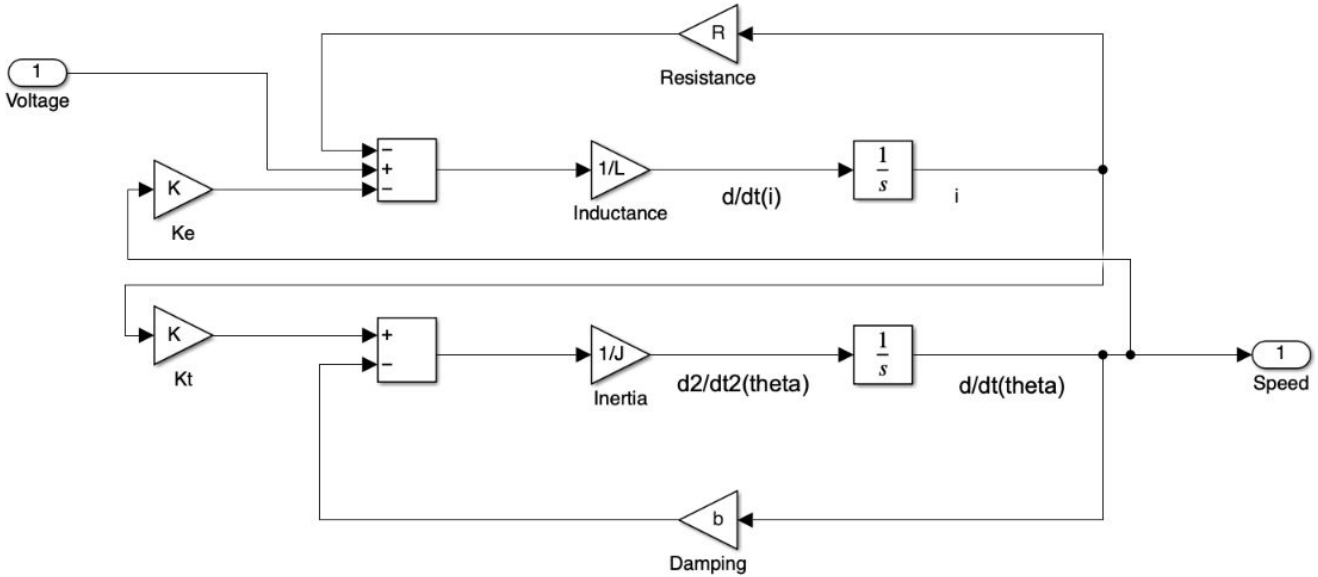
$$s(Js + b)\theta(s) = KI(s)$$

$$(Ls + R)I(s) = V(s) - s\theta(s)$$

The transfer function from the input voltage, V(s),to the angular velocity, ω,is:

$$T(s) = \frac{w(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2} \quad [\frac{rad/_{sec}}{V}]$$

Simulink modeling of DC Motor



Initial parameters used for DC Motor:

J = 0.0005 kg*mˆ2

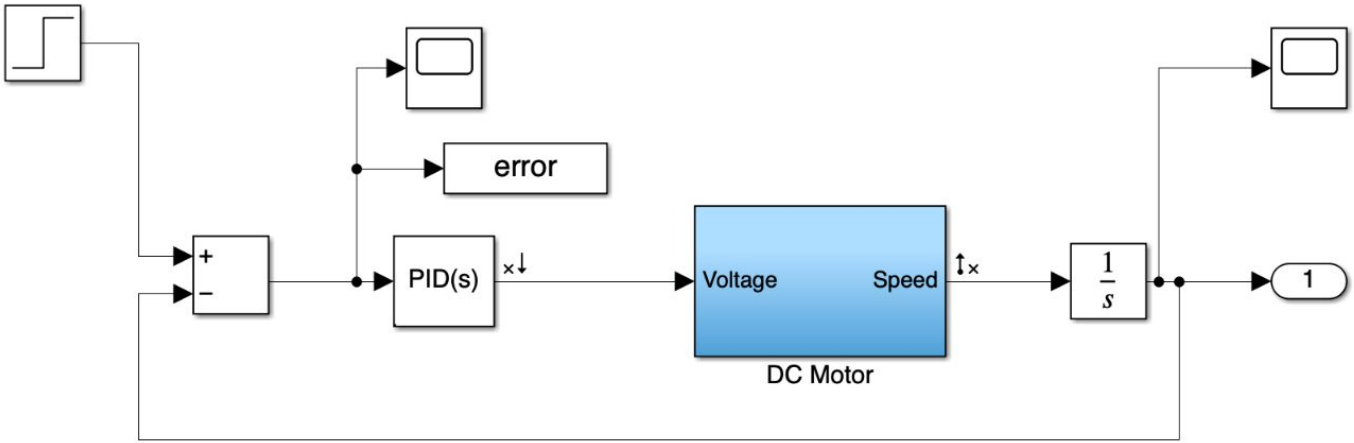
b = 0.000115 Nms

K = 0.17 Nm/A

R = 8 Om

L = 0.2 H

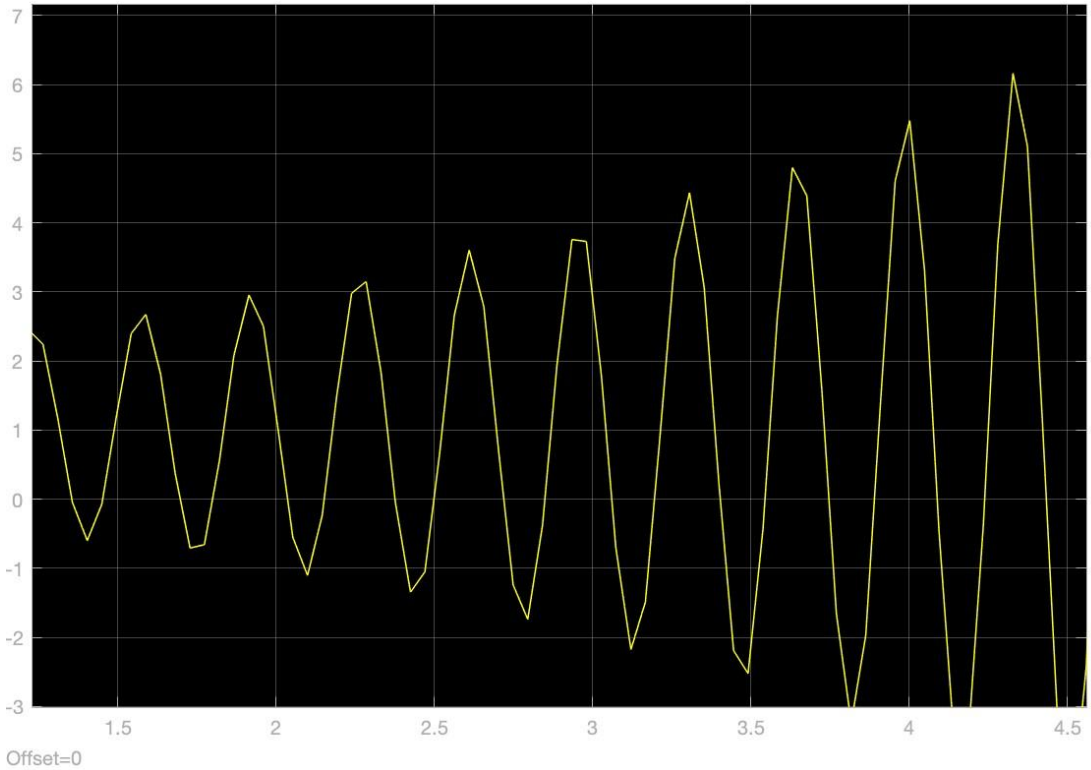
PID Controller Design for DC Motor Simulink



The Step Block: step time -0, initial value - 0, final value - 1.

The error block finds the difference between the setpoint and the actual output, helping the PID controller correct the motor's speed to match the desired value.

Position Feedback Scope

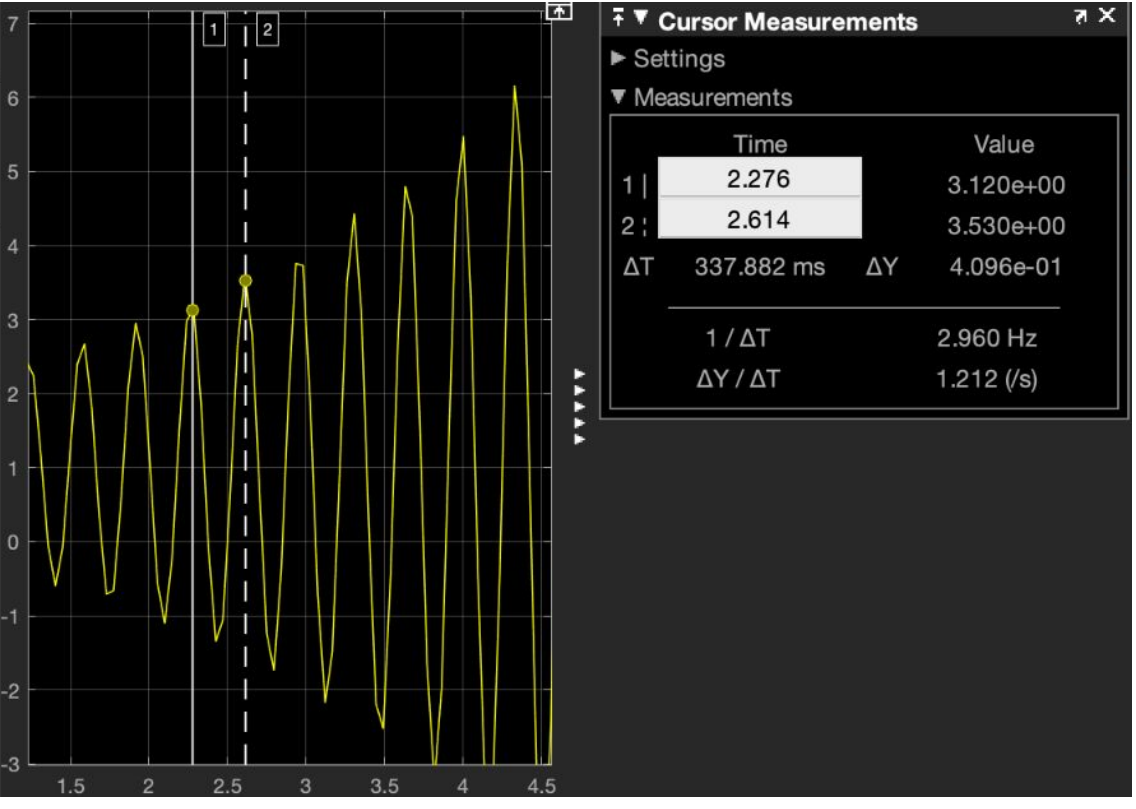


My system is correctly showing the necessary oscillations, meaning I can proceed with Ziegler-Nichols tuning.

My PID settings: 8, 0, 0. As we can see, the proportional gain(P) set to 8, with integral (I) and derivative (D) gains both set to 0. Stop time: 5s.

In the next slide, we can see the calculations.

Ziegler–Nichols Tuning



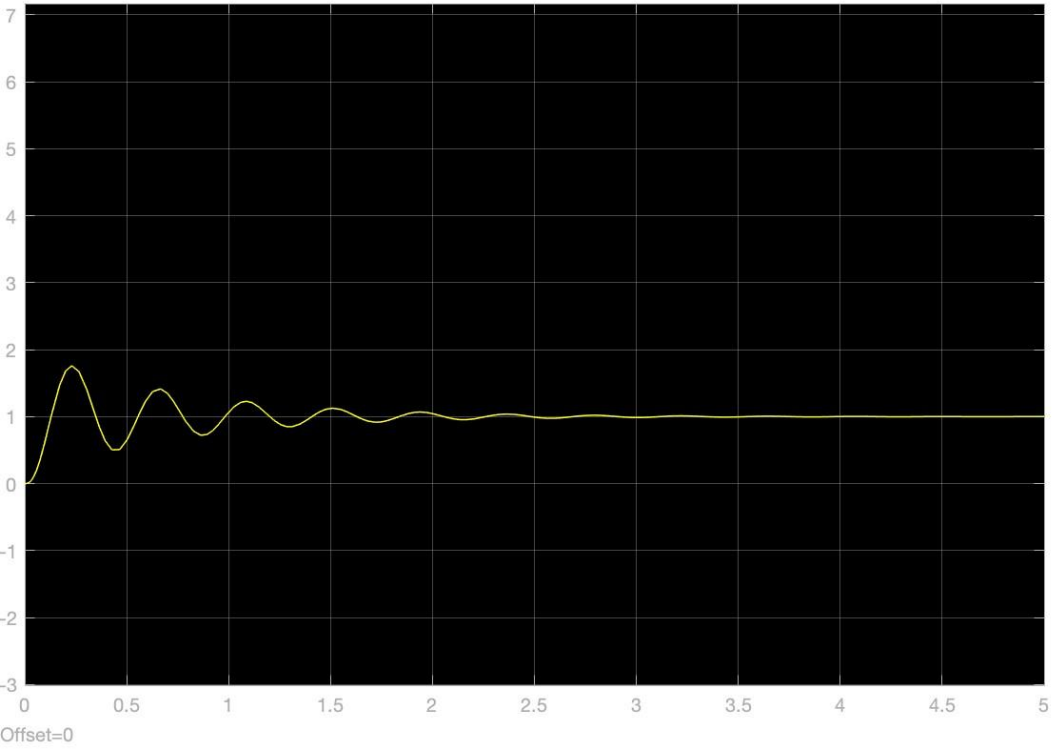
The time difference between the two peaks I have selected is shown as
$\Delta T = 337.882$ ms.
Respectively,
$Tu = 0.337882$ seconds

With the value of Tu and the proportional gain Ku that caused the oscillations, I can apply the Ziegler-Nichols tuning rules to set my PID parameters:

- **Proportional (P):** $Kp = 0.6 \times Ku = 0.6 \times 8 =$ **4.8**
- **Integral (I):** $Ti = 0.5 \times Tu = 0.5 \times 0.337 = 0.17s$
- **Derivative (D):** $Td = 0.125 \times Tu = 0.125 \times 0.337 =$ **0.042s**

Integral Gain(I) = 1/Ti = **5.88**

Simulation with tuned PID Controller



The simulation shows a small overshoot, which is typical of Ziegler-Nichols tuning. The settling time is reasonable, and the system stabilizes after a brief period. The system has minimal steady-state error, indicating that the PID controller is effectively regulating the motor position.

Briefly, the controller achieves the desired setpoint with some overshoot but settles quickly, which is consistent with Ziegler-Nichols tuning outcomes.

Ex. 3: using the attached dataset identify a ARX and an NFIR model of the second output of the process

I used MATLAB to construct both ARX and NFIR models.

First, I loaded the MATLAB input and output files into the workspace using the load command. For both models, I utilized 'l4_in' as the input and 'l4_out2' as the output. I then adjusted the model parameters to achieve the best performance.

MATLAB code(ex. NFIR)

First I split data into training and testing sets. As usual, I gave 70% for training and 30% for testing:

```
n = length(output);
train_size = round(0.7 * n);
test_size = n - train_size;

input_train = input(1:train_size, :);
output_train = output(1:train_size);
input_test = input(train_size+1:end, :);
output_test = output(train_size+1:end);
```

For identifying the NFIR model in Matlab, I used 'arx' function by setting the model orders properly.

```
na = 0
nb = [1 1 1 1 1]
nk = [1 1 1 1 1]

data = iddata(output_train, input_train);
nfir_model = arx(data, [na nb nk]);
```

Now we display the model:

```
disp(nfir_model);
```

Additionally, I validated the model on test data:

```
output_pred = predict(nfir_model, iddata(output_test, input_test)
```

And calculated the performance metric. In my case, I chose MSE:

```
mse = mean((output_test, output_pred.OutputData).^2);
disp(['MSE on the test data: ', num2str(mse)]);
```

Simulate the model:

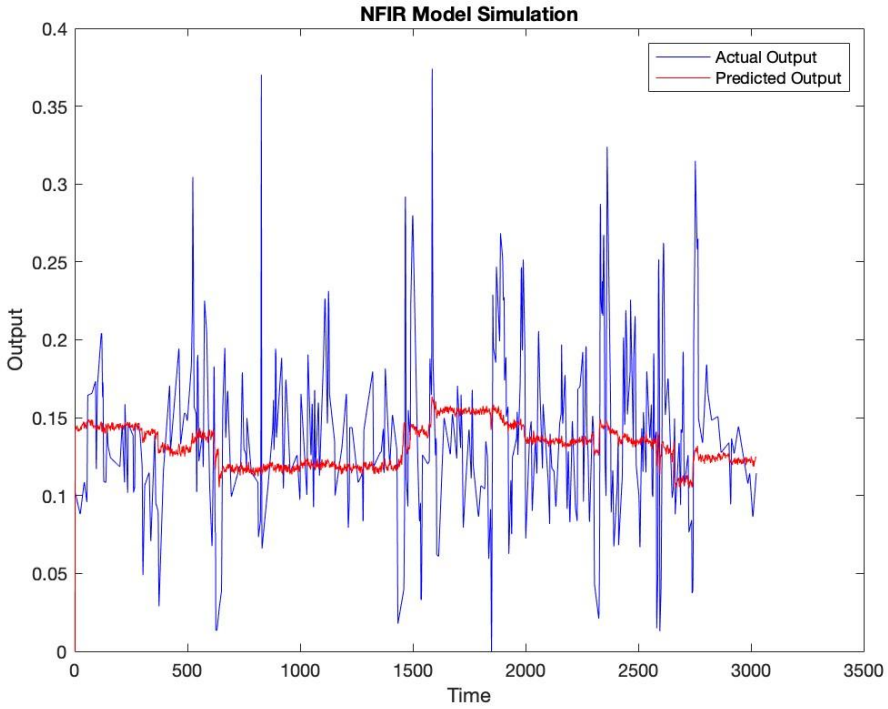
```
output_sim = sim(nfir_model, input_test)
```

NFIR PLOT

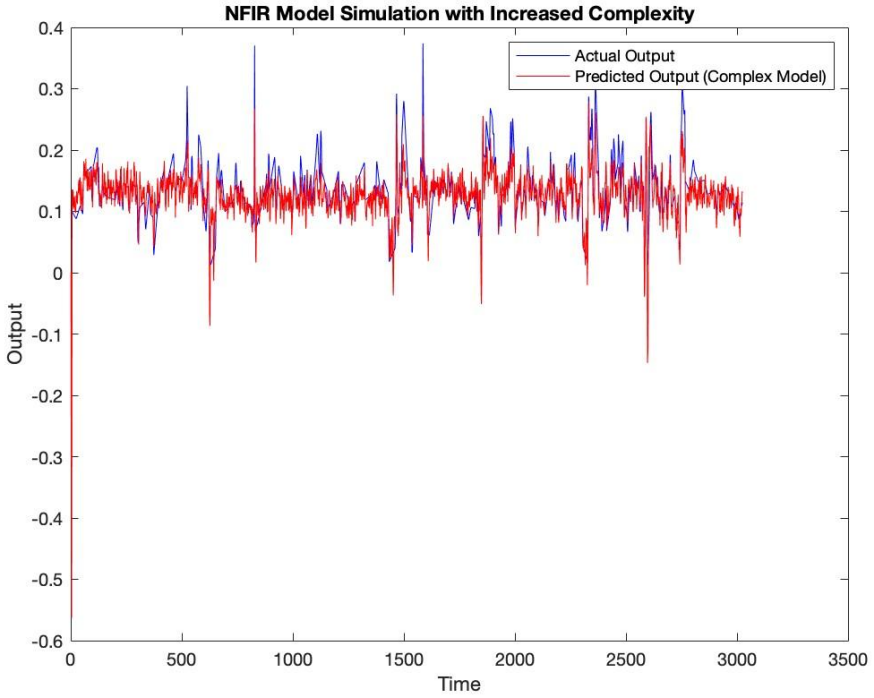
After plotting the actual vs predicted output using previous code, the figure shows that the NFIR model captures the general trend of the output. Unfortunately, it did not fully capture the high-frequency variations in the actual output data.

Because of that, I decided the increase the model complexity.

MSE on the test data: *0.0019855*



Complex NFIR PLOT



For increasing model complexity, I adjusted

nb = [5 5 5 5 5],

by increasing the number of delays for each input. After updating the model,

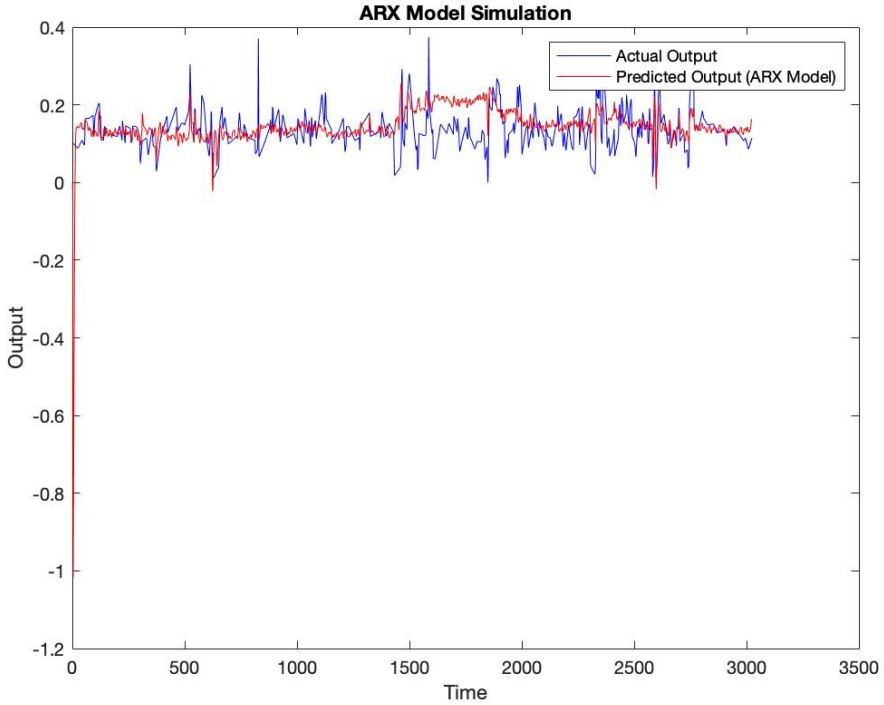
I could observe that the model fits the data better. Also, the predicted output in the plot closely follows the actual output across the time series. Additionally, MSE on the test data: *0.00089152*, it indicates increased model complexity has indeed improved the performance of the NFIR model.

Complex ARX PLOT

For ARX model, I chose the order of the autoregressive part to be 'na = 2', and other model orders left to be the same as in last NFIR model.

The MSE for the ARX model is *7.7929e-05*, which is significantly lower than the MSE of the NFIR model.

This indicates that the ARX model is better at capturing the dynamics of the system. The ARX model provides a better fit for the data compared to the NFIR model. However,, it visually appears to me that ARX model has a less close fit to the actual data compared to the previous NFIR model. It could be due to model overfitting to training data due to its complexity. I decided to lower the complexity.

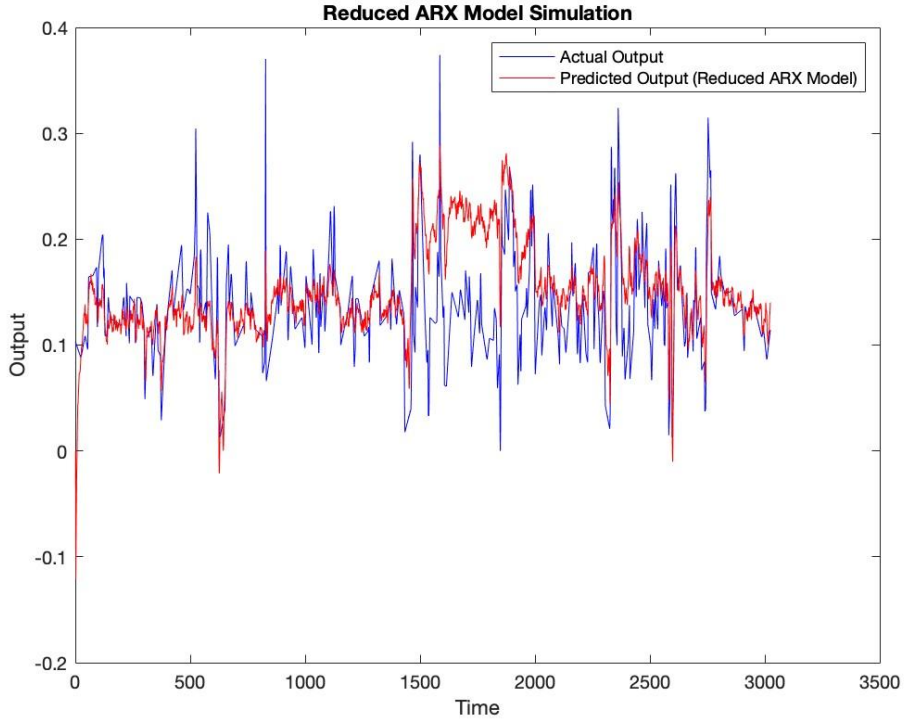


ARX PLOT

I reduced the number of delays, and order of autoregressive part parameters:

na = 1; nb = [2 2 2 2 2]; nk = [1 1 1 1 1];

This model achieved the lowest MSE of *0.00012444*, reflecting its superior accuracy in predicting the output. The visual fit between the predicted and actual outputs was also the best among all models tested, indicating that this ARX model effectively captured the system's dynamics without overfitting.



Conclusion

In this analysis, I evaluated the performance of different models—specifically, NFIR models with orders 1 and 5, and an ARX model with orders 1 and 2. The goal was to identify which model best captures the system's dynamics and provides the most accurate predictions.

The **NFIR(1)** model, due to its simplicity, didn't perform well. It had a higher MSE and didn't fit the data closely, missing much of the variability in the output. When the model order was increased to **NFIR(5)**, the performance improved, reflected by a lower MSE and a better fit to the data. However, this model still didn't fully capture the more complex patterns in the data, likely because it doesn't take past outputs into account.

The **ARX(1,2)** model, on the other hand, performed the best. It achieved the lowest MSE and provided a much closer fit to the actual data, successfully capturing the system's dynamics. This model's ability to consider both past inputs and outputs allowed it to model the system more accurately.

Overall, the **ARX(1,2)** model is recommended for its superior accuracy in capturing the system's behavior, while the **NFIR(5)** model serves as a simpler, though less precise, alternative.