EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, *SVERIGE 2019*

A comparison between a traditional PID controller and an Artificial Neural Network controller in manipulating a robotic arm

En jämförelse mellan en traditionell PID-styrenhet och en Artificiell Neural Nätverksstyrenhet för att styra en robotarm

JOSEPH ARISS

SALIM RABAT

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

A comparison between a traditional PID controller and an Artificial Neural Network controller in manipulating a robotic arm

En jämförelse mellan en traditionell PID-styrenhet och en Artificiell Neural Nätverksstyrenhet för att styra en robotarm

Joseph Ariss

Salim Rabat

2019-06-06

Bachelor's Thesis

Examiner
Örjan Ekeberg

Academic adviser
Jörg Conradt

KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science (EECS)
SE-100 44 Stockholm, Sweden

Abstract

Robotic and control industry implements different control technique to control the movement and the position of a robotic arm. PID controllers are the most used controllers in the robotics and control industry because of its simplicity and easy implementation. However, PIDs' performance suffers under noisy environments. In this research, a controller based on Artificial Neural Networks (ANN) called the model reference controller is examined to replace traditional PID controllers to control the position of a robotic arm in a noisy environment. Simulations and implementations of both controllers were carried out in MATLAB. The training of the ANN was also done in MATLAB using the Supervised Learning (SL) model and Levenberg-Marquardt backpropagation algorithm. Results shows that the ANN implementation performs better than traditional PID controllers in noisy environments.

Keywords:
Artificial Intelligence, Artificial Neural Network, Control System, PID Controller, Model Reference Controller, Robot arm

Sammanfattning

Robot- och kontrollindustrin implementerar olika kontrolltekniker för att styra rörelsen och placeringen av en robotarm. PID-styrenheter är de mest använda kontrollerna inom roboten och kontrollindustrin på grund av dess enkelhet och lätt implementering. PID:s prestanda lider emellertid i bullriga miljöer. I denna undersökning undersöks en styrenhet baserad på Artificiell Neuralt Nätverk (ANN) som kallas modellreferenskontrollen för att ersätta traditionella PID-kontroller för att styra en robotarm i bullriga miljöer. Simuleringar och implementeringar av båda kontrollerna utfördes i MATLAB. Utbildningen av ANN:et gjordes också i MATLAB med hjälp av Supervised Learning (SL) -modellen och Levenberg-Marquardt backpropagationsalgoritmen. Resultat visar att ANN-implementeringen fungerar bättre än traditionella PID-kontroller i bullriga miljöer.

Nyckelord:
Artificiell Intelligens, Artificiell Neuralt Nätverk, Kontroll System, PID-kontroller, Modellreferenskontroller, Robotarm

Table of Contents

List of Figures

List of Tables

List of acronyms and abbreviations

ANN: Artificial Neural Network
MAP: Maximum A Posteriori
MSE: Mean square error
PID: Proportional Integral Derivative
PV: Process Value
SL: Supervised Learning
SP: Setpoint

1. Introduction

1.1. Background

Machine learning is the topic of the decade, lots of research has been done in this field and is still ongoing, because of how broad its applications can be. Today, machine learning is almost a part of every industry, and can be used in different ways. One of these ways is Artificial Neural Networks, which are built to mimic the biological neural networks that humans have. [1]

One would think that with all the research being done in this field we would already find it everywhere and industries would race to replace the old and traditional with the new and smart. But this doesn't always apply, specifically in the control industry where traditional control methods are still being used and seen as dependable and robust.

Traditional PID (Proportional, Integral, Derivative) controllers are used in most of the industrial control applications because of its feasibility and easy implementation. However, PID controllers are not perfect and can perform poorly in complex, non-linear, or time/delayed linear systems. Because setting the Parameters (Kp, Ki,and Kd) of the traditional PID controller in these types of control systems can be a difficult task. [1] [2]
The most common problem with traditional PID controllers is overshooting, as in go past the intended point (the desired output). Another common problem is that PID controllers perform poorly in noisy environments, when the noise is applied to the feedback loop.

Many control research projects have been developed and presented in the last years specially in the robotic manipulators area. Robotic manipulators are recently gaining more interest in the medical, educational and industrial fields. Robots have the advantage of being able to operate in hazardous area such as nuclear reactors without risking or endangering human lives. Therefore, being able to increase and analyse the precision and movement of robotic arm manipulators is an important research topic.

Artificial Neural Network controllers are being tested, used and proven to be a working technology in the control industry. But then again, PID controllers are good enough in linear systems, and most non-linear systems can be linearized with the use of multiple PID controllers. So, if it's not broken don't fix it, right?

1.2. Purpose

This research discusses two implementations of control systems. Firstly, by designing a traditional PID controller and automatically tuned using built-in applications that are available in MATLAB. Secondly, a model reference control system based on artificial neural networks has been designed for the same plant.

The aim of this research is to compare the traditional PID controller with an artificial neural network controller and see in which scenarios would replacing the PID controller with an

ANN controller would actually be beneficial. The comparison will be based on the performance under different feedback noise levels and applied on robotic arm with different friction levels.

In addition, the comparison of both controllers will be also based on the accuracy and stability which can be derived from the error calculated from counting the distance between the desired output and the current output.

1.3. Research question

Due to the increased importance of Artificial Neural Networks in today's industries, this leads to the main research question: Where can having an artificial neural network controller be beneficial and more suitable than the traditional PID controllers in linear systems?

1.4. Outline

This thesis is organised as the following: The second chapter presents the theory behind all the technology and concepts used in the thesis. The third chapter presents the methods used to conduct the experiment that the research is based on. The forth chapter presents the results of the experiments that were conducted. The fifth chapter presents a discussion about the results from chapter 4 to understand them clearly and make sense of them. The last chapter presents the conclusions that could be drawn from the research.

2. Theoretical Background

2.1. PID (Proportional, Integral, Derivative) Controllers

PID is one of the most common controllers that are used to control different plants, such as temperature control, motor control, and level control. PID controller is a closed loop system that calculate an error e(t) between a desired setpoint (SP) and a process value (PV) that is measured from the plant as seen in the figure 2 below. [3]
Figure 01 shows the structure of a closed loop control system. It consists of a controller block C(s) and a plant/process block P(s).

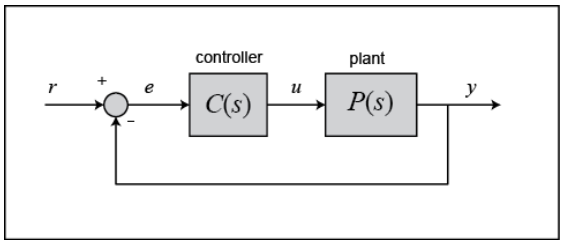


Figure 1: Closed loop control system

A PID controller consists of three parts: a proportional part, integral, and derivative part [3]:

- Proportional Part: The output will be proportional to the error e(t) which is multiplied by a proportional constant Kp to get the output.
- Integral Part: It eliminates the steady state error by integrating it over a specific period until the error becomes equal to zero.
- Derivative Part: Anticipates how the error behave in the future. Thus, the output depends on the rate of change of the error with respect to time.

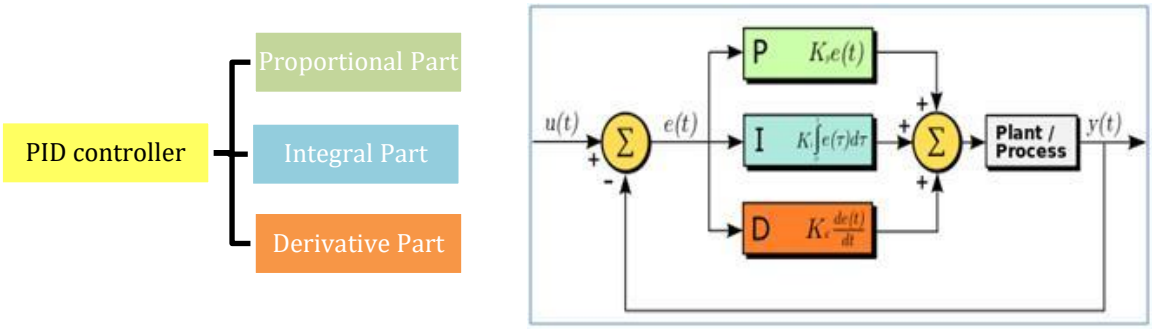


Figure 2: PID Controller connected to a plant/process where SP is referred as u(t) and PV is referred to as y(t)

The PID gives its output to the plant/process and then the output of the plant **y(t)** is compared to a set point or reference signal **u(t)**. Feedback signal from the plant/process block is then compared with the reference signal giving the error signal **e(t)** which is then fed to the PID. Then, the PID algorithm produce a command signal according to the proportional, integral and derivative calculations. The produced signal is then applied as a new input to the plant according to the formula:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')\,dt' + K_d \frac{de(t)}{dt}$$

2.1.1. PID Tuning

There are different methods of tuning a PID controller to produce an optimal control function. Tuning is the adjustment of the three control parameter (Proportional, Integral, and Derivative gain) to corrects values to achieve a desired response from the plant. Some of the most known tuning techniques are: Manual Tuning, Ziegler–Nichols method, and PID tuning software. [4]

- **Manual Tuning:** One method to manually tune a PID controller is to first set the integral gain Ki and derivative gain Kd to zero. The next step is to increase the proportional gain until the step response starts to oscillate and set Kp to half of that value. Then increase Ki until the output of the step response is corrected. Lastly, increase Kd to improve the stability of the output until the response is acceptably fast.

  Table 1 shows the study of three parameters, namely Kp, Ki and Kd, where the effect of increasing and decreasing the gains is determined.

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
Kp	Decrease	Increase	Small change	Decrease	Degrade
Ki	Decrease	Increase	Increase	Eliminate	Degrade
Kd	Minor change	Decrease	Decrease	No effect in theory	Improve if **Kd** is small

Table 1: The effects of increasing or decreasing the gains individually

- **Ziegler–Nichols method:** The method was developed by John G. Ziegler and Nathaniel B. Nichols. The starting point is to set the integral and derivative part of PID controller to zero and increase the proportional gain until we reach an ultimate gain Ku and an ultimate oscillation period Tu at which the output of the system starts to oscillate. [5]

  Ku and Tu are then used to find Kp, Ki, and Kd gains according to table 2 below:

Controller	Kp	Ki	Kd
classic PID	0.6Ku	1.2*Ku/Tu	(3*Ku*Tu)/40
some overshoot	Ku/3	0.666*Ku/Tu	(Ku*Tu)/10

no overshoot	Ku/5	(2/5Ku)/Tu	(Ku*Tu)/15

Table 2: Ziegler-Nichols Method

- **PID tuning software:** In most modern implementation of PIDs, tuning and loop optimization software and packages are being used to obtain the best PID control parameters to ensure the best optimal results. An example of those are: PID tune and Easy PID Tuning that are used by MATLAB. [6]

2.3. Open loop vs Closed loop systems

Control operations can be classified into open loop systems and closed loop systems. The main difference between both systems is the existence of feedback as seen in figure 3. An open loop system acts according to the input of the systems while the output does not have any effect on the control signal of the system. A closed loop system takes into consideration the current output of the system and use to keep the process on a desired setpoint. [7]

Open-loop systems are simple and do not compensate for any disturbances, physical changes such as circuit condition or other changes like temperature in the system which can affect the output. Therefore, a negative feedback control system was used in the implementation to eliminate these effects. A feedback system is a system that sample the output signal and then feed it back to the input to enable the system to adjust the reference signal, so the output reaches the desired response. A negative feedback system is a system that subtract the output of the plant from the input to get an error signals which describes how far is the output from the desired reference. [7]

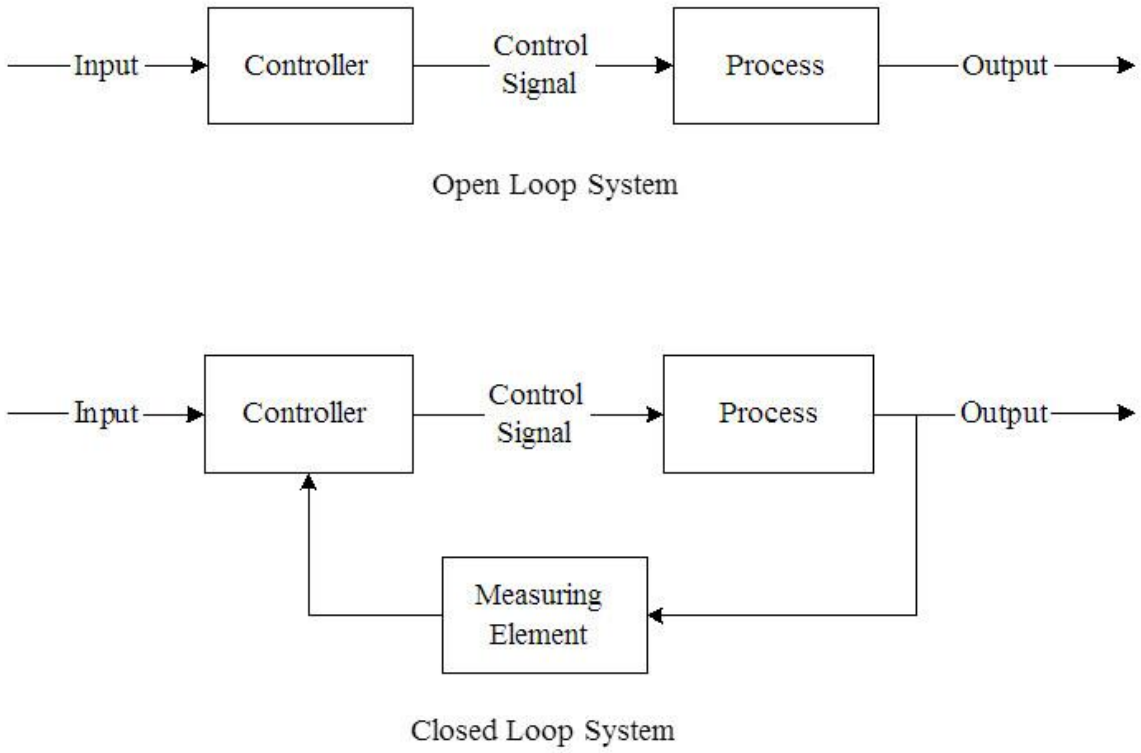


Figure 3: Open Loop System and Closed Loop System

2.4. Supervised learning

Supervised learning (SL) is a branch of machine learning and artificial intelligence which gives computers the ability to learn without being explicitly programmed. SL maps an input to an output according to input output pairs. In other words, we have several data points or sample described using predicted variable or features and a target variable.

The aim of SL is to build a model that is able to predict the target variable. SL can automate time-consuming or expensive manual tasks such as doctor's diagnostic. SL algorithms needs labelled data like labelled historical data of the process or to make experiments to get the data. In SL, the labelled data is divided into two sets of data. A training set and a test set. A training set is used to train the model to learn a specific mapping of inputs and outputs. The testing set is then used to validate and evaluate if the trained model can give valid outputs. [8]
There are a lot of different Supervised Learning algorithms such as: Classification and regression decision trees, Naive Bayes, and Artificial Neural Networks. [9]

- **Classification and Regression Decision Trees:** a decision tree is a flowchart like structure where each internal node of the tree denotes a test on an attribute, each branch represent an outcome of a test, and each leaf holds a class label which is used to make a prediction. In other words, the node of the tree represents a single input variable (x) and the leaf represents an output variable (y). [9]
- **Naive Bayes:** It is a simple machine learning classification model that makes perceptions according to Maximum A Posteriori decision rule (MAP) and it is used in very common applications like document and spam detection. [9]
- **Artificial Neural Networks (ANN):** Artificial neural Networks are the scope of our thesis. ANN replicates the way the neurons in our brain work. They consist of a number of connected artificial neurons connected to each other forming multiple layers. Each layer has several inputs X. Each input has a weight W which is added to a bias B forming a sum equation XW + B. The sum equation is fed into an activation function which will compute the output. Training algorithms is used to train the ANN which adjusts the weight W according to a cost function. One of the most common algorithms is Backpropagation which computes the gradient of the function to minimize the error by updating the weights.
  Simple or complex network of processing units called neurons work together to be capable of learning, adapting, and recognizing relationships. This has been demonstrated in many areas such as control application, function approximation and pattern recognition. Connections between neurons function like synapses in the brain and transmit signals from one neuron to another. The activated neuron processes the signal and signals downstream neurons that are connected to it.

  Typically, Neurons are organized in layers. Different layers may perform different kind of transformation on their input. A simple ANN networks with one input layer, one hidden layer, and one output layer (figure 4) looks as follow [9]:

- **Input layer:** each of the nodes of the input layer represent an individual feature form each sample within the data set that will passed into the model.

- **Hidden layer/layers:** are layer/layers between the input layer and the output layer. Each of the connection between the input layer and hidden layer transmit the data to the hidden layer. Each connection has its own assign weight that can be any number between 0 and 1. Weights represent the strength of the connection between the neurons. The inputs from the input layer is passed to the hidden layer via the connection and the input will be multiplied by the weight of this connection. A weighted sum is computed within each of the connection that is pointing to the neuron. The sum is then passed to an activation function that transform the results to a number between zero and one. The result is passed to the next layer (another hidden layer or output layer)
- **Output Layer:** it typically represents categories. The number of neurons (Nodes) in this layer is directly related to the kind of problem that the neural network is trying to solve.

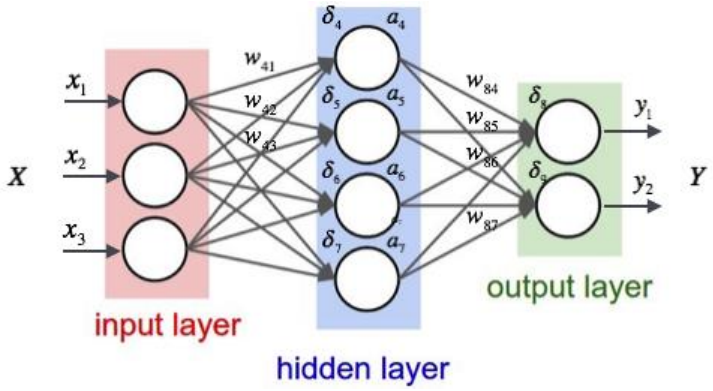


Figure 4: Architecture of a shallow neural network

2.5. Newton's method

Newton's method is an iterative method used to approximate the point of intersection of a function with the x-axis, as in the root of the function where $f(x) = 0$. The first iteration begins with a starting point that is usually guessed $f(x_0)$, and from that point, a tangent line is drawn (first derivative) and the point of intersection of the tangent line with the x-axis is found. That point is used to do the same process again to get even closer to the point of the graph's intersection with the x-axis according to the formula:

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ until the approximation reaches a sufficient value. [10]

2.6. Gauss–Newton algorithm

The Gauss-Newton algorithm is an iterative algorithm which is a modification of the Newton method that is used to solve non-linear least square problem by minimizing a sum of squared function values. When it is applied to optimization, it gives the procedure for finding the minimal of the function. [11]

2.7. Gradient descent

Gradient Descent is an iterative optimization algorithm that is mostly used in Machine Learning Model to find the minimum of a function. It minimizes the cost function of the model to its local minimum by iteratively going in the direction of the negative slope and adjusting the weights and biases of the model. [12]

2.8. Backpropagation

Backpropagation is the most common algorithm for training artificial neural networks to update the weights. It updates the weights that lessen the error function by calculating the gradient of the function with respect to each weight and altering them according to the weight that reduces the error function the most. [13]

2.9. Levenberg-Marquardt algorithm

The algorithm was first discovered by Kenneth Levenberg in 1944 and then rediscovered by Donald Marquardt 1963. Levenberg-Marquardt (LM) Algorithm is a hybrid algorithm that combines gradient descent with Gauss-Newton method which is used to solve nonlinear least square problems. LM is slow and behaves like the gradient descent method when the solution is far away from the correct one and behaves like the Gauss-newton method when the solution is close to the correct one. LM provides the advantages of both methods. It benefits from the speed of the Gauss-Newton algorithm and the stability of the Gradient Descent algorithm. [14] [15]

2.10. Model Reference Controller

Model reference controller is an implementation of neural network controllers that has two separate neural networks. One network is used for the plant identification as in the robotic arm's behaviour. The second neural network is the controller that trains the robotic arm on tracking a reference model. [16]

The neural network controller has three input neurons, ((r) takes the desired trajectory, (y) takes in the plant output as in feedback of the robotic arm, the third input neuron (u) takes in the output of the same neural network, which is the control signal. The plant neural network has two input neurons, one takes the input (u) and the other takes the input (y).

3. Method

3.1. Research Process

Our research process was done through reading different articles using Google, Google scholar and KTH databases. We read a bunch of articles that were found using the keywords “PID”, “ANN”, “Neural Network controller”, “robotic arms motor control”, “robotic manipulator”, “DC motor control” and “motor control simulation”. The articles were different in complexity, as we read some beginners tutorials, bachelor thesis, master thesis and phd thesis. The research process took the longest time to get acquainted with the subject we’re conducting the study on.

3.2. Software used

All the simulation was done in Simulink and the data used in the results was extracted using MATLAB. All the graphs were plotted in MATLAB.

3.2.1. MATLAB

MATLAB is a programming platform used by engineer and scientist to analyze, develop, and validate data and algorithms. Moreover, MATLAB is used for mathematical and scientific computation, modelling, simulation, and application development. MATLAB includes built in applications such as deep learning, control systems, and signal processing applications which were used in this paper. [17]

3.2.2. Simulink

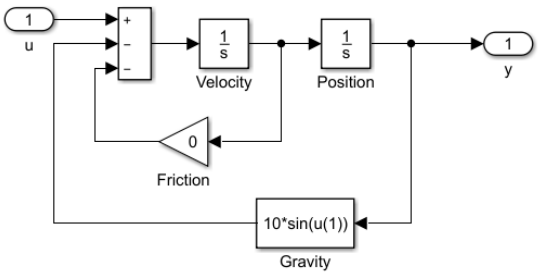
Simulink is one of the add-on applications that are provided in MATLAB. It provides interactive tools and customizable blocks for simulation and modeling of different systems and also to generate VHDL and Verilog codes. [18]

3.3. Implementation

3.3.1. Simulink blocks used

- Uniform Random Number: Output a uniformly distributed random signal. Output is repeatable for a given seed.
- Sum block: Add or subtract inputs
- PID: This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking.

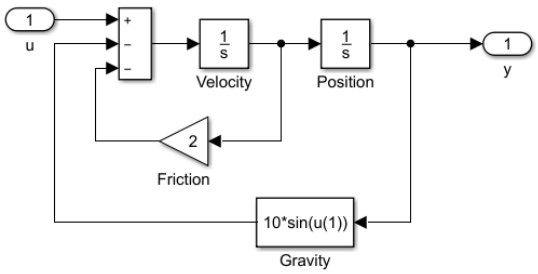
- Model Reference Controller Block: This block's UI is used to decide on the neural network architecture by deciding the size of the hidden layer, number of input and output neurons in both the neural network plant and neural network controller. It can also be used to generate training data in the UI for both neural networks and decide the number of training samples and the maximum and minimum input and output values in the neural network plant and the maximum and minimum reference value going through the neural network controller. It can also be used to decide how many iterations through the training data the neural networks are going to train and the training algorithm that's going to be used.
- Robot arm block: 4 single link robot arms were used which have different friction values and their motion equation is $\frac{d^2\Phi}{dt^2} = -10sin\Phi - x\frac{d\Phi}{dt} + u$ where Φ is the angle, t is time, x is the friction of the arm and u is the torque supplied by the DC motor.
    - Robot arm with zero friction x= 0:



$$\frac{d^2\Phi}{dt^2} = -10sin\Phi + u$$

Figure 5: Robot Arm with 0 friction

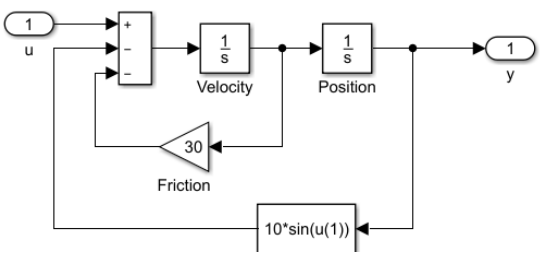
    - Robot arm with friction x= 2:



$$\frac{d^2\Phi}{dt^2} = -10sin\Phi - 2\frac{d\Phi}{dt} + u$$

Figure 6: Robot Arm with friction value 2

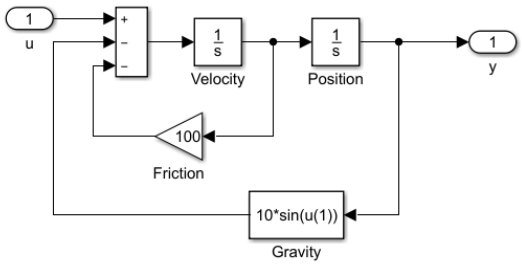
    - Robot arm with friction x= 30:



$$\frac{d^2\Phi}{dt^2} = -10sin\Phi - 30\frac{d\Phi}{dt} + u$$

Figure 7: Robot Arm with friction value 30

- Robot arm with friction x= 100:



$$\frac{d^2\Phi}{dt^2} = -10sin\Phi - 100\frac{d\Phi}{dt} + u$$

Figure 8: Robot Arm with friction value 100

3.3.2. PID controlled System design and parameters

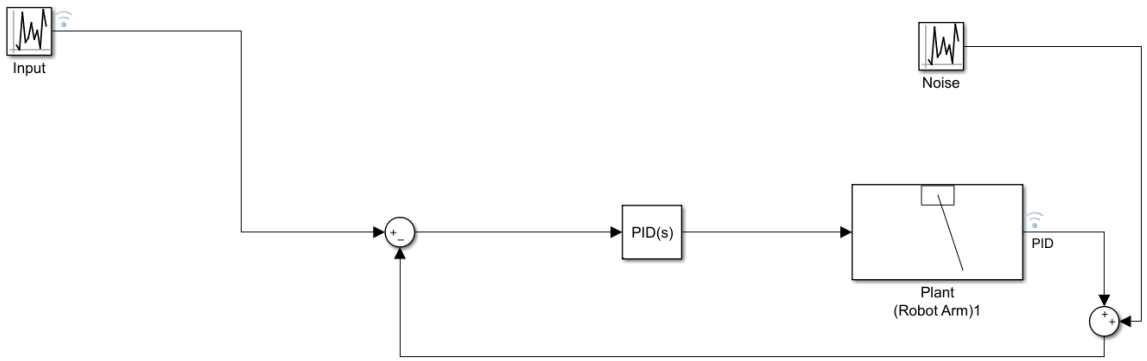


Figure 9: Simulink implementation of the PID controlled system

As shown in figure 9, negative feedback control system is used, where the PID is placed before the arm model block. The PID was tuned using the PID Tuner built in app in MATLAB.

Table 3 shows the values of P, I & D used in the implementation with the resulting rise time, overshoot and settling time.

Friction of the arm	Proportional, Integral & Derivative values	Rise Time	Overshoot	Settling Time
0	P = 161.077839849259 I = 182.787339008204 D =32.6320877633168	0.05 seconds	8.26%	1.35 seconds
2	P = 41.2919734129344 I = 60.1346220505642 D = 6.54807983459208	0.18 seconds	8.01%	1.75 seconds
30	P = 16.1494273053434	2.24 seconds	7.97%	8.27 seconds

	I = 10.3880415618791 D = -1.81974208883492			
100	P = 13.5094856642157 I = 2.7058413528365 D = -4.58978335117057	8.58 seconds	7.9%	30.6 seconds

Table 3: The values given by the PID controller after tuning it

Further increasing of the proportional part will lead to a slightly faster system with increased overshoot. While increasing the value of the integral part will make the system overshoot and oscillate. Moreover, Increasing the derivative part any further will make the system response slow. Thus, the selected values of the different gains show good response with acceptable overshoot and good steady state response.

3.3.3. ANN controlled System design and parameters

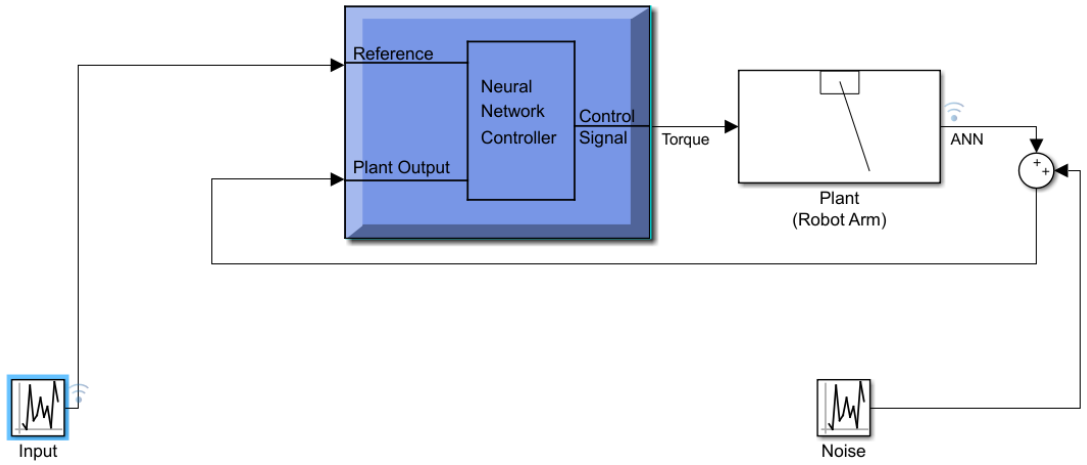


Figure 10: Simulink implementation of the ANN controlled system

As we see in figure 10 the model reference controller block is placed before the robotic arm block. Below are the parameters used for our experiment in the model reference controller:

- The neural network plant parameters:
  - Size of hidden layer: 10
  - Number of plant inputs: 2
  - Number of plant outputs: 2
  - Number of training samples: 10000
  - Maximum plant input: 15
  - Minimum plant input: -15
  - Maximum plant output: 3.1
  - Minimum plant output: -3.1
  - Maximum interval value: 2 seconds
  - Minimum interval value: 0.1 seconds
  - Number of training epochs: 500
- The neural network controller parameters:
  - Size of hidden layer: 10
  - Number of reference inputs: 2
  - Number of controller outputs: 1
  - Maximum reference value: 0.7
  - Minimum reference value: -0.7
  - Maximum interval value: 2 seconds
  - Minimum interval value: 0.1 seconds
  - Number of training samples: 6000
  - The reference model seen in figure 12 that the robot arm will track has the

- Training algorithm: trainlm (Levenberg-Marquardt backpropagation)

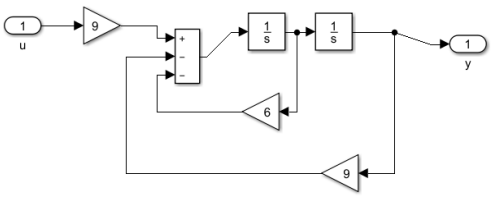


Figure 11: The reference model used

equation $\frac{d^2y}{dt^2} = -9y_r - 6\frac{dy_r}{dt} + 9r$ where y_r is the output of the reference model and r is the input reference signal.

- Number of training epochs: 500
- Number of training segments: 30

3.4. Data generation and collection

The training data for the model reference controller were generated by using the generate training data function in the model reference controller UI.

To obtain the data needed for the comparison, 11 simulation were running in Simulink for each robotic arm. The simulations differed in the noise that was introduced from 0% noise to ±10%. All the simulations were running for 20,000 time unit and every 200 time unit had a different angle. The reason behind choosing 20000 was to run 100 simulated arm movements with 100 randomized commands (20000/200 = 100), and the reason why we chose 200 time units for each command was to give enough time for the arm to reach the target and stabilize.

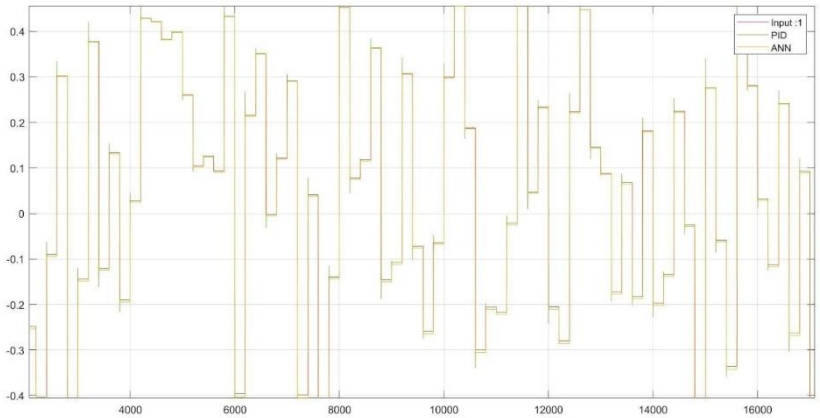


Figure 12: Simulation of the movement of the robotic arm with friction value 2

Figure 12 shows how the simulation looks like after it's ran and shows how the raw data looked like and what type of data we collected. The y-axis represents the angle of the robotic arm, and the x-axis represents the time.
From the obtained data we calculated the error between desired output and the outputs generated by both controllers and the error is used in our results to compare between the two controllers.

4. Results

Like it was mentioned in the Methods episode, 11 simulations for each robotic arm were done, for the noises 0%, ±1%, ±2%, ±3%, ±4%, ±5%, ±6%, ±7%, ±8%, ±9% and ±10%. We chose to show the boxplots to see the individual differences results for 0%, ±5% and ±10% in which they show the significant difference between the two controllers, because the difference between 0% and ±1% is not noteworthy, and the mean square error of the controllers under all feedback noise levels (0% to 10%).

4.1. The robotic arm with zero friction (No Friction)

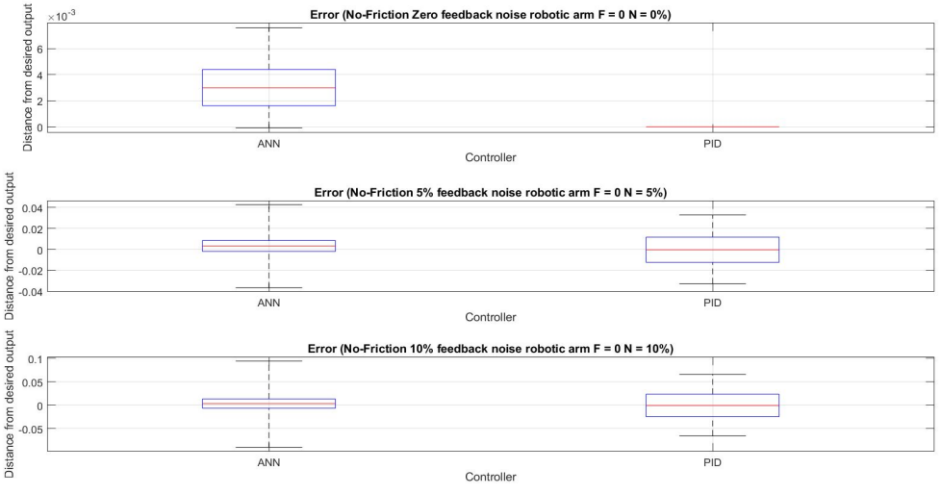


Figure 13: Boxplot of the error for a robot arm with 0 friction with different noise levels

In the environment with no noise in figure 13, the PID outperforms the Model reference controller, as it's seen in figure x where the PID is always on 0 error and the Model reference controller varies a lot relatively.
In the environments with ±5% and ±10% noise in figure 13, the median of the PID data points is very close to zero, but the box's upper bound and lower bound are further away from zero compared to the upper bound and lower bound of the box of the model reference controller where they're closer to zero. The Whiskers of the model reference controller stretch further away from zero than the whiskers of the PID.

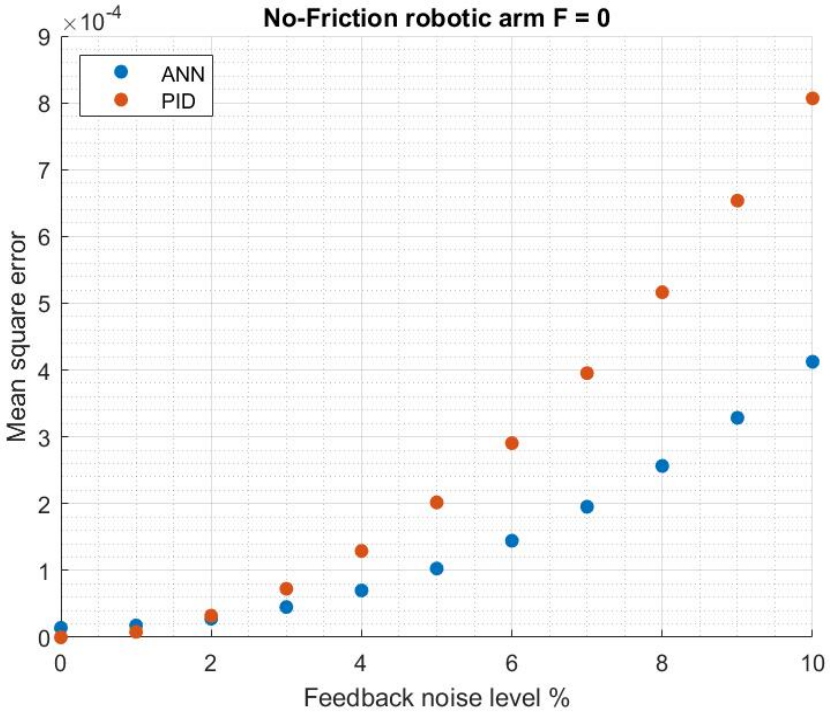


Figure 14: Mean Square Error for a robot arm with 0 friction

Figure 14 shows the mean square error (MSE) of both controllers under different feedback noise levels. The MSE of both controllers increases when the feedback noise level increases but the MSE of the model reference controller (the blue dots) increases relatively slower than the MSE of the PID controller.

4.2. The robotic arm with friction value 2 (Low Friction)

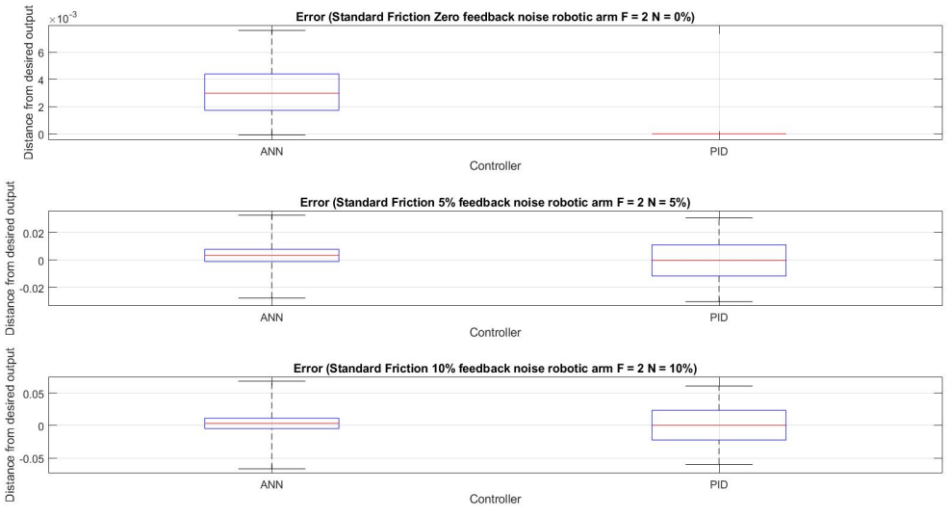


Figure 15: Boxplot of the error for a robot arm with 2 friction with different noise levels

In the environment with no noise in figure 15, the PID outperforms the Model reference controller, as it's seen in figure x where the PID is always on 0 error and the Model reference controller varies a lot relatively.
In the environment with ±5% feedback noise in figure 15, the median of the PID data points is very close to zero, but the box's upper bound and lower bound are further away from zero compared to the upper bound and lower bound of the box of the model reference controller where they're closer to zero. The upper whisker of the model reference controller stretches further away from zero than the upper whisker of the PID, but the lower whisker of the PID stretches further away from zero than the lower whisker of the model reference controller.
In the environment with ±10% feedback noise in figure 15, the median of the PID data points is very close to zero, but the box's upper bound and lower bound are further away from zero compared to the upper bound and lower bound of the box of the model reference controller where they're closer to zero. The Whiskers of the model reference controller stretch further away from zero than the whiskers of the PID.

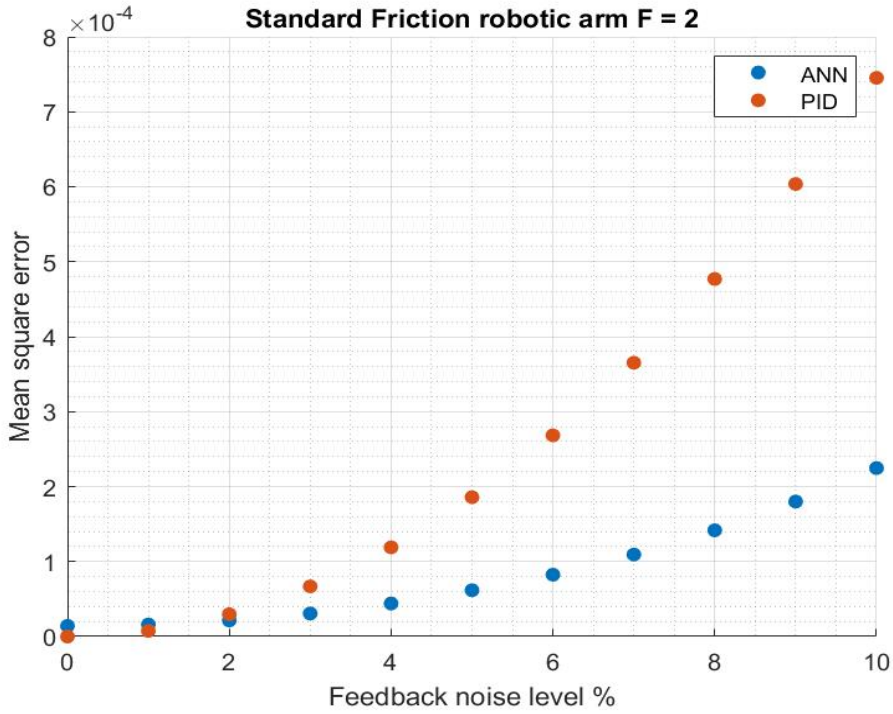


Figure 16: Mean Square Error for a robot arm with 2 friction

Figure 17 shows the mean square error (MSE) of both controllers under different feedback noise levels. The MSE of both controllers increases when the feedback noise level increases but the MSE of the model reference controller (the blue dots) increases relatively slower than the MSE of the PID controller.

4.3. The robotic arm with friction value 30 (Mid-High Friction)

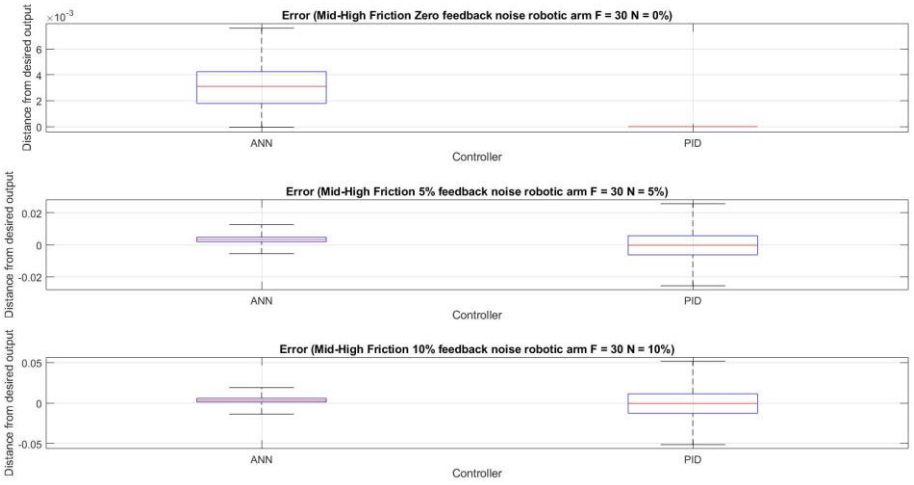


Figure 17: Boxplot of the error for a robot arm with 30 friction with different noise levels

In the environment with no noise in figure 17, the PID outperforms the Model reference controller, as it's seen in figure x where the PID is always on 0 error and the Model reference controller varies a lot relatively.
In the environments with ±5% and ±10% feedback noise in figure 17, the median of the PID data points is very close to zero, but the box's upper bound is further away from zero than the upper bound of the model reference controller, the same goes for the lower bound, but the difference here is that the zero error point is not part of the model reference controller box. But the overall error in the model reference controller is smaller compared to the error in the PID controller.

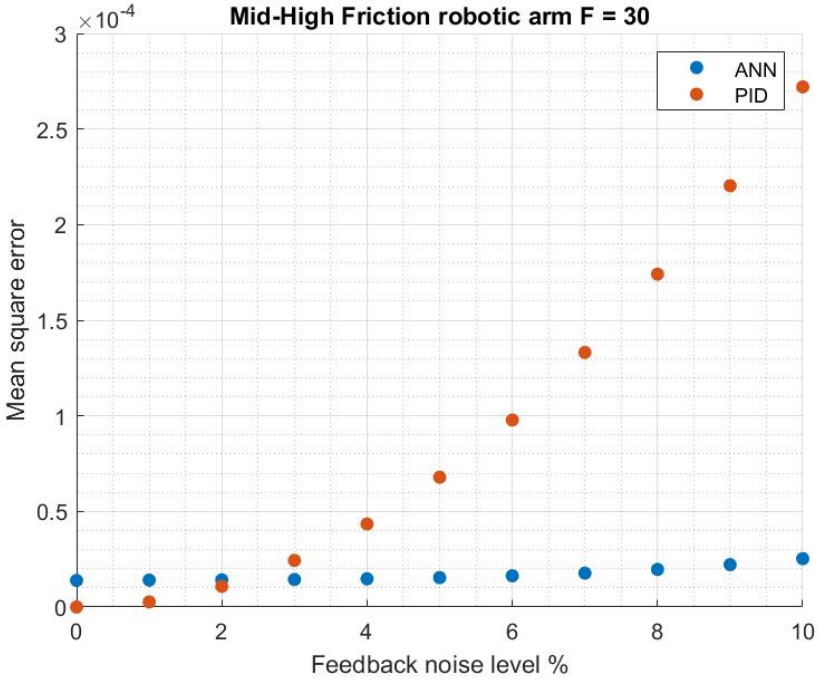


Figure 18: Mean Square Error for a robot arm with 30 friction

Figure 18 shows the mean square error (MSE) of both controllers under different feedback noise levels. The MSE of the PID controller increases relatively fast when the feedback noise level increases. But the MSE of the model reference controller increase is so slow that it is almost insignificant.

4.4. The robotic arm with friction value 100 (High Friction)

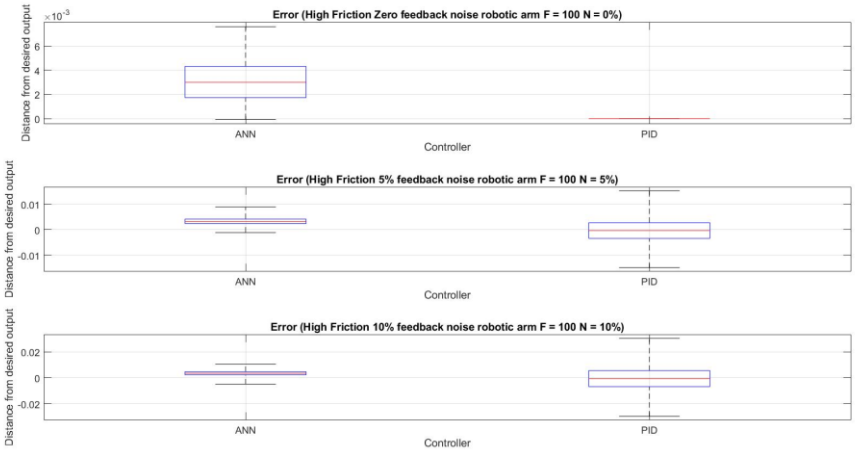


Figure 19: Boxplot of the error for a robot arm with 100 friction with different noise levels

In the environment with no noise in figure 19, the PID outperforms the Model reference controller, as it's seen in figure x where the PID is always on 0 error and the Model reference controller varies a lot relatively.
In the environments with ±5% and ±10% feedback noise in figure 19, the median of the PID data points is very close to zero, but the box's upper bound is further away from zero than

the upper bound of the model reference controller, the same goes for the lower bound, but the difference here is that the zero error point is not part of the model reference controller box. But the overall error in the model reference controller is smaller compared to the error in the PID controller.

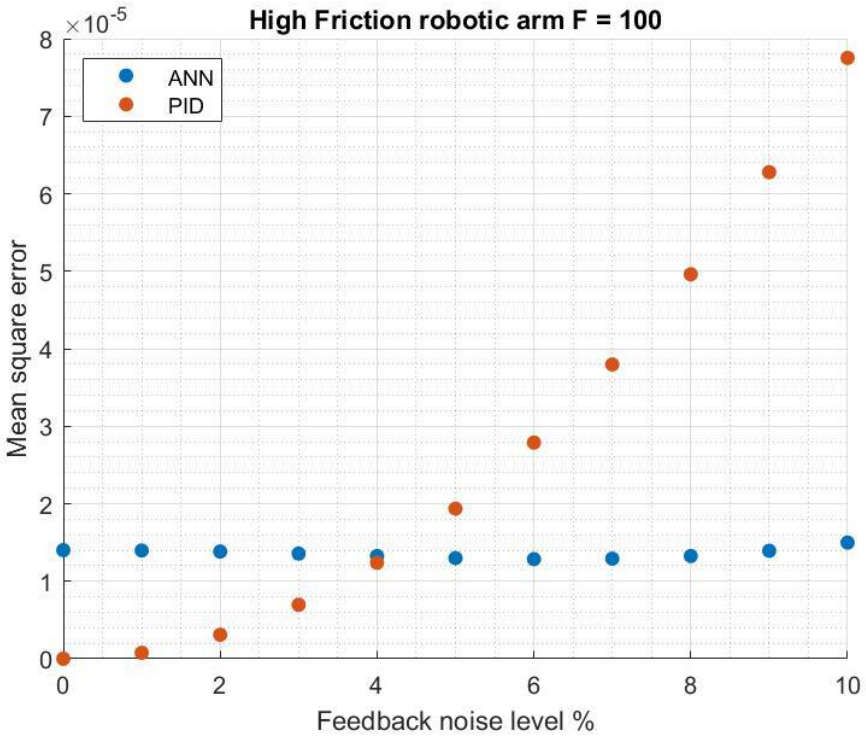


Figure 20: Mean Square Error for a robot arm with 100 friction

Figure 20 shows the mean square error (MSE) of both controllers under different feedback noise levels. The MSE of the PID controller increases fast when the feedback noise level increases. But the MSE of the model reference controller increase is so slow that it is almost insignificant and looks almost the same throughout the graph

5. Discussion

The PID performs exceptionally well in environments without feedback noise, which is expected considering that PID controllers perform great in linear systems like the ones we have. However, nonexistent feedback noise can only be provided under perfect settings, which are hardly found in the real world. We start to see the difference when we introduce feedback noise to the systems.
The results in noisy environment with low and nonexistent friction that we see in figures 13 & 15 show how the box in the model reference controller is smaller and closer to zero compared to the box in the PID controller, which means that the model reference controller is more stable compared to the PID controller and has noise filtering property. However, the median of the PID's data points is almost always on zero which means the PID controller has better accuracy and precision.

The difference between the PID controller and Model reference controller in noisy environments with mid-high and high friction robotic arms is noticeably significant. The model reference controller has a lower overall error, which is seen in figure 18 right at the 3% noise level and afterwards, and in figure 20 where the difference starts showing at the 4% noise level and afterwards where the MSE of the model reference controller is always smaller than the MSE of the PID controller.
The reason behind this is most likely the high friction which makes the arm's movement slower, and the apparent noise filtering ability that is seen in the neural networks which needs even further research to explain, but is clearly seen in figures 18 and 20 where the noise's effect on the model reference controller is so small that it almost looks like the mean square error in each noise level on the graph are on the same line, as in no big change between the noisy and non-noisy environments.

The PID controller performs poorly in these settings because it is a static mathematical equation in a closed loop system that is dependent on negative feedback which sends out corrupted commands when noise is introduced to the feedback which is clearly seen in the figures 14, 16, 18 and 20.
So, to say that the neural network controller gets better when noise is introduced is technically wrong. The PID controller performs worse when feedback noise is applied to the system.

6. Conclusions

- PID is always more precise when it comes to non-noisy environments.

- There’s no significant difference between Model reference controllers and PID controllers when it comes to robotic arms with low and nonexistent friction in noisy environments, as 50% of the model reference controller data points are closer to zero but the MSE of the PID controller is bigger compared to the model reference controller, but the difference is not that significant and can be called trivial, still technically the model reference controller performed better.

- Model reference controller is at its best in robotic arms with mid-high and high friction in noisy environments as the results show better precision and stability in Model reference controller compared to PID.

- Model reference controller tends to be rounder (and tries to play it safe) as the results shows a tendency towards stability but not towards precision which is noticed in figures 15, 17, 19 and 21 where the MSE is never zero and the zero error is not part of the box is not part of the box in figures 18 and 20.

References

[1] M. Aamir, “On replacing PID with ANN controller for DC motor position control,” *International Journal of Research Studies in Computing, vol. 2,* vol. 2, no. 1, 2013.

[2] M. Muna and M. A. Akbar, “Simulation of Fuzzy Logic Control for DC Servo Motor using Arduino based on Matlab/Simulink,” in *International Conference on Intelligent Autonomous Agents, Networks and Systems*, 2014.

[3] M. King, Process control: a practical approach, John Wiley & Sons, 2016.

[4] Š. Bucz and A. Kozáková, “Advanced Methods of PID Controller Tuning for Specified Performance,” in *PID Control for Industrial Processes*, 2018.

[5] J. G. Ziegler and N. B. Nichols, “Optimum Settings for Automatic Controllers,” *Journal of Dynamic Systems Measurement and Control-transactions of The Asme,* vol. 115, no. , pp. 220-222, 1993.

[6] Y. Li, K. H. Ang and G. C. Y. Chong, “Patents, software, and hardware for PID control: an overview and analysis of the current art,” *IEEE Control Systems Magazine,* vol. 6, no. 1, 2006.

[7] S. Kal, in *Basic electronics: Devices, circuits and IT fundamentals*, Prentice-Hall of India, 2012, pp. 192-210.

[8] G. N. Yannakakis and J. Togelius, Artificial Intelligence and Games, Springer, 2018.

[9] M. A. Nielsen, , Neural Networks and Deep Learning, Determination Press, 2015.

[10] E. Süli and D. Mayers, An Introduction to Numerical Analysis, Cambridge University Press, 2003.

[11] A. Björck, Numerical methods for least squares problems, Philadelphia: SIAM, 1996.

[12] “Reducing Loss: Gradient Descent,” Google, 5 March 2019. [Online]. Available: https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent.

[13] I. Goodfellow, Y. Bengio and A. Courville, in *Deep Learning*, MIT Press, 2016, p. 196.

[14] K. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares,” *Quarterly of Applied Mathematics,* no. 2, p. 164–168, 1944.

[15] D. Marquardt, “An Algorithm for Least-Squares Estimation of Nonlinear Parameters,” *SIAM Journal on Applied Mathematics,* no. 11, p. 431–441, 1963.

[16] R. A. Khalil, “Neural Network Based on Model Reference Using for Robot Arm,” *Al-Rafidain Engineering,* vol. 22, no. 4, pp. 100-109, 2014.

[17] “Mathworks MATLAB,” Mathworks, 2019. [Online]. Available: https://se.mathworks.com/products/matlab.html.

[18] “Mathworks Simulink,” Mathworks, 2019. [Online]. Available: https://se.mathworks.com/products/simulink.html?s_tid=hp_products_simulink.

[19] R. Reicherdt and S. Glesner, “Slicing MATLAB Simulink models,” in *34th International Conference on Software Engineering (ICSE)*, 2012.

TRITA-EECS-EX-2019:398