UNIVERSITY OF
WATERLOO

Department of Mechanical and Mechatronics Engineering

Design of a PID Auto-Tuner for an Automatic Levelling Wrist

Author:
James Graham-Hu

Student Number:
20555690

Date: September 14, 2019

September 14, 2019

William Melek, Director
Mechatronics Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear Professor Melek,

This report entitled ”Title”, was prepared as my Work Report 400 for the Department of Mechanical and Mechatronics Engineering at the University of Waterloo for the 4A term. Purpose of report.

Description of Amii.

Description of project and motivation behind project.

Thank yous. This report was written entirely by me and has not received any previous academic credit at this or any other institution.

Sincerely,

James Graham-Hu
ID 20555690
4A Mechatronics Engineering

Contents

List of Figures

List of Tables

1 Summary

Summary

2 Introduction

2.1 Problem Definition

Demos can provide an interesting and intuitive way to demonstrate the capabilities and advantages of machine learning to interested parties. Although a few software demos exist at Amii, there is currently no demo that implements machine learning on a hardware system. The advantage of implementing machine learning on hardware is that it is a hands-on way of demonstrating how machine learning can improve a system or be applied to a problem. The problem this project addresses is the lack of a hardware demo to demonstrate machine learning at Amii.

2.2 Objective

The automatic levelling wrist, developed by Dylan A. J. Brenneis as part of his Master's thesis in 2019, [4] is a good candidate for a hardware system that can be improved with machine learning. The objective of this project is to use machine learning to improve the automatic levelling wrist in a meaningful and demonstrable way.

3 Background

3.1 Automatic Levelling Wrist Background

Powered wrist movement is rare in commercial systems, and many powered protheses have only one degree of freedom (DOF), usually rotation [1]. These limitations in ease of wrist movement in many upper limb prostheses force people with major upper limb loss to use compensatory movements [2]. Compensation occurs with trunk, shoulder, and elbow movements, and has been associated with causing musculoskeletal pain in the neck, upper back, shoulder, and remaining arm [3].

A two DOF automatic levelling wrist was developed by Dylan J. A. Brenneis in 2019 that addresses the issues with ease of wrist movement in wrist prosthetics [4]. The two DOF automatic levelling wrist provides two degrees of freedom, rotation and flexion of the wrist. The user has the ability to switch between controlling the position of the flexion, or letting it automatically level itself to maintain its angle with the ground (see figure 1). The rotation of the wrist is always automatically leveled to be flat with the ground. These features are shown to reduce compensatory movements in vertically-oriented tasks. However, the current implementation of the automatic levelling wrist is reported as unreliable and unintuitive in user tests [4]. Possible reasons for a feeling of unreliability from users could be a result of slow response time, oscillations, and poor disturbance rejection in the automatic levelling system.

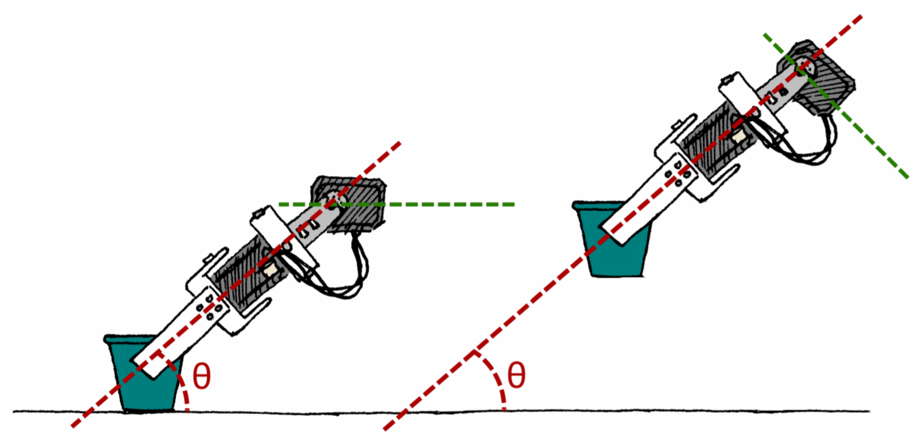

Figure 1: Automatic levelling of the flexion. The angle, θ, of the flexion servo is constant with the ground as the arm moves (represented by the green dashed line) [4].

Figure 2 shows the design of the automatic levelling wrist. Three MX-28AT servos are used to actuate the prosthetic wrist. Two servos provide the degrees of freedom for wrist rotation and flexion, and the third servo is used to manipulate a gripper. It should be noted that the automatic levelling wrist is designed as a bypass protheses so that an able-bodied person is able to use it. A higher statistical power is able to be achieved in a more time efficient way by running trials with able-bodied persons, because of limitations in participant availability when running trials with participants affected by amputation [4].

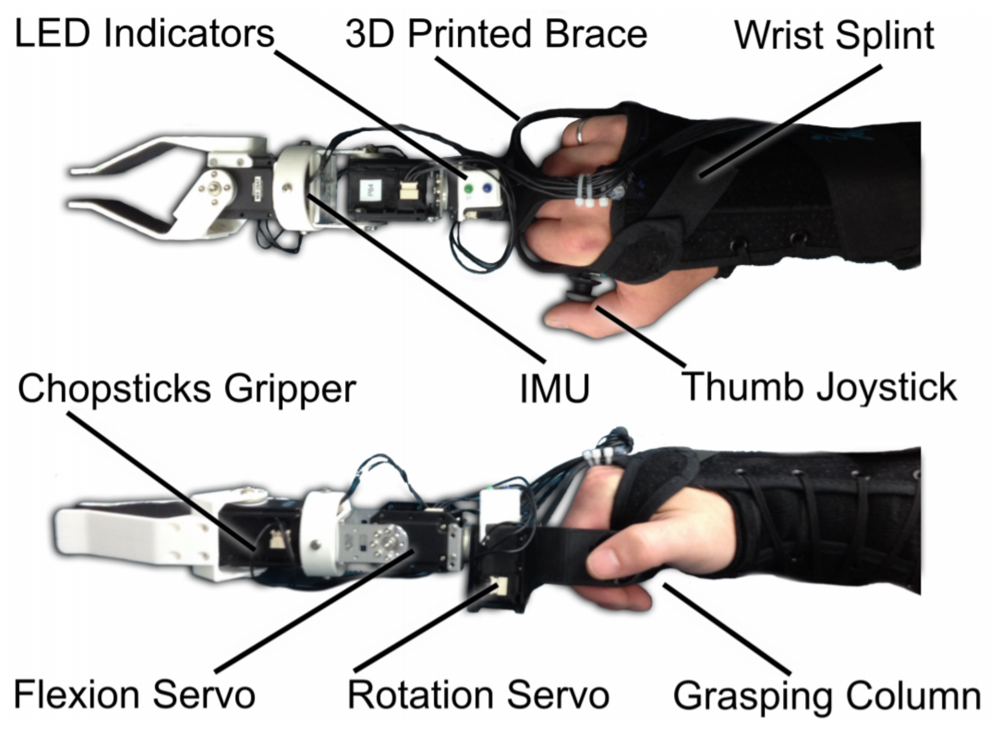

Figure 2: Diagram of the automatic levelling wrist bypass protheses [4].

3.1.1 Automatic Levelling Method

The wrist is automatically leveled using an Adafruit 9-DOF absolute orientation IMU fusion breakout (BNO055) attached to the base of the gripper, as seen in figure 2, and two independent PID controllers, one for the rotation servo and one for the flexion servo. The gravity vector from the IMU is used to calculate the current angle of rotation and flexion. The error between the calculated angle and the setpoint is fed into the PID controller to generate a control signal for the servo. The setpoint for the rotation servo is always set to $180°$ while the setpoint for the flexion servo is set to where the user last moved it to before switching to automatic levelling. Figure 3 shows the definitions for the coordinate system, the angle of rotation, ϕ, and the angle of flexion, θ.

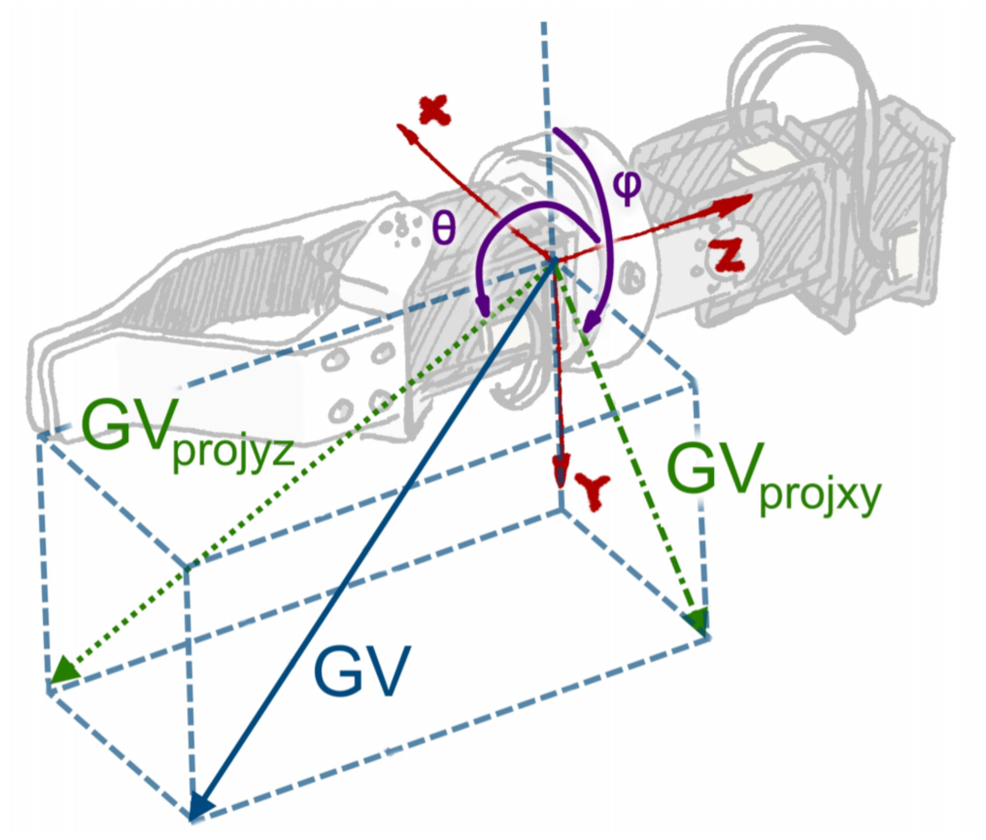

Figure 3: Coordinate system and angles, ϕ and θ. The projections of the IMU gravity vector (GV) can be used to calculate ϕ and θ [4].

ϕ is defined as the angle between the negative y-axis and the projection of the gravity vector in the x-y plane. θ is defined as the angle between the positive z-axis and the projection of the gravity vector projected in the y-z plane.

Figure 4 shows the block diagram for one servo in the automatic levelling wrist. A simple PID control loop controls the position of the servo. The equation for the PID controller signal, $u(t)$, is given by equation 1. The servo position is summed with a disturbance, $d(t)$ which models the angle the wrist is rotated by the user, and fed through the IMU to acquire either ϕ or θ. The block diagram for the rotation and flexion servos are the same other than the K_p, K_i, and K_d gain values for the PID controllers, the IMU function, and the value of the disturbance, $d(t)$. The disturbance is considered as the angle of the user's wrist with respect to a fixed coordinate system that is coincident with the user's wrist when it is completely level with the ground. Note that in the actual implementation of the control loop, the disturbance and servo position aren't directly used. Rather, they are implicitly included when the IMU measures the gravity vector since the gravity vector changes based on the disturbance and servo positions.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \tag{1}$$

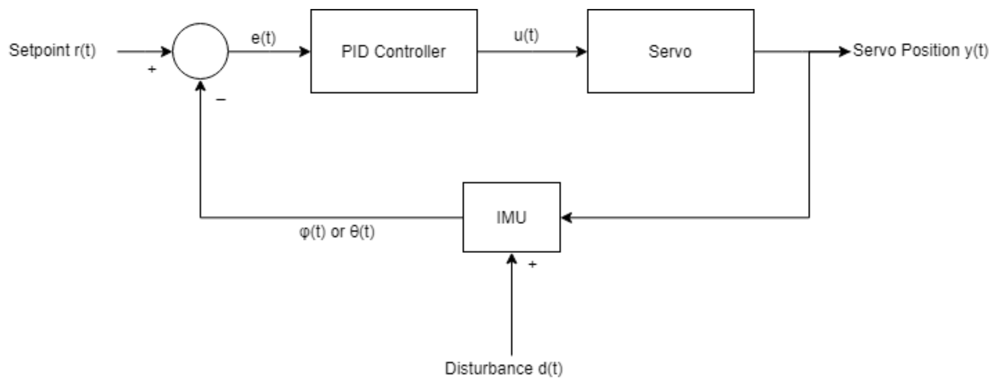

Figure 4: Block diagram of the control loop for one servo.

3.2 Neural Network Background

In machine learning, neural networks are used to represent non-linear functions by making use of one or more hidden layers and activation functions. Figure **??** shows the structure of a neural network with one hidden layer. In this network, the input vector, X, is multiplied by the weight matrix W_0 and summed with the bias vector b_0. The result of that is then put through an activation function, giving a vector of activations. These activations are then multiplied by the weight matrix W_1 and summed with the bias vector b_1 to give the output, y. The output can also be put through an activation function. In regression, a linear activation function is mostly used on the output. Activation functions used in the hidden layer include but are not limited to the hyperbolic tangent, sigmoid, and rectified linear unit (ReLU) functions. The choice of activation function is dependent on the application of the neural network.

3.2.1 Backpropagation

Learning for a neural network is most commonly accomplished by backpropagation. Backpropagation is performed by calculating the derivative of the squared error of the neural network with respect to each weight in the network. The squared error of the neural network is defined in equation 2. The equation is divided by two to simplify the derivative. Since the entire network including its activation functions is differentiable, the chain rule can be used to backpropagate the error between the neural network's output and what the output should be (the label) back to the weights. Equation 3 is the update function used for weight $w_{i,j}$ in the neural network which updates the weights at a learning rate α.

$$E = \frac{1}{2}(y_{true} - y_{pred})^2 \tag{2}$$

$$w_{i,j} \leftarrow w_{i,j} + \alpha \frac{\partial E}{\partial w_{i,j}} \tag{3}$$

3.2.2 Levenberg-Marquardt Algorithm

Another method for training neural networks is using the Levenberg-Marquardt algorithm. The Levenberg-Marquardt algorithm uses the jacobian of the output with respect to the weights of the neural network to solve for an adjustment for each weight. Equation 4 is solved for σ to determine the adjustments for each weight.

$$[J_w^T J_w + \lambda I]\sigma = J_w^T[E] \tag{4}$$

Here, J_w is the jacobian of the output with respect to the weights, λ is the damping factor, I is the identity matrix, σ is the adjustment matrix, and E is the difference between the outputs and the labels. Isolating for σ yields equation 5. The damping factor, λ, is increased if $[J_w^T J_w + \lambda I]$ is not invertible. Once σ is acquired, it is added to the previous weights to move them closer to the optimal values. This process is repeated iteratively until the weights converge.

$$\sigma = [J_w^T J_w + \lambda I]^- 1 J_w^T[E] \tag{5}$$

For a system, $y(\mathbf{u_i}, \mathbf{w})$ with one output, an input vector, $\mathbf{u_i}$, n timesteps, and k weights, the jacobian with respect to the neural network weights is expressed as equation 6.

$$J_w = \begin{bmatrix} \frac{\partial y(\mathbf{u_0},\mathbf{w})}{\partial w_0} & \frac{\partial y(\mathbf{u_0},\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial y(\mathbf{u_0},\mathbf{w})}{\partial w_k} \\ \frac{\partial y(\mathbf{u_1},\mathbf{w})}{\partial w_0} & \frac{\partial y(\mathbf{u_1},\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial y(\mathbf{u_1},\mathbf{w})}{\partial w_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y(\mathbf{u_n},\mathbf{w})}{\partial w_0} & \frac{\partial y(\mathbf{u_n},\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial y(\mathbf{u_n},\mathbf{w})}{\partial w_k} \end{bmatrix} \tag{6}$$

3.2.3 Numerically Calculating the Jacobian of the Servo

To numerically calculate jacobian of a servo, each weight is individually changed by a small amount, ϵ and then an input signal is sent to the servo. The change in servo response before and after the small weight change is recorded and used to calculate a finite backward difference to approximate the partial derivative of the system with respect to the weight that was changed. Equation 7 demonstrates this calculation of the approximate jacobian matrix. The dimension of the resultant jacobian matrix is n rows by k columns.

$$
\begin{aligned}
&J_w = \begin{bmatrix} \frac{\partial \mathbf{y}(\mathbf{u},\mathbf{w})}{\partial w_0} & \frac{\partial \mathbf{y}(\mathbf{u},\mathbf{w})}{\partial w_1} & \dots & \frac{\partial \mathbf{y}(\mathbf{u},\mathbf{w})}{\partial w_k} \end{bmatrix} \\
&\frac{\partial \mathbf{y}(\mathbf{u},\mathbf{w})}{\partial w_i} = \frac{\mathbf{y}(\mathbf{u},\mathbf{w}) - \mathbf{y}(\mathbf{u},\mathbf{w} - \mathbf{h}\epsilon(\mathbf{w_i}))}{\epsilon(w_i)} \\
&h_p = 1, p = i \\
&h_p = 0, p \neq i
\end{aligned}
\tag{7}
$$

3.3 Amplitude Modulated Pseudo-Random Binary Signal

An amplitude modulated pseudo-random binary signal (APRBS) can be used to excite a system such that all frequencies are excited equally [5]. The signal approximates the spectrum and statistical properties of white noise, as it has a mean of one and variance of zero. The signal is also deterministic, making it consistent between trials. The method used to generate an APRBS in this project is outlined in Johannes Günther's paper [6]. Figure 5 is an example of an APRBS with a maximum amplitude of 1, 500 step length, and maximum hold time of 100 steps. The orange signal is the same APRBS with a first order butterworth filter applied to make the edges less ”sharp”.

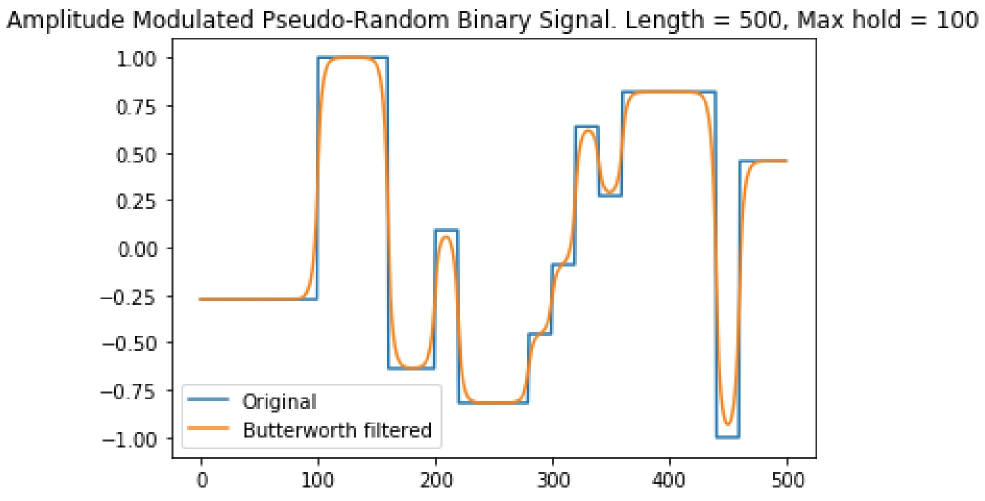

Figure 5: APRBS with a maximum amplitude of 1, 500 step length, and maximum hold time of 100 steps. With and without first order butterworth filter.

4 Proposed Solutions

4.1 Criteria and Constraints

The proposed solutions for the project were constrained according to the following constraints.

- The solution must improve the performance of the automatic levelling wrist.
- The solution must implement machine learning in some way.

The following criteria considered in choosing an appropriate solution are chosen such that the automatic levelling wrist reliably improves the user's ability to use the prosthetic.

- The solution should minimize steady state error.
- The solution should minimize response time.
- The solution should minimize settling time.
- The solution should maximize reliability and consistency.

Due to the nature of the problem, training time and data are difficult to acquire. Without a simulation, data and training can only be collected and run in real time. Therefore, the solution should require minimal training time and data, or be able to make use of a simulation.

4.2 Neural Network Methods

The three neural network methods proposed in this section make use of the PID control loop described in Figure 4. A neural network that outputs the PID gains, K_p, K_i, and K_d is added to the control loop. The neural network takes the error between the setpoint and the current angle of the servo, the current angle of the servo, and the velocity of the servo. The goal is to train the neural network so that it will output appropriate gains for any situation, such that the system has the best response time and error for that situation. The rotation and flexion servos will use separately trained neural networks to output gains for their respective PID controllers. In this section the rotation servo will be used to explain the method, but the same method can be applied to both rotation and flexion servos.

4.2.1 PID Auto-Tuning, Trained Model-Free using Backpropagation

Since no labels exist for the PID gains at the output of the neural network (it is unknown what the gains should be in any given situation) the error term used to train the neural network, E, must be the squared error between the servo angle, $\phi(t)$ and the setpoint, $r(t)$ as shown in equation 8. To train the neural network, this error must be backpropagated through the servo and the PID controller to the neural network output, $\mathbf{m}$. Performing the chain rule on the partial derivative of E with respect to $\mathbf{m}$ yields 9. This partial derivative can then be backpropagated through the neural network to train the weights.

$$E = \frac{1}{2}(r - \phi)^2 \tag{8}$$

$$\frac{\partial E}{\partial \mathbf{m}} = \begin{bmatrix} \frac{\partial E}{\partial \phi}\frac{\partial \phi}{\partial u}\frac{\partial u}{\partial K_p} \\ \frac{\partial E}{\partial \phi}\frac{\partial \phi}{\partial u}\frac{\partial u}{\partial K_i} \\ \frac{\partial E}{\partial \phi}\frac{\partial \phi}{\partial u}\frac{\partial u}{\partial K_d} \end{bmatrix} \tag{9}$$

Here, ϕ is the rotation servo angle, u is the control signal from the PID controller defined in equation 1, and K_p, K_i, and K_d are the PID gains. The solutions to the individual partial derivatives are shown in equation 10

$$\begin{aligned} \frac{\partial E}{\partial \phi} &= r - \phi \\ \frac{\partial \phi}{\partial u} &= ? \\ \frac{\partial u}{\partial K_p} &= e(t) \\ \frac{\partial u}{\partial K_i} &= \int e(t)dt \\ \frac{\partial u}{\partial K_d} &= \frac{de(t)}{dt} \end{aligned} \tag{10}$$

Since the time domain function of the servo, $\phi(t)$ is not known, it is not possible to solve for $\frac{\partial \phi}{\partial u}$. To address this problem, a second neural network that mimics the servo is used to output the servo angle rather than the actual servo. This allows the error to be backpropagated through the neural network, through the PID controller and to the PID neural network. Figure 6 is the control loop block diagram with the added PID neural network and servo neural network. The servo neural network will be trained on the actual response of the servo. By treating the input to the servo as the input to the neural network, and the output of the servo as the labels for training, backpropagation can be performed to train the servo neural network on a dataset of inputs and outputs given by the actual servo. Figure 7 demonstrates the training process for the servo neural network. Once trained, the servo neural network is implemented into the control loop to train the PID neural network using backpropagation. After training the PID neural network, the PID neural network is implemented into the actual system.

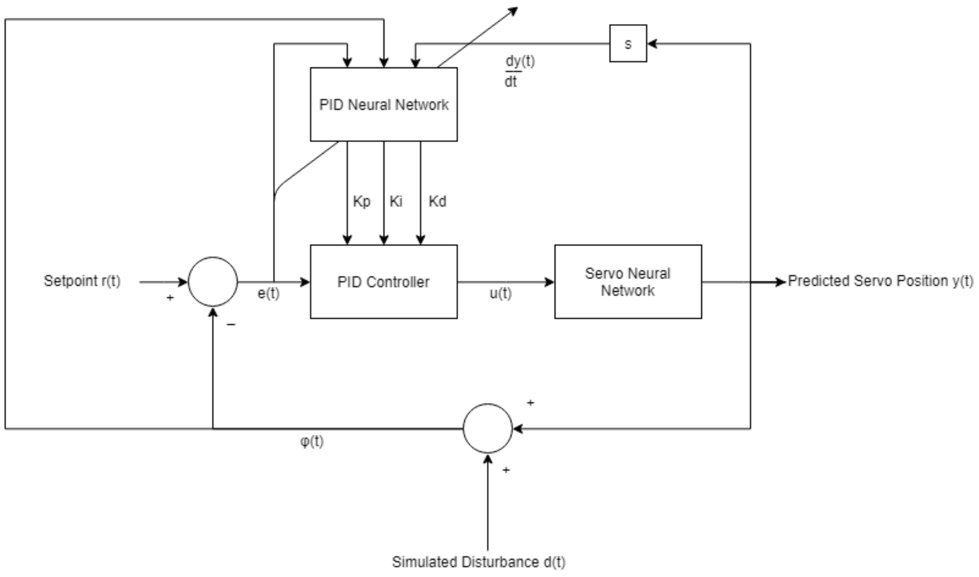

Figure 6: Block diagram of the training process for the PID neural network using the servo neural network to mimic the actual servo output.

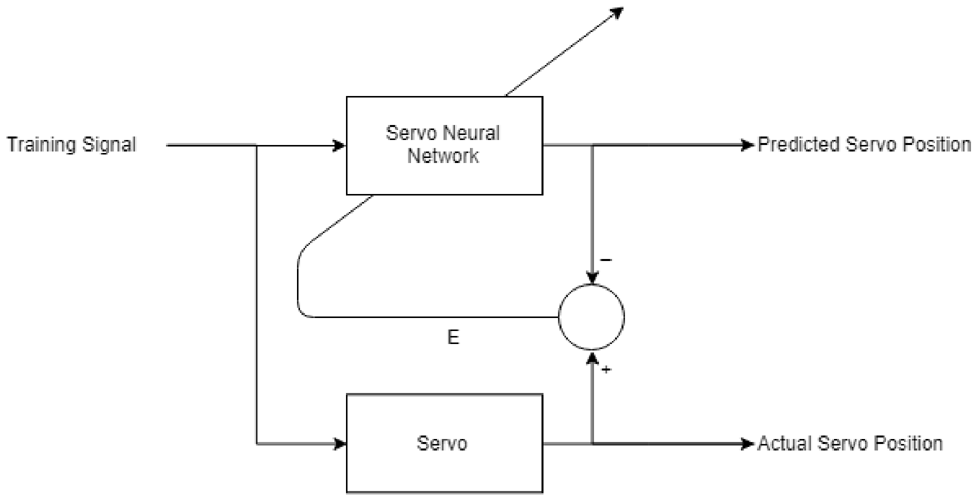

Figure 7: Block diagram of the training process for the servo neural network.

4.2.2 PID Auto-Tuning, Trained Model-Free using the Levenberg-Marquardt Algorithm

Rather than using backpropagation to train the weights of the PID neural network, the Levenberg-Marquardt algorithm is used to train the PID neural network as described in section 3.2.2. Equation 11 is used to solve for the weight adjustments, σ, using the difference between the servo angle, ϕ, and the setpoint, r, as the error term. A servo neural network, trained the same way as described in 4.2.1 is used to simulate the servo response to small weight changes needed to calculate the jacobian of the servo outlined in section 3.2.3.

$$\sigma = [J_w^T J_w + \lambda I]^- 1 J_w^T [\phi - r] \tag{11}$$

4.2.3 PID Auto-Tuning, Trained with a Model using the Levenberg-Marquardt Algorithm

Similar to section 4.2.2 this method uses the Levenberg-Marquardt algorithm the train the PID neural network. However, when numerically calculating the jacobian, rather than using a neural network to simulate the servo response to small weight changes, a mathematical model of the servo is used. Equation 12 is the simplified transfer function of the servo, assuming the electrical dynamics of the servo are much faster than the mechanical dynamics [8].

$$\frac{Y(s)}{U(s)} = \frac{N\eta K_t(K_d s^2 + K_p s + K_i)}{R(J_l + J_m N^2\eta)s^3 + (b_m N^2 \eta R + \frac{K_t N^2 \eta}{K_\omega} + N\eta K_t K_d)s^2 + N\eta K_t K_p s + N\eta K_t K_i} \tag{12}$$

Here, $Y(s)$ is the laplace transform of the servo angle, $U(s)$ is the laplace transform of the input signal. The MX-28AT servo contains an internal PID controller with internal gains. Since the servo is controlled in ticks rather than radians, the gains must be converted so that they can be used in the transfer function which is in radians. Equation 13 shows the conversion process.

$$\begin{aligned} K_p &= \frac{P}{8}\frac{2048V_s}{511\pi} \\ K_i &= \frac{1000I}{2048}\frac{2048V_s}{511\pi} \\ K_d &= \frac{4D}{1000}\frac{2048V_s}{511\pi} \end{aligned} \tag{13}$$

Here, V_s is the source voltage which is $12V$, and P, I, and D are gains set by the user, which are $P = 32$, $I = 0$, and $D = 0$ by default. The user manual for the MX-28AT also states that the gains set by the user must be converted to get their values in ticks, hence the multiplication by $\frac{1}{8}$, $\frac{1000}{2048}$ and $\frac{4}{1000}$ for P, I, and D respectively [9].

The load inertia, J_l, depends on the servo. For the rotation servo the inertia is estimated as the inertia of a rectangular prism, which represents the flexion servo, rotating on its center axis, and a point mass, which represents the gripper, whose position above the axis of rotation depends on the angle of the flexion servo. For the flexion servo the inertia is estimated as the inertia of a rectangular prism, which represents the gripper, rotating around an axis offset from its center by half its length and the length of the bracket attaching the gripper to the flexion servo. Although these assumptions oversimplify the inertia, the load inertia, J_l, is relatively small compared to the other term, $J_m N^2 \eta$, in the s^3 term of the denominator (J_l is on an order of magnitude of 10^-4 and $J_m N^2 \eta$ is on an order of magnitude of 10^-3). Therefore, any error incurred due to these assumptions will be small. The inertia calculations are shown in equation 14.

$$\begin{aligned} J_{l,rot} &= m\frac{b^2+h^2}{12} + (l+c)^2 sin(\theta)^2 \\ J_{l,flex} &= m(d^{2/3} + dl + l^2) \end{aligned} \tag{14}$$

Here, m, is the mass of the MX-28AT servo, b is the width of the servo, h is the height of the servo, d is the depth of the servo, l is the length of the bracket connecting the gripper to the flexion servo, and c is the center of mass of the gripper.

The other parameters for equation 12 are described in table 1.

Table 1: Parameters for the MX-28AT servo model [8]

Parameter	Symbol	Value
Motor resistance	R	8.3Ω
Motor gear ratio	N	193
Motor gear efficiency	η	0.836
Speed constant	K_ω	$93.1 rad/V$
Torque constant	K_t	$0.0107 Nm/A$
Motor inertia	J_m	$8.68 \times 10^{-8} kgm^2$
Friction	b_m	$8.87 \times 10^{-8} Nms$

To account for voltage saturation of the power supply at $12V$, the input signal is saturated so that it doesn't exceed this limit. As an initial guess, the saturation limit is set to $10°$. This means that the control signal, $u(t)$, is saturated at $-10°$ and $10°$.

Figure 8 is the control loop block diagram with the PID neural network and the servo model for training. Once training of the PID neural network is complete, the neural network weights can be used to generate the PID gains for the actual servo in the physical implementation.

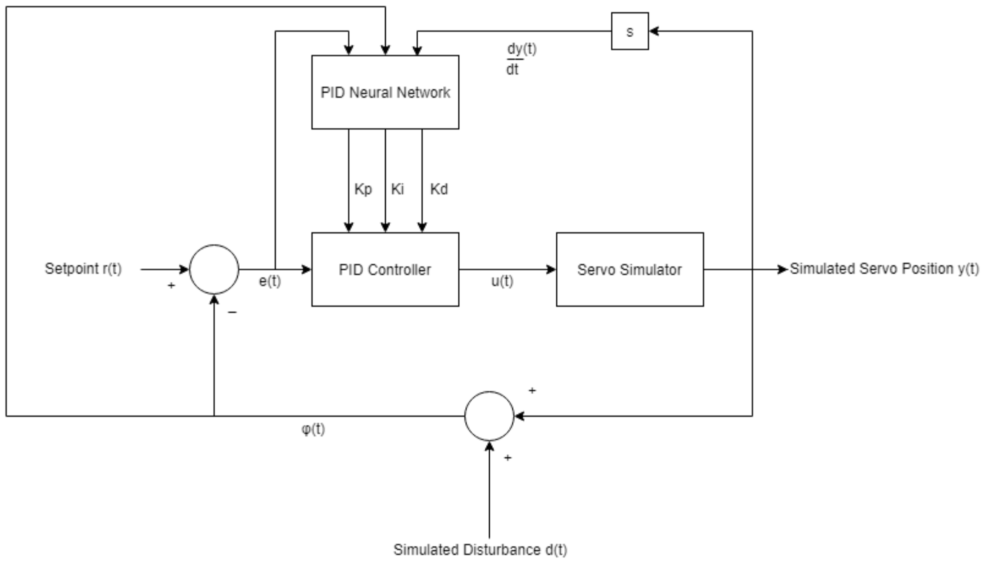

Figure 8: Block diagram of the training process for the PID neural network using the servo model.

5 Solution Evaluation

Judging the three proposed solution methods without implementing them shows that they each have the potential to improve the performance of the automatic levelling wrist according to the criteria. The deciding factor is the amount of data needed to train each solution method. For the model-free methods, the methods that use the servo neural network, real servo data is needed. Although servo data can be recorded and reused, many data points are needed to train the neural network in order for the neural network to learn the servo's time-domain function. A neural network is trained on a 8000 data point training set and its performance is tested on a 2000 data point test set. The training and test data consist of an APRBS with a maximum amplitude of $\frac{\pi}{2}$ radians as the input, and the actual servos output as the label. Data is collected at a rate of 5ms per time step. Figure 9 shows that the neural network's prediction on the test set is close to the actual servo.

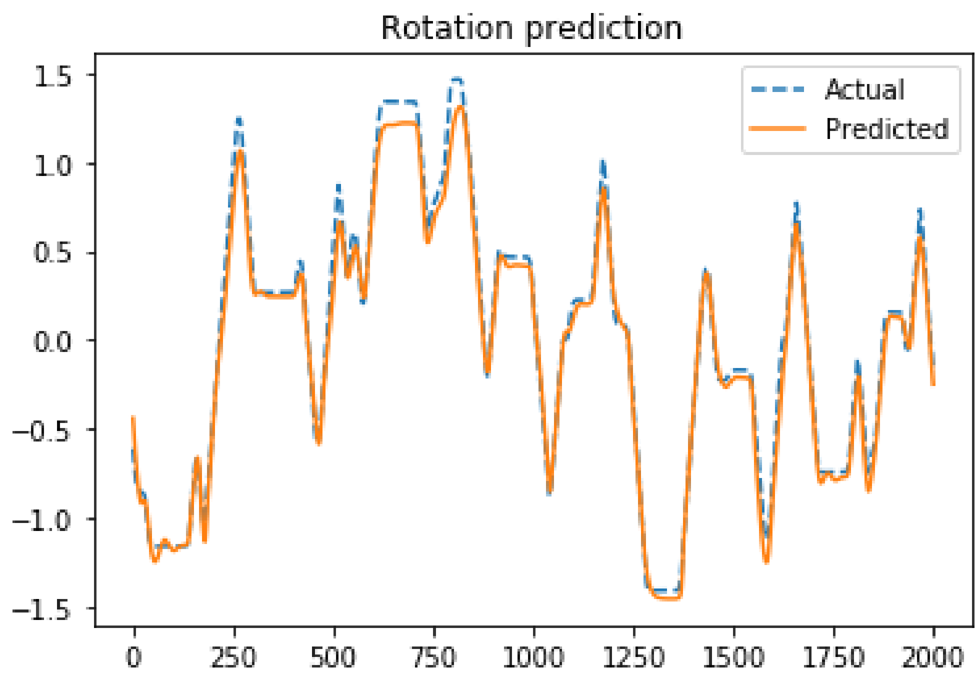

Figure 9: Actual and predicted servo position for APRBS with a maximum amplitude of $\frac{\pi}{2}$ over 2000 time steps, at 5ms per time step.

Figure 10 shows the results of testing the neural network on a $\frac{\pi}{2}$ step input. While the response somewhat resembles what the actual servo's response, it does not perfectly track the input.

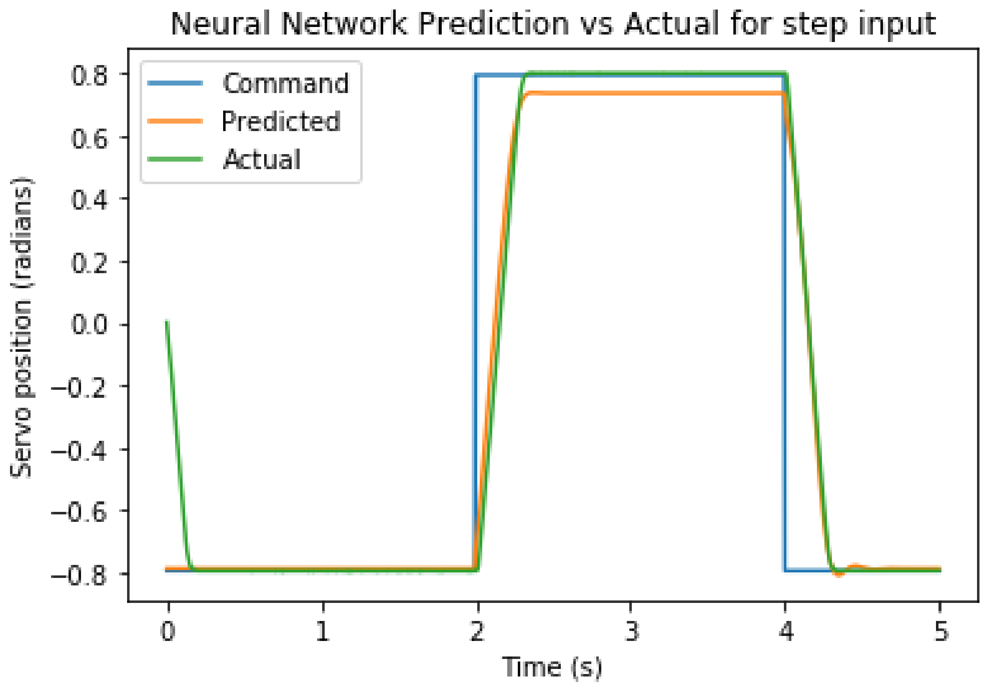

Figure 10: Neural network prediction vs actual for the step input.

These problems could potentially be rectified by using more training data. However, the results for the mathematical model are promising, without the need to collect data. Figure 11 shows the same test as the one ran on the neural network model. It shows that the mathematical model can closely simulate the servo.

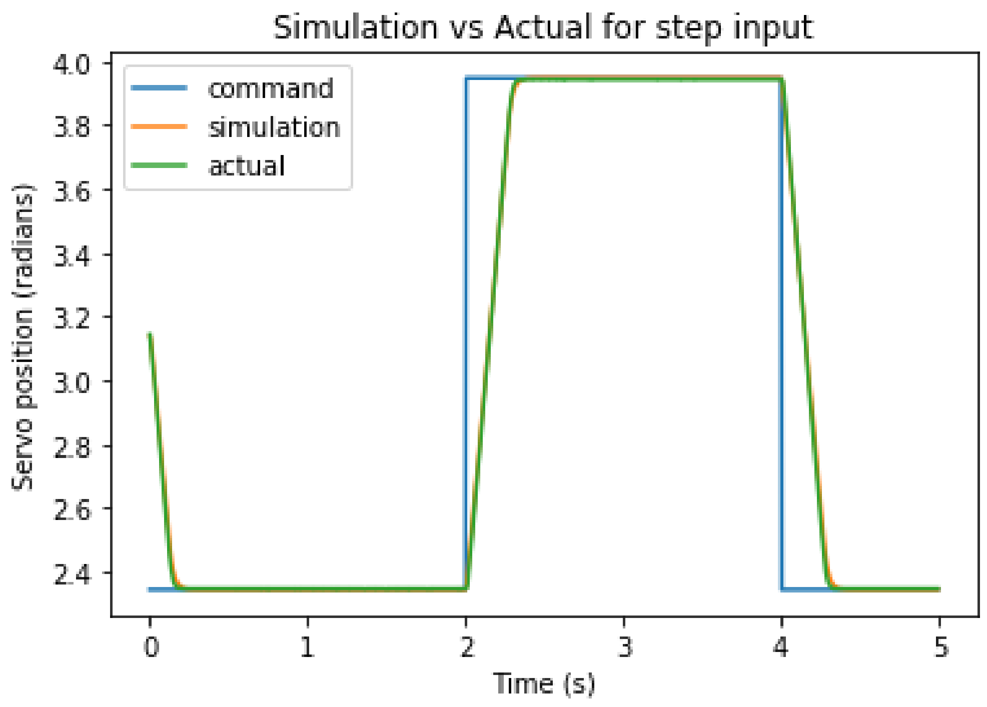

Figure 11: Simulation vs actual for the step input.

Since no data is needed for mathematical model and the mathematical model can closely simulate the actual servo, the chosen solution method is PID auto tuning trained with a model using the Levenberg-Marquardt algorithm.

6 PID Auto-Tuning Implementation

The control software for the automatic levelling wrist is implemented in C# [10]. However, Python provides a better environment for machine learning. First, Python is used for training the PID neural network. Then after the neural network is trained, the weights are copied into an identical neural network in the C# control code.

6.1 Simulation Implementation

The method used to simulate the servo from the servo's mathematical model is SciPy's odeint function [11]. This function receives an initial state, an array of timesteps, and an ordinary differential equation (ODE). The ODE is found by computing the inverse Laplace transform of the servo's transfer function. Equation 15 is the ODE for the servo.

$$\ddot{y}(t) = \frac{-b_m N^2 \eta R - \frac{K_t N^2 \eta}{K_\omega + N\eta K_t K_d}}{R(J_l + J_m N^2 \eta)} \dot{y}(t) + \frac{-N\eta K_t K_p}{R(J_l + J_m N^2 \eta)} y(t) + \frac{N\eta K_t K_p}{R(J_l + J_m N^2 \eta)} u(t) \tag{15}$$

To better simulate the actual servo, an angle sampling delay, a_{angle}, and a control signal delay, $a_{control}$, are implemented into the simulation. This is because the angle reading from the actual IMU, and the reaction from the servo to a control signal from the PID controller can be delayed. The delays are mathematically described in equation 16.

$$\begin{aligned} \phi_{delayed}(t) &= \phi(t - a_{angle}) \\ u_{delayed}(t) &= u(t - a_{control}) \end{aligned} \tag{16}$$

To tune the simulation to match the actual servo, a grid search over a parameter space for the time step length, angle sampling delay, saturation, and control delay, is performed. For each parameter combination, the response of the simulation is compared to the actual servo's response to a step response. The parameter combination with the lowest root-mean-squared error when compared to the actual response is chosen as the best parameter combination. Figure 12 compares the servo response to a step input for the actual servo and the simulation with the best parameters. Table 2 shows the best parameters from the grid search.

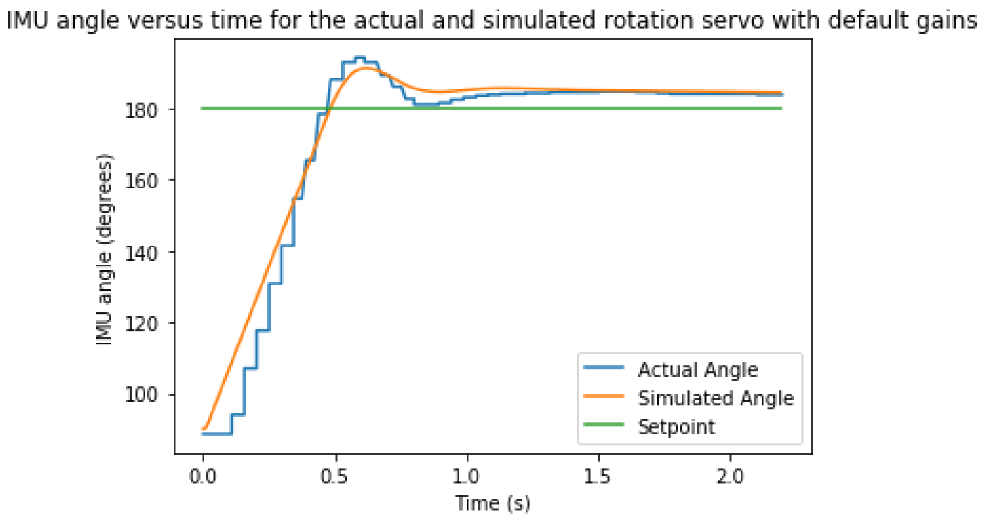

Figure 12: Tuned simulation vs actual for a step input.

Table 2: Best Parameters for the Simulation

Time step length	dt	0.04 s
Angle sampling delay	a_{angle}	3 time steps
Saturation point	s	8.789°
Control sampling delay	$a_{control}$	0 time steps

Algorithm 1 shows how Figure 8 is implemented in code to run a simulation. Here, T is the number of timesteps, each with a length dt, t_i is the time at timestep i, $r(t)$ is the setpoint in degrees, $d(t)$ is the disturbance in degrees, $y(t)$ is the angle of the servo in radians, $\phi(t)$ is the IMU angle in degrees, $e(t)$ is the error between the setpoint and the IMU angle, $u(t)$ is the control signal in degrees, y_0 and $\dot{y}_0$ are the initial conditions of the servo. $\phi(t)$ is defined such that when $y(t) = 0\,\text{rad}$ and $d(t) = 0°$, the servo angle is $\phi(t) = 180°$. This is consistent with how the servo angle is defined in figure 3.

Algorithm 1 Servo Simulation

```
1: Input: $T$, $r(t)$, $d(t)$, $y_0$, $\dot{y}_0$.
2: Output: $\phi(t)$.
3: for $i = 0$ to $T$ do
4:     $e(t_i) = r(t_i) - \phi(t_{i-1-a_{control}})$        ▷ Get error with delayed angle sample.
5:     $\dot{y}(t_{i-1}) = \frac{y(t_{i-1})-y(t_{i-2})}{t_{i-1}-t_{i-2}}$        ▷ Calculate velocity with first backwards difference.
6:     $K_p, K_i, K_d =$ Get_Gains($e(t_i), \phi(t_{i-1}), \dot{y}(t_{i-1})$) ▷ Predict the PID gains with the PID neural network.
7:     PID.Set_Gains($K_p, K_i, K_d$)
8:     $u(t_i) =$PID.Get_Control_Signal($e(t_i)$)        ▷ Get control signal.
9:     $u_{delay} = u(t_{i-a_{angle}})$        ▷ Get delayed control signal.
10:    $u_{delay\_sat} =$min(max($u_{delay}, -s$), $s$)        ▷ Saturate the control signal.
11:    $\dot{y}_0, y_0 =$SciPy.odeint($\dot{y}_0, y_0, [t_{i-1}, t_i], u_{delay\_sat}$)        ▷ Run odeint for one timestep.
12:    $\phi(t_i) = y_0\frac{180}{\pi} + 180 + d(t_i)$        ▷ Calculate $\phi$ by adding disturbance and 180°.
13: end for
14: return $\phi$        ▷ Return time response of the servo.
```

6.2 PID Neural Network Training

The PID neural network is trained using the Levenberg-Marquardt algorithm. Algorithm 2 describes this process [12]. Here, the values are the same as in algorithm 1. For training, $d(t)$ is an APRBS signal with length 25.0s, peak-to-peak amplitude of 45° and maximum hold time of 5.0s. x starts at 200 samples, and increases by 200 for each trial. 10 trials are run in total. The best weights are chosen from the 10 trials, according to which weights yield the lowest root mean squared error (RMSE) between the servo angle and the setpoint. Because the difference in load inertias for the rotation servo and flexion servo is small, it is assumed that the neural network will work similarly for both servos, so the same weights can be used for both servos.

Algorithm 2 Servo Simulation

```
1: Input: $T$, $r(t)$, $d(t)$, $y_0$, $\dot{y}_0$.
2: Output: Trained PID Neural Network Weights $\mathbf{w}$.
3: **while** Trials left **do**
4:     Perform truncated random initialization for neural network weights, $\mathbf{w}$.
5:     $T_{tmp} = T(t_0...t_x), r_{tmp} = r(t_0...t_x), d_{tmp} = d(t_0...t_x)$    ▷ Sample first $x$ values of $T$, $r$, and $d$.
6:     **while** Not converged **do**
7:         $J_w$ =Compute_Jacobian($T$, $r(t)$, $d(t)$, $y_0$, $\dot{y}_0$)    ▷ Numerically Compute jacobian using equation 7 and simulation from algorithm 1.
8:         $\phi_{sim}$ = Simulate($T$, $r(t)$, $d(t)$, $y_0$, $\dot{y}_0$)    ▷ Simulate servo response.
9:         $\sigma = [J_w^T J_w + \lambda I]^- 1 J_w^T[\phi_{sim} - r]$    ▷ Solve Levenberg-Marquardt equation for $\sigma$.
10:        **if** RMSE for simulation with $\mathbf{w} + \sigma <$ RMSE for simulation with $\mathbf{w}$ **then**
11:            $\mathbf{w} \leftarrow \mathbf{w} + \sigma$
12:            $\lambda = \frac{\lambda}{2}$
13:        **else**
14:            $\lambda = 2\lambda$
15:        **end if**
16:    **end while**
17:    **if** RMSE for simulation with $\mathbf{w} <$ RMSE for simulation with $\mathbf{w_{best}}$ **then**
18:        $\mathbf{w_{best}} \leftarrow \mathbf{w}$
19:        Increase number of samples, $x$.
20:    **end if**
21: **end while**
22: **return** $\mathbf{w_{best}}$
```

6.3 Neural Network Structure

The structure of the neural network is shown in figure 13. The neural network structure consists of 5 input nodes, 4 hidden nodes, and 3 output nodes. The input nodes consist of normalized error, normalized angle, normalized velocity, and activation values from hidden nodes 2 and 3. Normalization is performed using equation 17 which scales the inputs to a value between 0 and 1. Normalization is important, as it ensures the scale of each input value is equal relative to each other, so no single input can ”overshadow” the other inputs []. The activation values from hidden nodes 2 and 3 are used to give the network a one time-step recurrence, which can aid in applications of temporal nature such as a control system [6].

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{17}$$

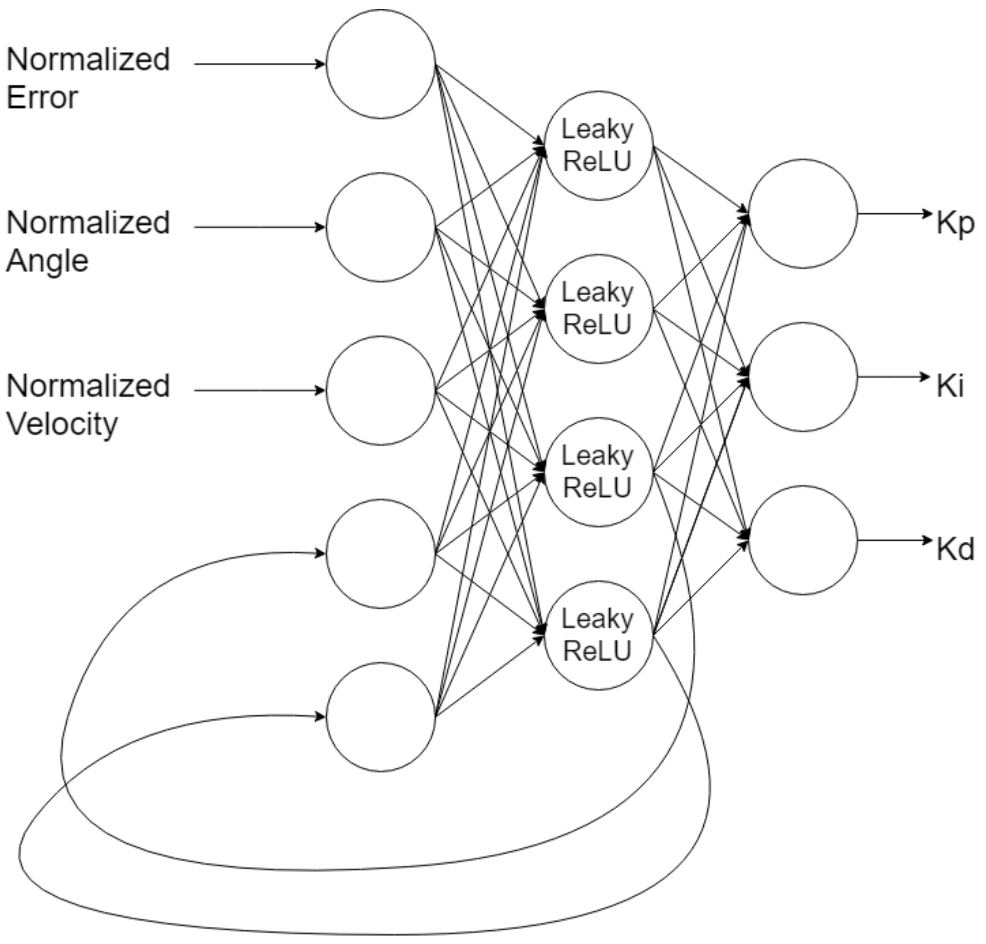

Figure 13: Structure of the PID neural network.

The activation function used at the hidden layer is a leaky ReLU with a slope of 0.3. Equation 18 is the equation describing a leaky ReLU.

$$a = \begin{cases} 0.3z & z \leq 0 \\ z & z > 0 \end{cases}$$

7 Results

The PID neural network training was performed for the rotation servo for 10 trials using the APRBS shown in figure 14. The best trial was trial three, which yielded an RMSE of 22.034 on the training APRBS.

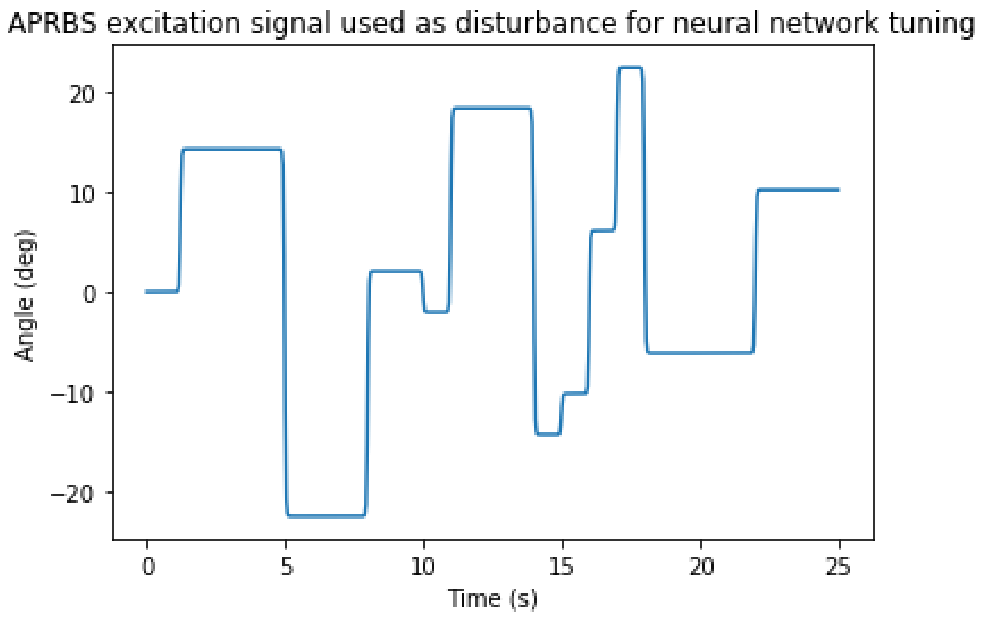

Figure 14: APRBS used for training.

After training, the neural network weights were copied into the PID neural network for both the rotation servo and the flexion servo in the C# program. A test was run to compare the step response of the automatic levelling wrist with and without PID Auto-Tuning. Figure 15 show the results of this test for the rotation servo. Figure 16 shows the value of the gains at each step in the test. Table 3 is the RMSE values for the rotation and flexion servo step response with and without PID auto-tuning. Analyzing figure 15 and figure 17 shows that PID auto-tuning can reduce the response time of the system, while also reducing overshoot and oscillations. RMSE was decreased by approximately 30% for both the rotation and flexion servos when PID auto-tuning was used.

Table 3: RMSE Values for Step Responses

	w/o PID Auto-Tuning	w/ PID Auto-Tuning
Rotation	1383.64	995.86
Flexion	1553.36	1027.76

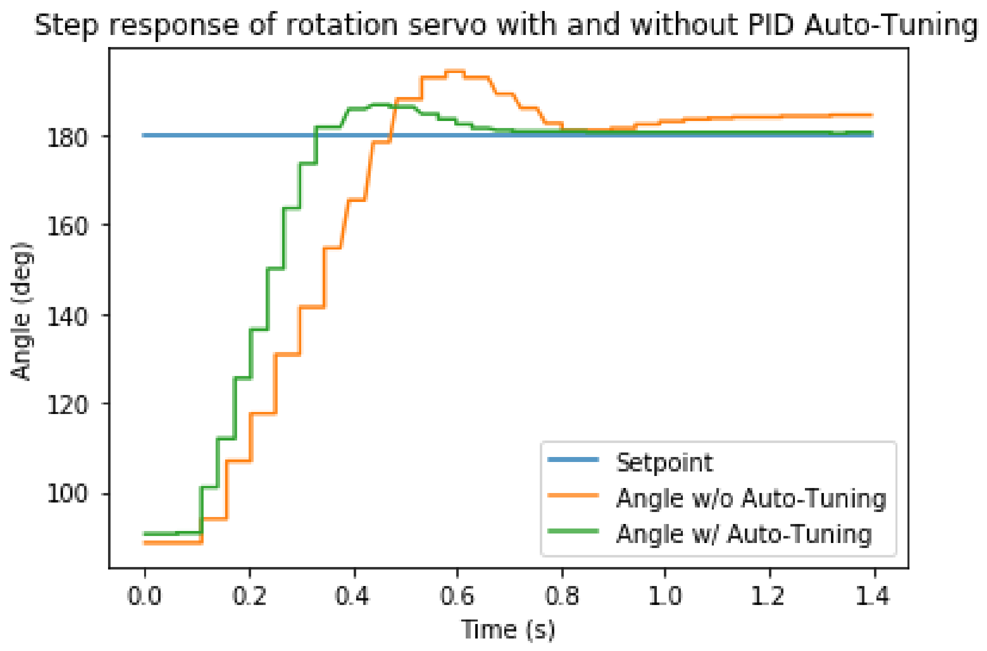

Figure 15: Rotation servo step response with and without PID Auto-Tuning.

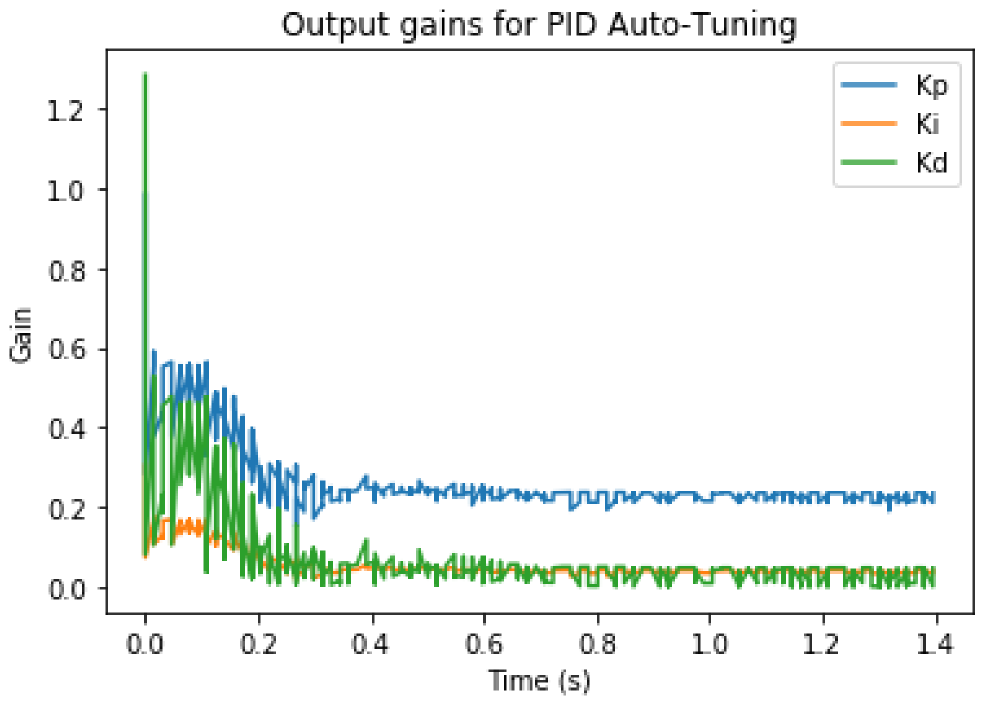

Figure 16: PID gains output by the rotation PID neural network during the step response test.

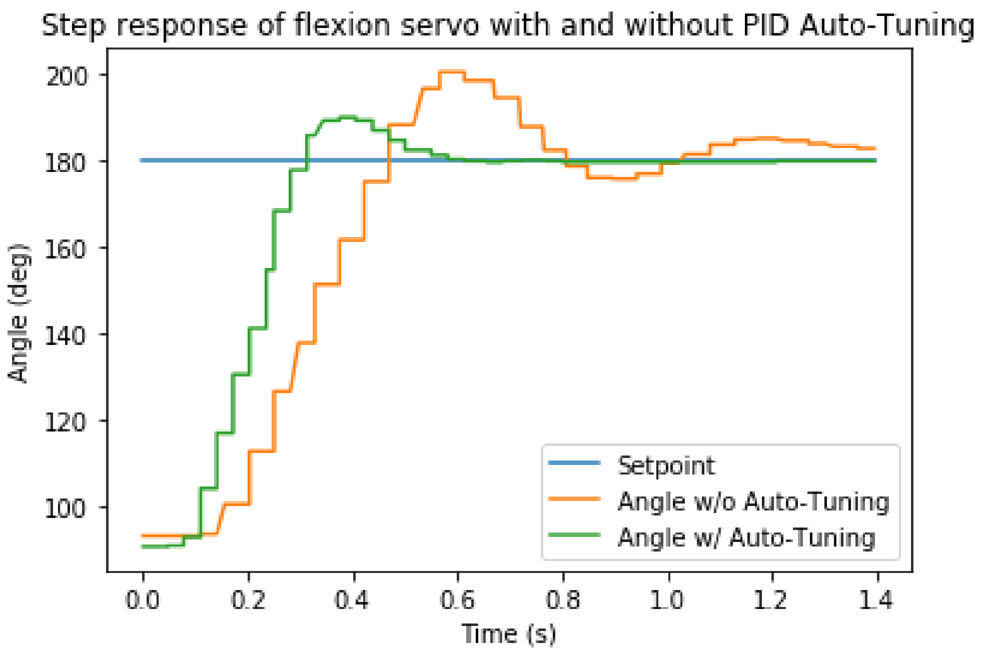

Figure 17: Flexion servo step response with and without PID Auto-Tuning.

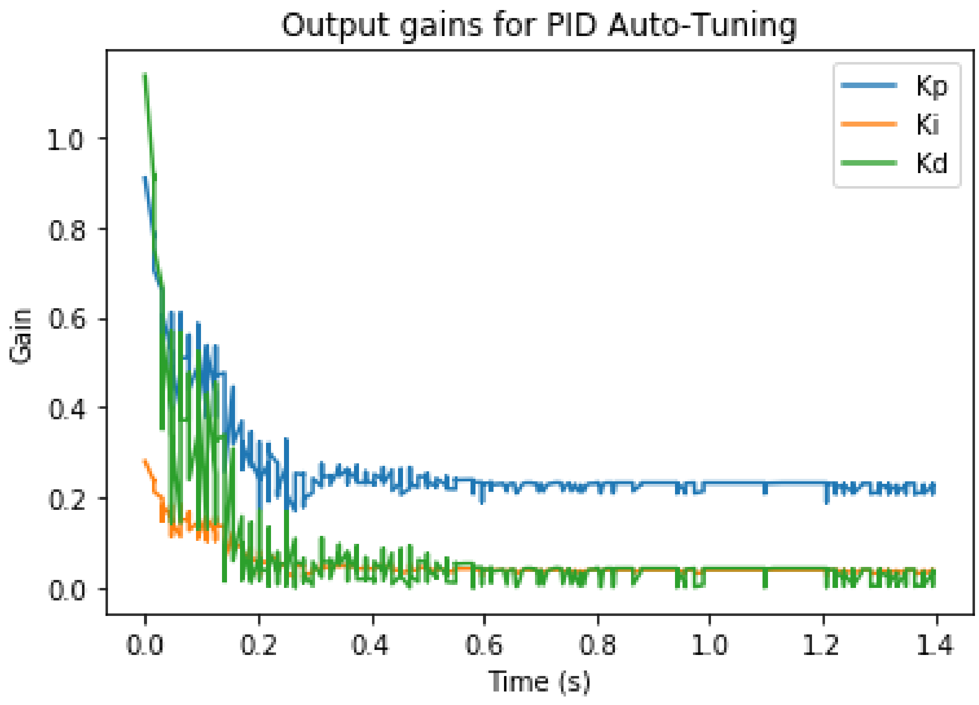

Figure 18: PID gains output by the flexion PID neural network during the step response test.

8 Conclusion

From the results it can be seen that PID auto-tuning can significantly improve the performance of the automatic levelling wrist. PID auto-tuning shows an RMSE decrease of approximately 30% for both servos, as well as reduced response time, overshoot and oscillations. Figure 16 and figure 18 show how the neural network acts in reaction to a disturbance. First the neural network increases the gains of K_p and K_i, to increase response time and tracking. Then as the servo moves closer to the setpoint, the PID gains are decreased to more stable values so that overshoot and oscillations are reduced.

From the results it can also be seen that the consequences of sharing weights between the rotation and flexion servos do not greatly affect the performance of the PID auto-tuning network, as the performance of the rotation and flexion servos are on par with each other.

In conclusion, PID auto-tuning for the automatic levelling wrist provides a tangible demo to show how machine learning can significantly improve the performance of a hardware system.

9 Recommendations

Although the PID auto-tuner can improve the performance of the automatic levelling wrist there is room for improvement.

The training process can be improved to include more trials as well as variations in the neural network structure. More trials would increase the chance that a good initialization is found, as initialization is random. Trying different neural network structures such as changing number of nodes, changing activation functions, and changing which node is recurrent could lead to an increase in performance, as the structure proposed in this project is not known to be optimal.

The position of the flexion servo can affect the response of the rotation servo because the load moment of inertia changes depending on the position of the flexion servo. A similar problem arises when a load is applied to the gripper, as the added load will increase the load moment of inertia for the rotation and flexion servos. These changing moments of inertia are not considered in the simulation and as a results the PID neural network is not trained to handle these situations. To address this problem the neural network should include the flexion servo position and sensed load in its inputs (the MX-28AT servos have load sensors). To supplement this, training should include situations with varying loads so that it can learn to adapt to these differing situations.

With these improvements and with further tuning of the simulation parameters, PID auto-tuning can greatly improve the performance of the automatic levelling wrist so that it is more reliable, and more intuitive for users to use.

References

[1] N. M. Bajaj, A. J. Spiers, and A. M. Dollar, “State of the art in prosthetic wrists: Commercial and research devices”, in *2015 IEEE International Conference on Rehabilitation Robotics (ICORR)*, Aug. 2015, pp. 331–338.

[2] S. L. Carey, M. J. Highsmith, M. E. Maitland, and R. V. Dubey, “Compensatory movements of transradial prosthesis users during common tasks”, *Clinical Biomechanics*, vol. 23, no. 9, pp. 1128 –1135, 2008.

[3] K. Østlie, R. J. Franklin, O. H. Skjeldal, A. Skrondal, and P. Magnus, “Musculoskeletal pain and overuse syndromes in adult acquired major upper-limb amputees”, *Archives of Physical Medicine and Rehabilitation*, vol. 92, no. 12, pp. 1967 –1973, 2011.

[4] D. J. A. Brenneis, ”Automatic Levelling of a Prosthetic Wrist”, *University of Alberta*, 2019.

[5] O. Nelles, ”Nonlinear system identification, Classical approaches to neural networks and fuzzy models”, *Berlin, Heidelberg: Springer Verlag*, 1 ed., 2010

[6] J. Gunther, ”Machine intelligence for adaptable closed loop and open loop production engineering systems”, *University of Alberta*, 2019.

[7] R. S. Sutton and A. G. Barto, ”Introduction to reinforcement learning”, Dissertation, *Technische Universität München*, Munich, Germany, 2018.

[8] M. R. O. A. Maximo, C. H. C. Ribeiro and R. J. M. Afonso, ”Modeling of a position servo used in robotics applications”, *Autonomous Computational Systems Lab (LAB-SCA) and Electronic Engineering Division, Computer Science Division, Aeronautics Institute of Technology, Pra¸ca Marechal Eduardo Gomes*, 2017.

[9] ”Robotis e-manual”, *Robotis*, 2019. [Online]. Available: http://emanual.robotis.com/docs/en/dxl/mx/mx-28/. [Accessed: 16-Jul-2019].

[10]

[11] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

[12]

Appendices

A Appendix