Shape Estimation Of A Concentric Tube Robot Using A Machine Learning Model For Use In Bronchoscopy

Ruairi O'Hare

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2021

Abstract

A concentric tube robot is a small robot made up of 3 pre-curved extendable metal tubes, where the segments of the robot can extend and rotate. The robot can be used as a bronchoscope for use in bronchoscopy surgery, allowing access to very small areas in the lungs due to its small size. During surgery, it's important to have an idea of the current shape of the robot, as it changes during control as it extends and rotates. When the robot is inside of the body, a surgeon can not see the shape of the robot, so the shape needs to be estimated accurately and quickly, due to the time critical nature of some surgeries.

An existing physics based model can calculate the shape using a set of differential equations with mixed boundary values. The shape is represented by a series of x, y, z coordinate points that can be plotted in a 3D space. However, this model is difficult to solve and computationally inefficient. Additionally, this model is not totally accurate as external factors are not taken into account, such as uncertain curvature, torsion and friction. Instead, machine learning techniques could be used to train a model that could take the robot's inputs (the extensions and rotations of each segment of the robot) and predict the robot's shape accurately and efficiently, making improvements over the existing physics based-model.

This report investigates the viability of using a neural network model to estimate the shape of the concentric tube robot using simulated data from the existing physics model. Curve parameterization was used to reduce shapes down from a large number of coordinate points to just 15 polynomial coefficients. A neural network model was built that could predict these coefficients given the robot's inputs by training on hundreds of sample inputs parameters and output shapes. The model was evaluated against another neural network model using non-parameterized data and against the original physics-based model. The neural network model using curve parameterization was found to be highly efficient and able to represent the majority of robot shapes with a good degree of accuracy.

Acknowledgements

Thanks to my supervisor Mohsen Khadem for the amazing support and guidance throughout this project, it has been greatly appreciated.

Table of Contents

Chapter 1

Problem Outline And Background Research

1.1 Chapter Description

This chapter first explains the concentric tube robot and existing physics-based model in order to fully describe the problem this report will attempt to solve. Previous uses of machine learning techniques for shape estimation of the concentric tube robot are also discussed, as well as the existing codebase and data the project will be based on.

1.2 The Concentric Tube Robot

Concentric tube robots are very similar in design to medical needles and catheters (see Figure 1.1), meaning they are minimally invasive and are ideal for use in microsurgery due to their flexibility and ability to reach into small sections of the body, making them ideal for use as a bronchoscope [14]. In lung operation, the pre-curved design of the concentric tube robot can reduce force on the patients head and neck by 95% due to limited device angulation [2].

The physical robot that will be the basis of the models discussed in this report is a concentric tube robot. It is made up of 3 lightweight, flexible, nested tubes, A, B and C, of lengths 28.58cm, 20.25cm, 9.45cm respectively, which are pre-curved. The tubes can rotate from 0 to 360 degrees and can extend from their starting position based on their length. Each tube's thinness, lightweight and ability to rotate and curve allow for access to hard to reach places during minimally invasive surgery. This makes the concentric tube robot model ideal for use as a bronchoscope that can be used to reach inside the small airways (bronchi) of the body and lungs.

A surgeon could be specially trained to control the concentric tube robot as a bronchoscope, but during surgery it is important for the surgeon to know the shape of the robot while it is inside of the body in order to decide how to control the robot and to avoid direct contact with sensitive areas of the bronchi. This can be done by calculating the current shape of the robot as a set of 3D coordinates using a mathematical model. Us-

ing the current lengths and rotations as the input data, a 3D plot of the robot's shape can be created using the output coordinates. The 3D plot of the shape of the robot provides the surgeon with a visual guide of the robot's current shape and position.

An issue with this method is that large matrix-based mathematical techniques needed to calculate the shape, such as calculating the Jacobian matrix or compliance matrix, are computationally intensive and therefore results are not always immediate. Additionally, due to the flexible, lightweight materials typically used for the tubes, they can often bend slightly more or less than the mathematical model can anticipate, leading to some error.

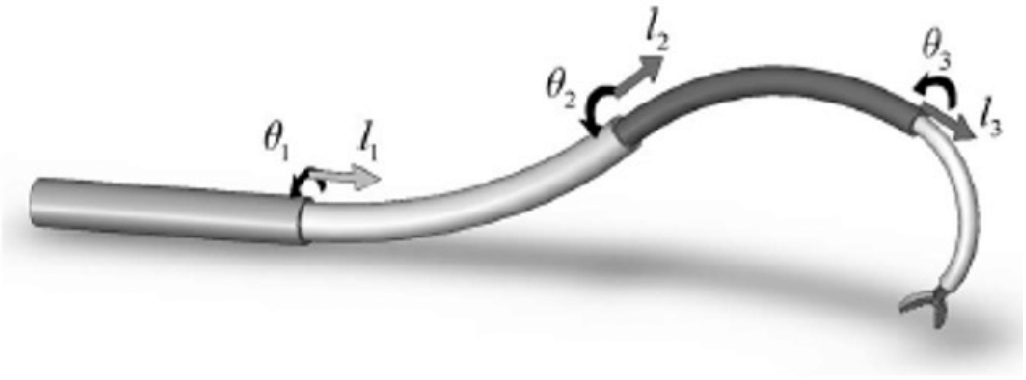

Figure 1.1: Diagram of the concentric tube robot [5]

1.3 The Existing Physics-Based Model

The shape of the model is calculated using kinematics equations. The following parameters of each tube are preset in the model:

- Length,
- Curved length,
- Inner diameter,
- Outer diameter,
- Stiffness,
- Torsional stiffness,
- X-curvature,
- Y-curvature

The joint variables are also defined, as well as the initial position, q0, and twist of the joints for the initial value problem solver. The force f on the robot is also defined. The 6 joint variables are our input parameters, q, used in the mathematical model to calculate the shape are:

q = [Length1, Length2, Length3, Degree1, Degree2, Degree3].

Length3 can range from 0 to 9.85cm. Length2 can range from 0 to 20.25cm minus the difference in Length3 from the maximum length of tube 3. Length1 can range from 0 to 28.58cm minus the difference in Length2 from the maximum length of tube 2. Degree1, Degree2 and Degree3 can each range from 0-2π radians to rotate.

The model uses a set of differential equations with mixed-boundary values, which is difficult to solve and computationally inefficient. The output of the model is a shape, represented by an array of 3D points that is then plotted to give a visual representation of the shape to the controller of the robot. q will serve as the input parameters to the machine learning model that will be discussed later in this report.

1.4 Problem Outline

This report evaluates how estimation of the shape of a concentric tube robot for use in minimally invasive bronchoscopy surgery can be improved using machine learning methods instead of live mathematical calculation using a physics-based model. This is for two reasons:

- The mathematical calculation is slow [9] and a data-driven approach could make the shape estimation faster. When the shape of the concentric tube robot is calculated using the current codebase [10] at a single time, it can take over 10 seconds on average to produce a shape. This would be problematic during live-use control of the robot as a surgeon needs estimates of the robot's shape rapidly, in order to decide where to move the robot next.
- The mathematical model is not always a totally accurate representation of the physical model due to external factors such as friction [12], non-homogeneous properties and imprecisely shaped tubes, uncertain curvature and torsion.

By building a machine learning model and implementing a data-driven approach to estimation of the shape, these problems could possibly be solved. Using a machine learning method could remove the need for complex mathematical methods using large computationally complex matrices, reducing time needed to estimate the shape of the model, instead producing a shape based on a confident prediction, calculated by training based on a large number of shapes.

A machine learning model could possibly produce more accurate results by being trained on data that takes into account the mean error of the robot's estimated shape by calculation, compared to the real shape (e.g. by scaling shape points by +0.5-0.5%), so shape estimations could be made with the mean error accounted for in the model. There are many possible machine learning models that could be used to estimate the shape, such as support vector machines, neural networks and regression techniques. Neural networks [11] have been explored before with positive results using this type of concentric tube robot model, but it could be possible to further improve the machine learning algorithms used to increase efficiency and increase accuracy of the result.

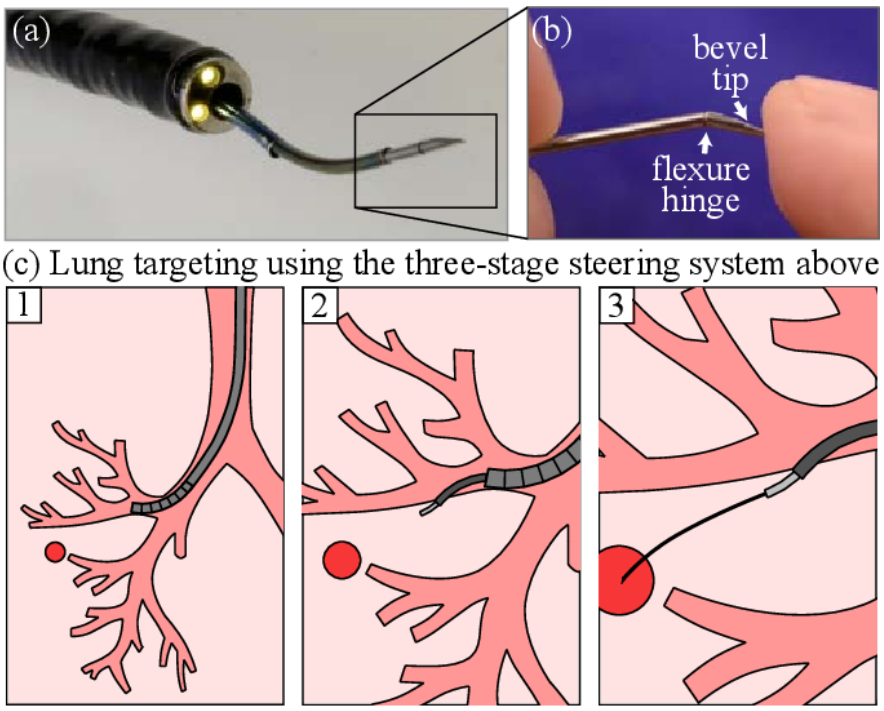

Figure 1.2: Diagram demonstrating how CTR is used during bronchus surgery [18]

1.5 Learning Robot Shape Using Machine Learning

The concentric tube robot shape calculated using kinematics has previously been modelled using deep, feed-forward neural networks using a mixture of simulated and physical data [11]. The physics-based model that the learned model was compared to had uncertainty in curvature and torsion, which the learned model would attempt to account for. To generate training data, images were taken of the real model, whilst a colouring threshold was used to segment the shape. Silhouette algorithms were used to produces voxels that could be used to generate a set of 3D points for the shape. Due to restricted lab access caused by the ongoing pandemic, the model described later in the report will be built solely using simulated data based off the existing physics-based model.

Training on 100,000 data points, plotted results of the shape produced by the learnt model could be compared with images of the real robot as a measurement of accuracy. Results were evaluated using maximum deviation, mean squared error and sum of the deviation from the robot's shaft. Timing of computation using the learnt model was also measured against the physics-based model and the learnt model was found to be faster due to neural networks allowing computation to be batched, allowing many shape computations to happen in parallel. Results showed that the learnt model showed improvement against the physics-based model, allowing for safer motion planning and control of the robot, and also proved that neural networks are a viable method for improving shape estimation of the robot.

Control of concentric tube robots has also been explored using deep reinforcement learning using inverse kinematics [9]. Simulated data is used to train a multi-level

perceptron network for inverse kinematics (using Markov decision process) on the tube robot. Zero-mean Gaussian noise, Ornstein–Uhlenbeck noise and parameter noise were then added to the learned policy in order to improve the data-driven approach to control.

Markov Random Fields have been used to overcome inaccuracies and shape deformation of the robot due to changes in load and collision [19], which was also found to be a more robust solution for shape estimation than solely using kinematics.

Machine learning methods that could further improve the speed at which the shape of the concentric tube robot is estimated could be investigated, in order to make control of the robot easier and more reliable for surgeons, especially when using older computers which may not be able to computationally keep up with existing physics-based models for shape estimation. Whilst there are many machine learning techniques and models that could be viable to accurately estimate the large number of 3D points needed to make the shape of the robot, this report will focus on using a neural network due to previous positive results for this type of problem.

1.6 The Current Codebase

In this project, data will be generated for the concentric tube robot using an existing codebase provided by the project's supervisor [10]. There are 5 python files, utilising common python libraries such as Numpy, Scipy, and Matplotlib. The parameters of the physical tubes that makes up the robot are initialised. These are the same preset parameters described before such as curve length, stiffness and torsion, not to be confused with the joint variables, q.

The robot is then broken down into multiple segments between transition points. Segments can contain multiple tubes. The model of the concentric tube robot is simulated given an initial value problem. The tube parameters, the initial position of the joints and the joint variables are given to the model as input. Differential equations using different mixed boundary values are solved to produce an estimation of the robot's shape as output.

After computation, the model's shape can be accessed as a set of 3D coordinate points that can then be plotted and visualised (see Figure 1.3). To generate the data using this model, the joint variables will be adjusted hundreds of times to create hundreds of shapes. Each tube length will be adjusted based on a random value ranging from 0 to the maximum displacement possible based on the other tube's displacement values. Each tube will be randomly rotated between 0 and 2π radians. The dataset will consist of approximately 500 instances of the joint variables and resulting shape. Each instance in the dataset will consist of the joint variables as the initial 6 features, followed by approximately 150 points, in the format [x,y,z,x,y,z...] representing the shape that resulted from the input joint variables.

The machine learning model chosen will be expected to take valid joint variables as input, and produce an accurate shape consisting of approximately 150 points as the output. A validation set will be created using a subset of the randomised data and will

be unseen by the model during training and used for evaluation only. After evaluating error of the machine learning model using a validation set [15], the model could be further tuned to increase accuracy and over-fitting could be mitigated by continuously training and testing the model using newly generated, randomised data. Once an accurate model is trained, efficiency of the model will be evaluated to help decide the viability of the model during live-control procedures such as surgery.

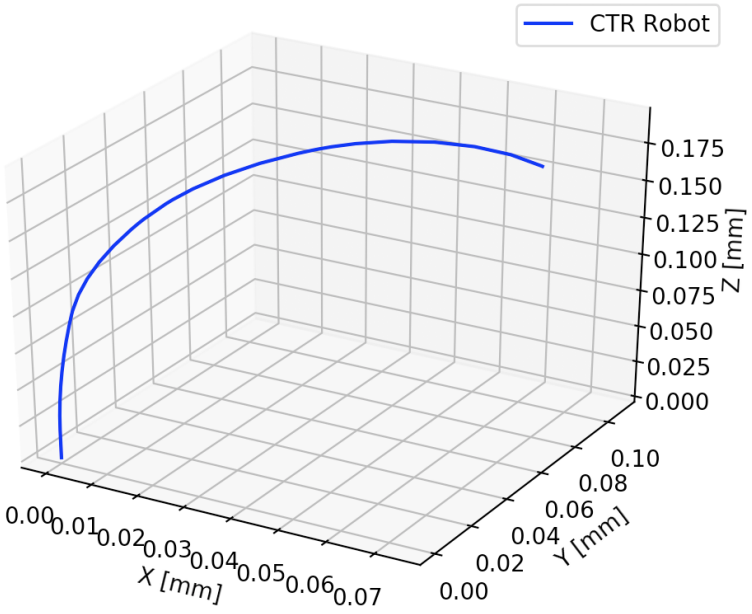

Figure 1.3: Plot of the robot produced by the existing codebase, given input q= [0.123 mm, 0.0796mm, 0.0724mm, 1.973 radians, 2.648 radians, 2.802 radians]

Chapter 2

Methodology

2.1 Chapter Description

This chapter describes how the data needed to train a neural network model was generated and standardised. An initial neural network model using non-parameterized data is described. Problems of training and predicting large shapes of 90 points were found, leading to an alternate method to be required to represent the data. Curve parameterization is then discussed, reducing each shape from 90 points down to 15 coefficients, solving problems found when building a model using non-parameterized data. Finally, a linear regression method is discussed that was used to predict arc-length of the shape for use with curve parameterization.

2.2 Data Generation

In order to create a machine learning model to estimate the shape of the CTR (concentric tube robot), a large amount of data needed to be generated and standardised. This data could then be used as a baseline for comparison between the existing mathematical model discussed and the machine learning model. 500 shapes worth of data was decided on as a starting point, which could then be increased or decreased based on future results.

The 500 shapes were generated based off random configurations of the 6 input parameters. The minimum extension was set to 0cm and the maximum extension is the total robot length, 28.58cm. The 3 segment lengths of the CTR, A,B,C, were 8.33cm,10.8cm and 9.45cm respectively. The C segment extends first. The length B can extend is dependent on C's displacement, whilst A is dependent on the displacement of both B and C.

- C could move between 0 and 9.45cm.
- B can move between 0cm and 20.25cm-(9.45cm-C's displacement).
- A can move between 0cm and 28.58cm-(20.25cm-B's displacement).

The angles for the 3 segments were also randomised, each independently set between [0,2π] radians. This allows for a large variety of shapes to be created, and gives a higher degree of fairness to the model as it has to deal with some unconventional shapes due to the random factor, as opposed to training the model only on more conventional shapes. An alternate approach considered was to sequentially increment each part of the robot with every possible angle and length, with the hope to procedurally cover every shape possible. It was decided that this 'brute-force' approach was unjustifiable, firstly due to the massive amount of data generated could make training the model a much more difficult task and the need to create a method that would cover all shapes. Additionally, every shape would only appear once in the training set so we would lose the advantage of the model being very accurate at predicting the most common shapes the robot takes.

Generating the data consisted of defining the bounds of the random input parameters and running the existing codebase to record the input parameters and coordinates of the resulting shape as rows in a CSV (Comma Separated Value) file, allowing for easy access to the data in Excel. CSV is also commonly supported in many python libraries. The first 6 columns in this data set represent A, B, C length extensions, followed by the A,B,C angle rotation degrees. The following number of columns per each row varied widely from approximately 90 points (30 coordinates) to 1467 points (489 coordinates).

Running the existing codebase just 500 times proved very computationally intensive. The mean time to produce one shape was normally over 10 seconds, meaning producing the dataset would take approximately 83 minutes. This is primarily due to the need to calculate the compliance matrix followed by the Jacobian matrix. Additionally, the huge amount of coordinates for some shapes caused issues such as memory overflow on a home laptop, so an alternate more capable machine was needed to retrieve all 500 samples.

The large amount of time to generate a single sample recorded, in addition to the large memory requirements of estimating brand new shapes frequently, helped further justify the need for a faster, less computationally intensive method that a trained machine learning model could potentially provide.

2.3 Data Standardisation and Downsampling

The next step in processing the data was to standardise the output data. This is because large variation in size of output shapes from 90 points to 1467 points makes comparison between shapes more challenging. Additionally, some machine learning models such as sequential neural networks (discussed further in Section 3.1) require each input to be the same size for the model to work correctly. Upsampling the smaller samples to 150 points and downsampling larger samples to 150 points was considered.

An issue with upsampling is that we create new data points not present in the original set. This could cause problems when evaluating predicted data using the model and cause issues training. The trade-off of downsampling is loss of crucial information if downsampling is not performed uniformly and consistently across all shapes.

The average number of points was found to be 262.92, which felt like an excessive target as the shape of a curve can still be observed with a lower resolution of coordinates. The decision was made to downsample all samples uniformly to the minimum number of coordinates as 90 points was found to be enough to capture the shapes when a graph of the shape was produced and observed. The most important aspect of the data that had to be kept was the start and end point on the robot, as the remainder of the shape could still be represented with various choices of coordinates between the start and end of the shape.

Alternate methods exist to downsample data, such as the Ramer–Douglas–Peckham algorithm [17]. However this method requires an optimum epsilon for the algorithm to be found. Due to this drawback, a simpler, uniform method was used to downsample the data.

Each row in the data set was reshaped by removing the input parameters and reshaping the coordinates into a single column of triplets to evaluate each x, y, z set as a single coordinate. A list of indexes was calculated uniformly by multiplying the number of coordinates in each shape by an iterator from 0 to 30, then dividing this number by 30.

This method was used to the find the indexes of the coordinates that could be extracted as the 30 evenly spaced coordinates for the line, always including the initial and final coordinate. This meant that every single shape our model would train on was of equal length and each created in the same uniform procedure. This method also ensured points were taken in a fair distribution. Each shape having the same number of coordinates also allows for fair comparison between shapes when building and evaluating the model.

At this stage 2 data sets have been produced:

- Xtrn, size 500x6, consisting of 500 samples of [Length1, Length2, Length3, Degree1, Degree2, Degree3], the input parameters for each shape. This will be the input training data for our model.
- Ytrn, size 500x90, consisting of 500 samples of [x1, y1, z1, x2, y2, z2...x90, y90, z90], the output coordinates representing the shape. This will be the output training data for our model.

2.4 Investigating Neural Networks

After standardising the 500 shape dataset, the first 'brute-force' neural network was built to investigate what results were possible from training on large sized samples with many numbers. The large output 90 point samples could prove difficult to train on without fine tuning. This is also due to the difficulty of correlating relationships between x's, y's and z's in coordinates, rather than training on a more linear numerical sequence. TensorFlow, an open-source machine learning platform [1] with support for deep training and neural networks, was chosen to build this initial model.

Keras [8] Sequential model, a Python API built on top of TensorFlow was used. 6 layers were used in the model, each consisting of 90 neurons per layer and 100,000 epochs

was selected arbitrarily to obtain a relatively high accuracy. After 100,000 epochs, this network was able to obtain 82.40% accuracy on the training data. However, this figure is based off mean squared error, meaning that the model is really only measuring the distance between the predicted and original output rather than finer intricacies of the shape. For example, we could have a very different shape, but if this shape's points lie close to the original shape, it would still produce high accuracy despite obvious visual differences. This means that this high number isn't quite meaningful without visualisation compared with the original shape.

I trained this original network using 250 training samples and 250 testing samples for evaluation. Selecting a handful of samples from the 250 predicted shapes and comparing with 250 testing samples, an arbitrary accuracy of 66% is achieved on the 250 test samples.

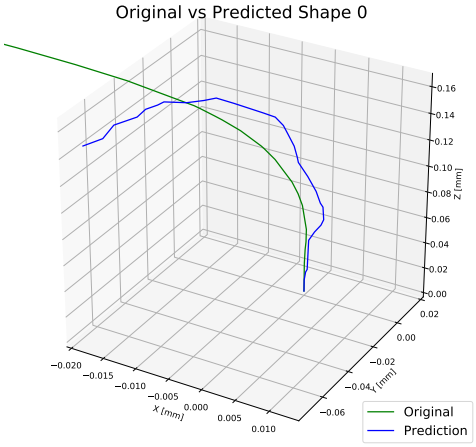

Figure 2.1: Shape 0

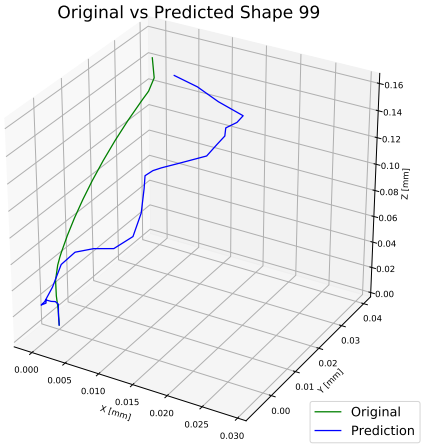

Figure 2.2: Shape 99

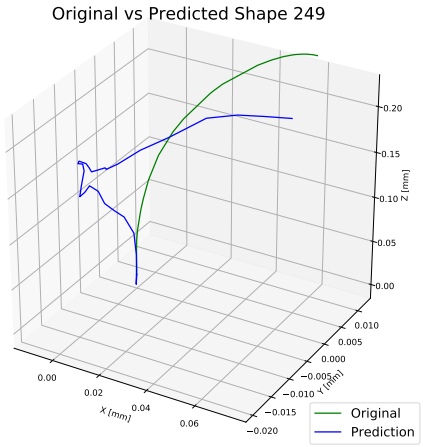

Figure 2.3: Shape 249

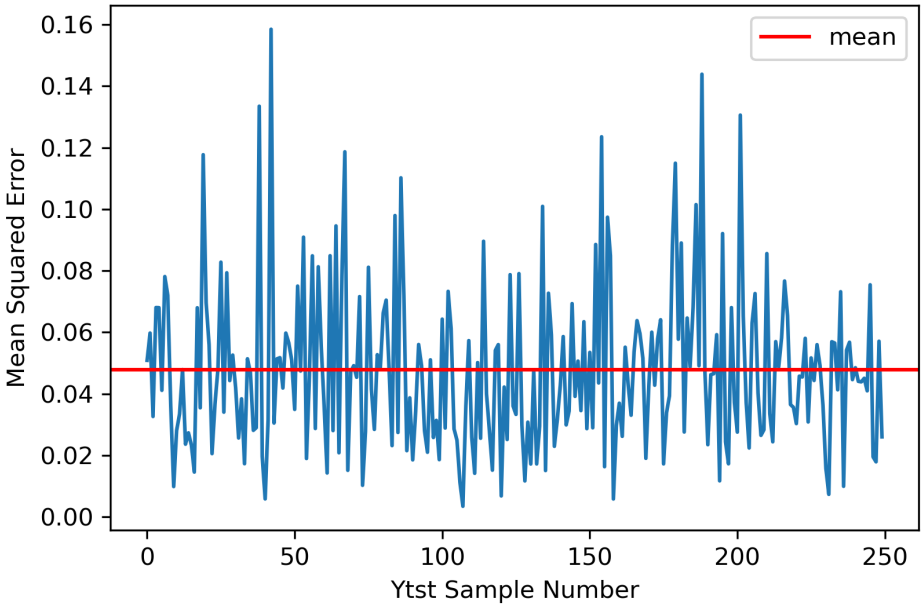

Figure 2.4: Mean Squared Error Across The 250 Sample Test Set

As seen in Figure 2.1-2.3, the predicted curves are very 'jittery', capturing the shape in a very broad manner, but lacking smoothness as each point of the curve jumps around to adjust and improve mean squared error. Figure 2.4 shows the mean squared error between the final point of the curve of the 250 predicted shapes measured against the final point of the curve of the original 250 test shapes. The root mean squared error (Figure 2.5) here was found to be just 0.0477mm, which is 20.1% of the mean total length of the shapes, 0.2369mm. This is a surprisingly low error given the brute force nature of this original network. However, this measurement, whilst useful for showing how the curve ends up in a relatively accurate position, doesn't capture how volatile the resulting shapes are.

$$\sqrt{\frac{1}{n}\sum_{n=1}^{n}(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

Figure 2.5: Root Mean Squared Error Formula

It was determined that these predicted shapes were inaccurate partly due to the large number of coordinates per shape making it challenging to determine relationships between the 90 continuous points. Before thinking about further tuning this model, it was decided that it would be important to try to find a method that could allow further reduction of the number of points in the dataset, whilst taking into consideration extra parameters such as the length of the curve.

2.5 Curve Parameterization

A neural network processes an input by multiplying each parameter of the input by the weights of each neuron using an activation function. The output is then passed through multiple hidden layers with different weights using an activation function (it is optional to change the activation function at each layer) and finally through a final layer of neurons to make a prediction. Neural networks can be used for regression problems when figures such as coordinate points need to be accurately predicted based on past data.

However in order to make accurate predictions, a neural network must be trained and the weights of the neurons in each layer must be determined. The quality of the network's training has a large impact on its success. An appropriate activation function that will measure error in an appropriate way for the input is needed to determine useful weights of the neurons. The number of neurons in each layer can be tuned, as well as the number of hidden layers. The input layer must have the same number of neurons as input parameters given and the output layer must have the same number of neurons as the number of parameters required in the output.

The process of tuning the neural network using cross-validation will be further discussed during the evaluation section of this report. However, a fundamental machine learning issue is that a very large input with many variables makes it more difficult to find the relationship between all the parameters of the input. This problem is commonly known as the curse of dimensionality [6]. This can cause predictions made by the network to be less accurate, and cause training and predictions to take longer to compute. Additionally, with very large inputs, to accurately determine the relationship between every parameter, a much larger number of sample inputs to train on would be required for the neural network to begin to become confident in predicting the relationships.

The relationship between parameters becomes even more difficult when the parameters are related in a non-linear, complicated way. For example, in the current data set, each set of 3 numbers (x,y,z) are clearly related, which needs to be identified. Additionally, each x will influence the next x, each y will influence the next y and so on. In the 90 point curve dataset, this means that the triplet relationship must be taken into account when predicting points as well as the relationship between every 4 points to account for the individual relationships between the 3 axis from one point to the next.

A much reduced input and output for the data would be very helpful for increasing accuracy. Additionally, a way to format the data to very clearly separate the x,y and z in the input and make the input have a more linear progression would make training the neural network a much simpler and reliable task. In order to achieve this curve parameterization [7] was considered and implemented.

3D curves can be represented using polynomial coefficients as the parameters using the arc-length of the curve as a linear progression. The 3D curve is represented using three polynomials, each representing x,y or z. To fit a polynomial to an axis, each point is mapped to the cumulative arc-length. The arc-length is recalculated cumulatively for each x value e.g. the first x-point maps to arc length 0. The next x-point is mapped to

the total arc length calculated from point 1 to point 2. The third x-point maps to the total arc length from point 1 to point 3 cumulatively and so on until the final thirtieth x-point is mapped to the the total arc length of the curve.

The arc-length is calculated between each point of the 30-point long curve. This is done by cumulatively adding the distance between coordinate points on the curve using euclidean distance (Figure 2.6). The NumPy function PolyFit is then used to map each x-value to the corresponding cumulative arc-length. A degree of 4 was chosen for the formula to produce 5 coefficients for the axis. The coefficients are fit by minimising the squared error between the x-values and the arc-length using a Vandermonde matrix in PolyFit.

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Figure 2.6: Euclidean Distance Formula

When this process is performed for the x, y and z-axis, the result is 3 arrays of 5 coefficients which altogether represent the curve. This is a significant reduction in number of parameters from 90 points per curve to 15 points per curve, an 83.33% reduction. Rather than train a neural network using triplet coordinates, the neural network model will be trained to predict these 15 coefficients. Each data sample will consist of 5 x-coefficients, 5 y-coefficients and 5 z-coefficients, giving each sample a far clearer progression to identify, reducing training and prediction times due to the decreased number of parameters and increasing accuracy due to less numbers to predict.

Reduced number of parameters should also cause images of predicted curves to be smoother, as a large number of coordinate predictions can cause the curve to make many small jumps between each point, resulting in a jittery curve. A parameterized curve should solve this by following the arc-length evenly for each point and work particularly well for a sequential neural network.

To recreate a curve from its polynomial coefficients, the length of the curve is predicted first (this will be discussed in the next section 2.6) and divided up equally into 30 points from 0 to the total length using the Numpy Linspace function. This allows us to evaluate the polynomial sequentially, using Horner's Method [4] to produce an x-value for 30 continuous points. When this is done for all 15 coefficients, this results in x, y and z coordinates produced from coefficents that were calculated using the original 90 coordinates and captures the path and shape of the original curve.

2.6 Linear Regression For Arc-Length

To begin recreating curves from the parameters, the original arc-length values were reused to check that the curves recreated from polynomials were similar to the original curves. This was found to be true but in order to accurately predict a curve from the 6 input parameters only, a way to predict the length of the curve needed to be found, as curve lengths greatly varied from 0.1476mm to 0.3793mm. The total arc-length

was calculated for every curve and made and a dataset was made mapping each input parameters to the total arc-length for the curve.

Linear regression was determined as an effective method, since only a single output, the total arc-length, needed to be predicted for each curve. This was very fast to train and resulted in a high accuracy of 91.74% due to the simple nature of the regression. Now to predict and plot a curve based on it's polynomial coefficients, the coefficients would be predicted with the neural network model and the total arc-length would be predicted by this linear regression method. The arc-length would then be divided into 30 evenly distributed points from 0 to max arc-length. The polynomial would then be evaluated using the cumulative sequential arc-length array to generate 30 x-points, 30 y-points and 30 z-points, allowing a 3D curved to be plotted based solely on the 6 input parameters.

Chapter 3

Evaluation

3.1 Chapter Description

In this chapter, the selected parameterized neural network model is fully evaluated. Implementation and design decisions are compared and justified for the model using experiments to determine different aspects of the network such as the optimal number of hidden layers, epochs and activation functions selected. The parameterized neural network model is compared to the original physics-based model, as well as the original non-parameterized neural network model through statistical and visual evaluation. Finally, efficiency of the parameterized neural network model is evaluated against the original physics-based model.

3.2 Sequential VS Functional API

Keras has two API's to build a neural network with, the Sequential API and the Functional API. The Sequential API is known to be simple and fast to work with, creating a neural network through stacking layers and handling a single input and output. The Functional API is more flexible, offering layers to connect to many more layers, allowing multiple input and output layers and generally allowing creation of more complex neural networks. The decision was made to use the Sequential API as it more closely met our model's requirements of needing a single input and single output at a time, and potentially be easier to set up and run experiments with.

3.3 Activation Function

The size of the input layer is the same size of the input, in our case 6 neurons, representing the 6 input parameters of the robot. The number of neurons in the output layer is the same size as the output, which in this case is 15 neurons, representing the number of coefficients our model needs to predict. The next aspect to be selected is the activation function. Keras contains various activation functions to choose from. ReLU (Rectified Linear Unit), Sigmoid and TanH are common activation functions

$$\sigma(x) = \begin{cases} max(0, x) & , x >= 0 \\ 0 & , x < 0 \end{cases}$$

Figure 3.1: Relu Formula [3]

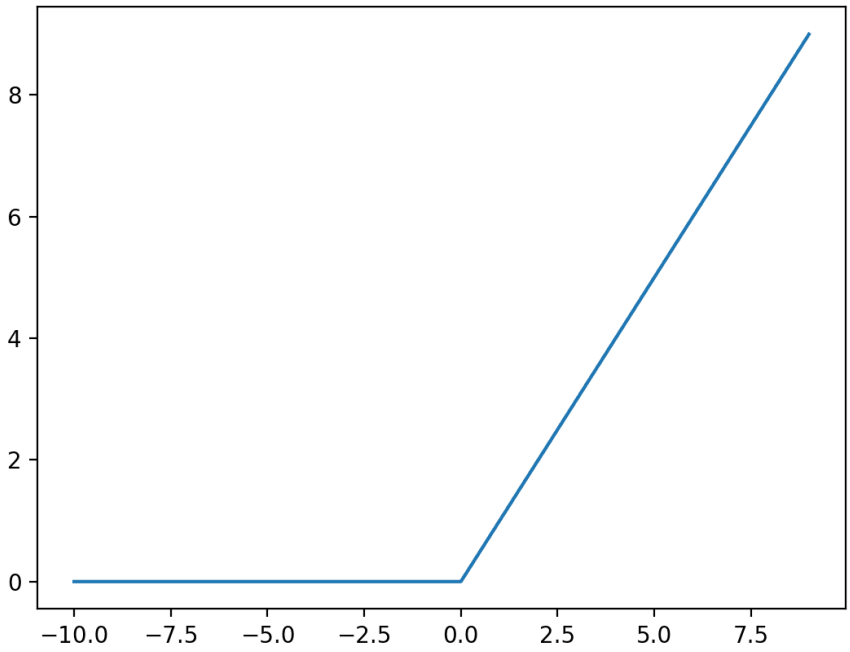

Figure 3.2: Plot Of ReLU Function

in neural networks. The ReLU function is fast to compute due to its simple formula, which takes an input x and returns x if greater than 0 and otherwise returns 0 (Figure 3.1-3.2). In contrast, TanH and Sigmoid are more computationally demanding due to their exponential functions.

The drawback of using ReLU is the issue of Dying ReLU [13], where some neurons in the network become stuck at 0, effectively losing the neuron from the network. This issue can be mitigated by using an alternate version of ReLU called Leaky ReLU which multiplies x by a hyper parameter as the lower limit to prevent the neuron becoming stuck at zero. Due to time requirements of shape estimations of the robot to be generated rapidly during medical procedures, ReLU was chosen to increase computation speed during prediction. The original ReLU was chosen over Leaky ReLU to avoid tuning the hyper parameter with the knowledge that the network could be further improved by changing this activation function later if results were insufficient. Lastly a linear activation function is used to perform regression on the final layer.

3.4 Hidden Layers

The number of neurons per hidden layer and number of hidden layers needed to be determined next. Determining the number of neurons per layer is not strictly defined but generally should be somewhere between the size of the input and size of the output. 10 neurons was chosen for the hidden layer size. The size of every hidden layer could be changed but each layer was kept the same size here to determine the optimum number of hidden layers. After deciding on 10 neurons per hidden layer, a model with 1 hidden layer was created and then the number of hidden layers was increased to find the optimal number of hidden layers by measuring the accuracy of the training model using cross validation. The number of epochs was set to 10,000 (epochs role in training the neural network will be discussed in section 3.5).

The existing 500 shape dataset was split into 4 subsets. 400 sets of input parameters Xtrn, 400 sets of output corresponding coefficients Ytrn, 100 sets of input parameters Xtst, and 100 sets of corresponding output coefficients Ytst. Xtrn is the input and Ytrn is the output that the models would be trained on, with Xtst and Ytst available to be used as a validation set for evaluation of the model later. At this stage we have:

- Xtrn, size 400x6, consisting of 400 samples of [Length1, Length2, Length3, Degree1, Degree2, Degree3], the input parameters for each shape. This will be the input training data for the model.
- Ytrn, size 400x90, consisting of 400 samples of [x1-x5, y1-y5, z1-z5], the output coefficients representing the shape. This will be the output training data for the models.
- Xtst, size 100x6, consisting of 100 samples of [Length1, Length2, Length3, Degree1, Degree2, Degree3], the input parameters for each shape. This will be the validation input data for evaluating the model's accuracy.
- Ytst, size 100x90, consisting of 100 samples of [x1-x5, y1-y5, z1-z5], the output coefficients representing the shape. This will be the validation output data for evaluating the model's accuracy.

5 neural networks were trained using Xtrn and Ytrn, each using 10,000 epochs, with identical input and output layers. In each mode, the number of hidden layers was changed to observe how much impact hidden layers had on improving accuracy. Each model was evaluated with the training accuracy based on how it scores with the training data, and then on the validation data which each model has not seen before. Accuracy in each model is measured using a loss function. The Mean Squared Error loss function was selected, which simply calculates the mean squared error between predicted output values and actual output values from Ytrn.

The number of epochs is the number of times the neural network will adjust the weights. Too few epochs can cause underfitting where the model has not had enough opportunities to tune the model to an accurate level. Too many epochs can cause overfitting where the model is too closely fitted to the training data and thus makes poorer predictions on unseen data in the validation data set. For these experiements to determine the optimal number of hidden layers, the number of epochs was set to 10,000 to

ensure none of the models were underfitted and each model could achieve some desirable level of accuracy. Then the number of epochs could be tuned further after the optimal number of hidden layers was determined.

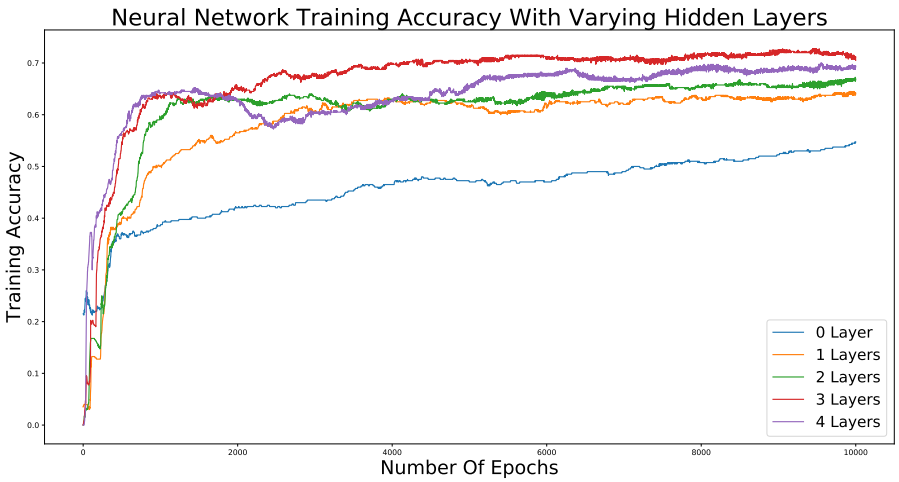

Figure 3.3: Hidden Layers Experiment 1 Graph

Layers	Training Acc	Testing Acc	Training Time (Seconds)
0	0.548	0.540	390.47
1	0.640	0.690	375.60
2	0.668	0.630	373.94
3	0.704	0.690	370.76
4	0.692	0.560	356.35

Figure 3.4: Table Of Results For Hidden Layers Experiment 1

It can be observed in Figure 3.3 that the model containing 0 hidden layers has a significantly lower training accuracy than all other models containing varying numbers of hidden layers. It can also be observed that the 1-hidden layer model's accuracy is further below the models containing 2, 3 and 4 layers, which are higher than the 1-hidden layer model and similarly around 0.66-0.70, indicating that multiple hidden layers outperforms 0 or 1 hidden layers. By visual inspection, the 3-layer hidden model has the highest training accuracy.

In Figure 3.4, we can observe that training time for each model took approximately 6 minutes to complete due to the unnecssairly large number of epochs. This was to ensure that number of epochs wouldn't be a limiting factor in this set of experiments. Whilst the 4-layer model has a high training accuracy and low training time, it has a significantly lower testing accuracy compared to the models containing 1,2 and 3 hidden layers. Some disparity between training accuracy and testing accuracy was expected in this experiment to the high number of epochs increasing chance of overfitting the model to the training data. However, the 4-layer's model having 13.2% difference

between training accuracy and testing accuracy implies overfitting. In this experiment, 3-hidden layer model seems to be the most sensible due to it obtaining the tied-highest testing accuracy on the unseen testing data set and having a the lowest disparity in training and testing accuracy of all the models, suggesting this model is reliable and not overfitted.

At first glance, all the testing accuracies in Figure 3.4 may appear lower than desirable since they mostly lie in the mid-60's range. However it is important to note that this neural network only predicts the curve coefficients and in order to truly evaluate the model's performance, the coefficients must be used to produce full-sized curves which will then be the basis for comparison between the original mathematical model for curve estimation and the machine-learning based technique.

The experiment was performed once more and the results were broadly similar (see Figure 3.5). Models with 1-3 hidden layers had a small decrease in testing accuracy whilst the 0 and 4-layer models had a small increase in testing accuracy. Notably, the 3-layer model performed very closely to the first experiment, achieving the highest testing accuracy again at 68% this time and only having a 0.7% difference between training and testing accuracy. This helped to confirm that the 3 layer model would be the best candidate for further evaluation.

Layers	Training Acc	Testing Acc	Training Time (Seconds)
0	0.548	0.570	378.38
1	0.652	0.640	374.63
2	0.647	0.620	373.29
3	0.673	0.680	383.61
4	0.663	0.600	360.40

Figure 3.5: Table Of Results For Experiment 2

3.5 Epochs And Measuring Loss

To select the optimal number of epochs, the loss value found against the validation set was plotted. The loss value is the total sum of mean squared error found between outputs predicted by the model using the validation input Xtst, and the expected outputs in the validation output Ytst. Figure 3.6 gives a clear indication of how each of the models improve over time and continuously attempts to decrease the total loss value. It is important to note that the model is not trained on this validation data, instead the model is evaluated against the validation data at every epoch during training to produce this loss value, but the loss value the model attempts to decrease is the one between the training data in Xtrn and Ytrn.

As observed in Figure 3.6, the loss value initially decreases rapidly and then gradually plateaus for each model. Interestingly, for the 3 and 4-hidden layer models, we can observe that loss actually begins to increase again on the validation set after 6000 epochs. This a sign of the models beginning to become overfitted to the training data at this point and beginning to perform worst with unseen future data. To prevent this, the

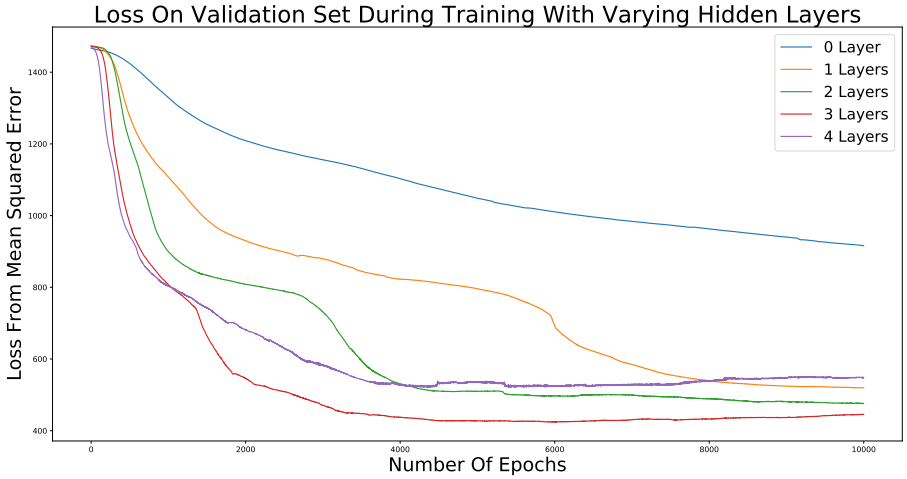

Figure 3.6: Loss Graph Produced using models from experiment 2

number of epochs will be decreased in the previously selected 3 hidden layer model to 6,000 epochs instead of 10,000, which will help increase accuracy by preventing overfitting and also reduce training time for the model.

3.6 Evaluating The Selected Model

The selected model was trained using 3 hidden layers, each containing 10 neurons. ReLU was the activation function for each layer, aside from the final layer which used the linear activation function. The input layer was 6 neurons representing the 6 input parameters and the output layer was 15 neurons representing the 15 output coefficients. The model was fitted to the training data of 400 sets of input parameters (Xtrn) and 400 sets of coefficients (Ytrn). Figure 3.7 visualises the final model's structure.

To predict 100 sets of coefficients using the 100 inputs from the validation input set (Xtst) took a mean time of 0.067 seconds per sample. The accuracy score for the validation set was 66% for the coefficients. One of the most important factors for controlling the concentric tube robot is knowing the current position of the tip as this is what the the user is ultimately trying to guide during control and use for interactions. To fully evaluate the viability of this model, the final point of the predicted shape must be close to the final point of the original shape. The shape estimation is also important and a helpful visual for controlling the robot, but a close estimation of the end point of the tip is the a key factor when deciding what to do next during control.

Using the 100 predicted sets of coefficients and making 100 predictions for arc-length using the linear regression model, 100 shapes were constructed from the coefficients, each consisting of 30 coordinates which could then be compared to the 100 original sets of coordinates for the same shapes that were created with the same input parameters. The mean tip error between the original shape's final coordinates and the predicted shape's final coordinates was found to be 24.08% with a standard deviation of

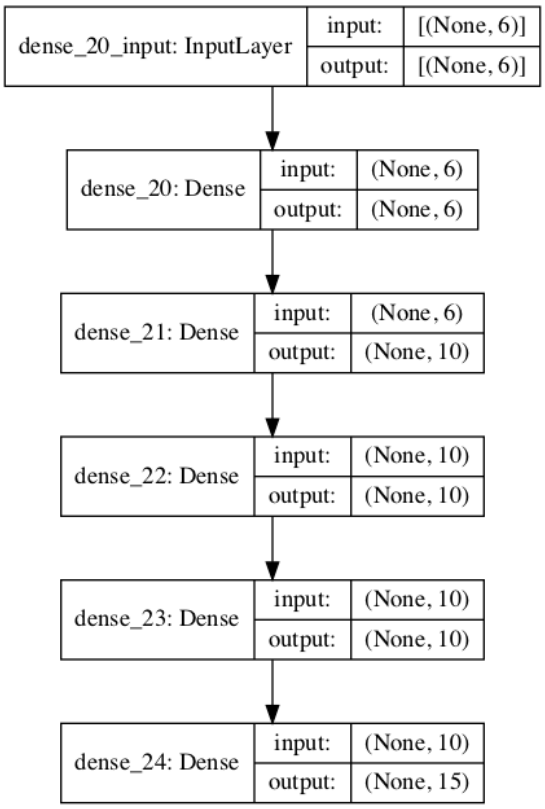

Figure 3.7: Neural Network Model Structure

18.81%. Since the neural network accuracy can vary each time it is compiled (unless one constant seed was used) due to a randomness factor with the starting weights, this procedure was run multiple times. The results in the following figures are from the run that produced the model with 66% accuracy for the coefficients discussed. The tip error across the 100 validation samples is shown in Figure 3.8.

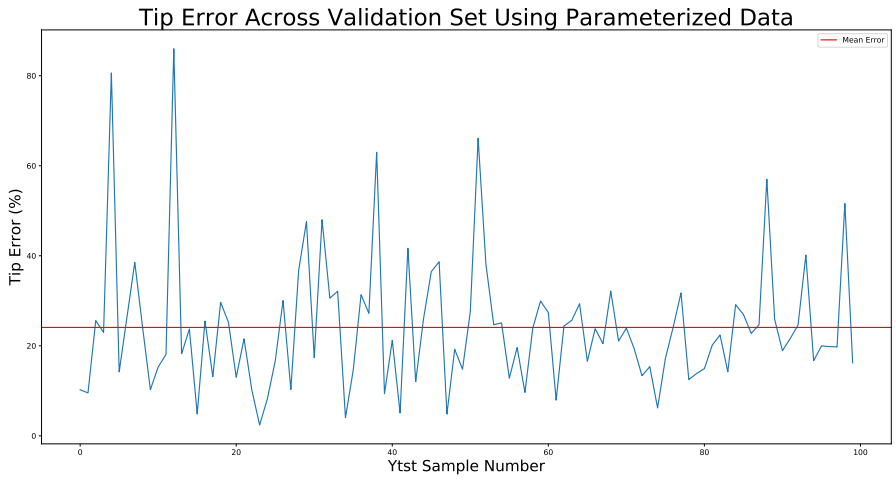

Figure 3.8: Tip Error Across The 100 Validation Samples

As seen in Figure 3.8, the majority of samples are below or close to the mean tip error with some notably larger outliers for this run of the model. The average arc-length in the test set was found to be 0.23m, giving a mean tip error of approximately 24.08% of the original arc-length of the curve. Existing physics-based model's have produced a maximum tip error of 8% of the robot's length [16]. The minimum error in the neural network model was found to be 2% which is a great improvement on the original mathematical model. However the maximum error in this test set is greater than 80% of the length of the original curve. This is a notable outlier for the test set. An assumption could be made regarding this outlier is that the shape was likely an unusual one that the model had not encountered enough in the 400 samples it had trained on. We believe by identifying weak points in the model, custom shapes could be added to the training set to mitigate these errors. However this model is able to predict a shape in 0.067 seconds, showing the model created can be highly efficient but also have a higher possibility of error which is a trade-off. The next step is to evaluate the predicted shapes visually to the original model.

3.7 Visual Comparisons With Original Model

In order to avoid cherry picking graphs to discuss and compare, the first 10 shapes were selected from the validation set to compare here, in order to avoid bias and discuss some accurate predictions and some predictions which are far off.

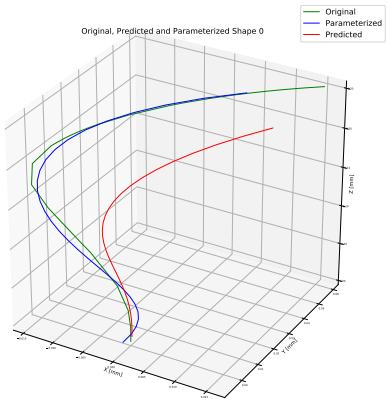

Figure 3.9: Shape 0

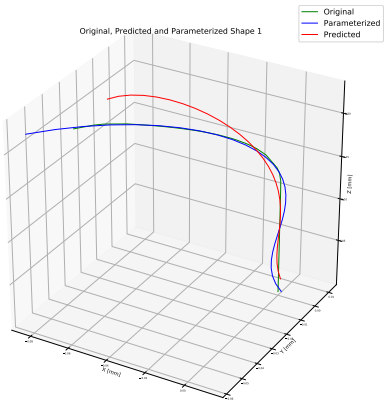

Figure 3.10: Shape 1

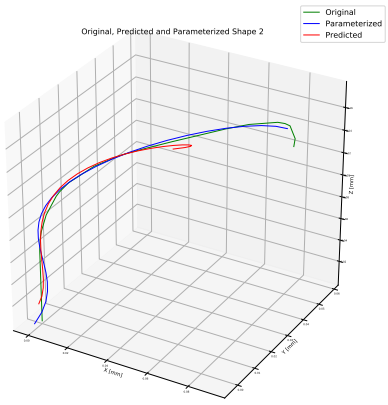

Figure 3.11: Shape 2

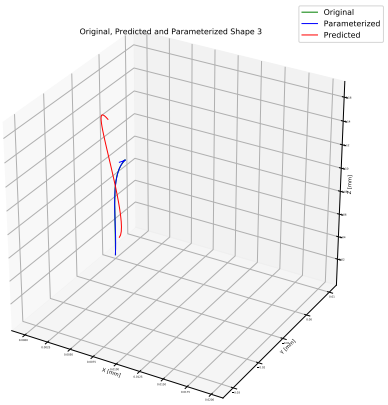

Figure 3.12: Shape 3

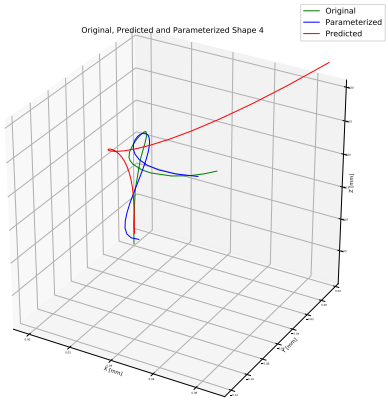

Figure 3.13: Shape 4

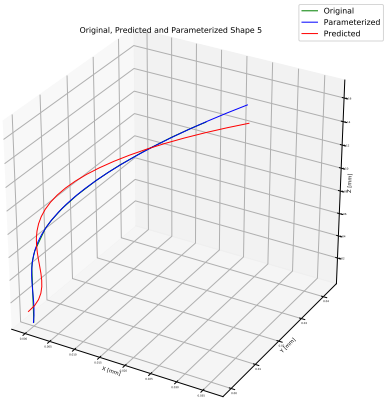

Figure 3.14: Shape 5

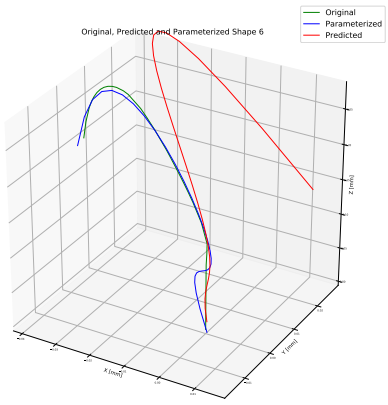

Figure 3.15: Shape 6

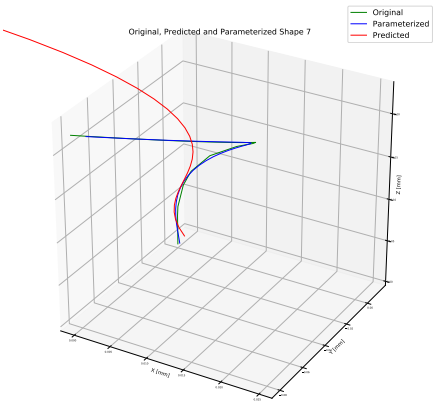

Figure 3.16: Shape 7

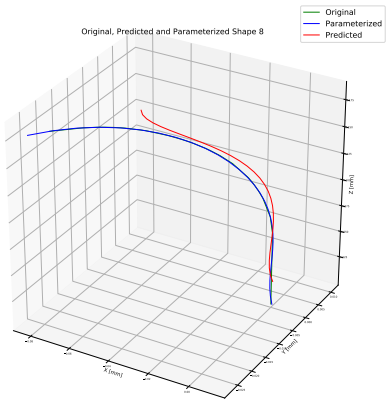

Figure 3.17: Shape 8

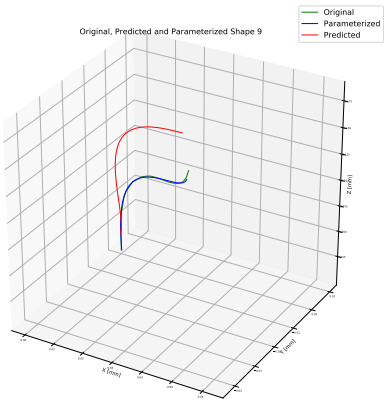

Figure 3.18: Shape 9

With Figures 3.9-3.16, it can firstly be observed that all 10 shapes that were produced through curve parameterization all closely match the original shape. These were all reconstructed using an arc-length predicted through linear regression and each shape looks nearly identical in length to the original shape, making a strong case for representing the curves in this way.

Continued visual inspection shows that shapes 0,1,2,3,5,8 and 9 match the original and parameterized shapes quite closely with some disparities in length, whilst shapes 4,6 and 7 differ significantly. Shape 4 is quite far off the original, but notably the original shape 4 is a very unusual shape that shows how the dataset was randomly generated as shape 4 would be a difficult shape to recreate with the real robot. Another interesting aspect about this shape is how the predicted shape managed to capture the loop in the middle of the shape. Shape 6 has similarities in shape to the original, but appears to be oriented in the wrong direction, pointing right rather than to the left. Shape 7's shape begins correctly and points in the right direction, but the angle and length are quite off.

Overall, 70% of these randomly selected shapes match the original curve to a good degree. Using a machine learning model guarantees much greater speed but observations show inconsistency in results. Interestingly length does appear to be a common issue in the predicted shapes, which is a result of arc-length beind predicted through linear regression rather than being given as an input. It is possible that calculating arc-length could be done in an alternate way and added as a 7th input to the model to help improve the predicted shapes.

Figures 3.19 shows the predicted arc-lengths and the real arc-lengths of the 10 shapes discussed. The degree of accuracy is high, but this is expected as there is not a huge amount of variation in the original arc-lengths. Figure 3.20 records the different euclidian distances between the models. Pred-Param is the distance between the final coordinate of the predicted shape and parameterized shape. Pred-Orig is the distance between the final coordinate of the predicted shape and original shape. Param-Orig is the the distance between the final coordinate of the parameterized shape and original shape. Each of these can be divided by the original arc-length and multiplied by 100 to find the tip error as a percentage of the entire length of the shape.

Shape Number	Predicted Arc Length (mm)	Original Arc Length (mm)
0	0.2715	0.2879
1	0.2825	0.26780
2	0.2465	0.2685
3	0.1652	0.1505
4	0.3702	0.3793
5	0.1725	0.1602
6	0.2996	0.2970
7	0.2600	0.2639
8	0.2216	0.2150
9	0.1964	0.2131

Figure 3.19: Table Of Results For First 10 Shapes - Arc-Lengths Of Predicted VS Original

Shape Number	Pred-Param (mm)	Pred-Orig (mm)	Param-Orig (mm)
0	0.01878	0.02949	0.01708
1	0.03588	0.02553	0.01913
2	0.05066	0.06872	0.02522
3	0.02217	0.03459	0.01446
4	0.3062	0.3058	0.0.01156
5	0.01684	0.02276	0.01217
6	0.08248	0.07837	0.007470
7	0.1007	0.1017	0.003223
8	0.05681	0.05184	0.006727
9	0.02125	0.02187	0.01689

Figure 3.20: Table Of Results For First 10 Shapes - Euclidian Distances Between The Tip Of The Different Models

3.8 Non Parameterized VS Parameterized Model

Observing the same 10 shapes again using the original non-parameterized neural network model, we can visually compare how the parameterized model shows improvements over the non-parameterized model:

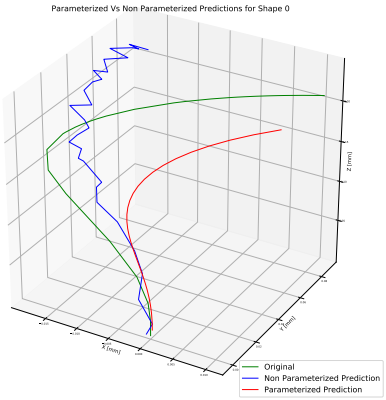

Figure 3.21: Shape 0

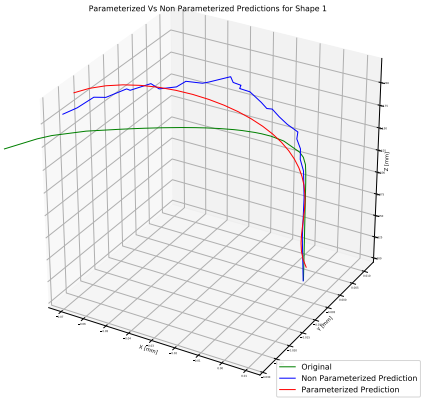

Figure 3.22: Shape 1

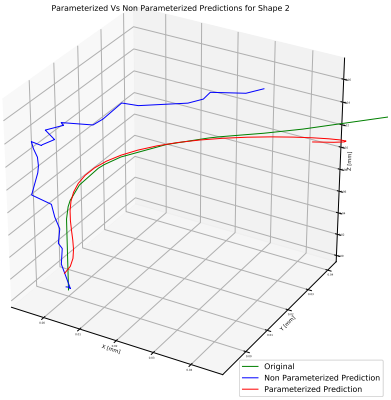

Figure 3.23: Shape 2C

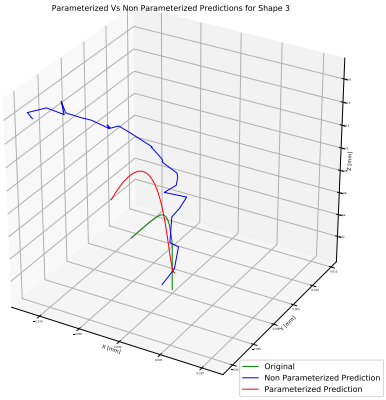

Figure 3.24: Shape 3

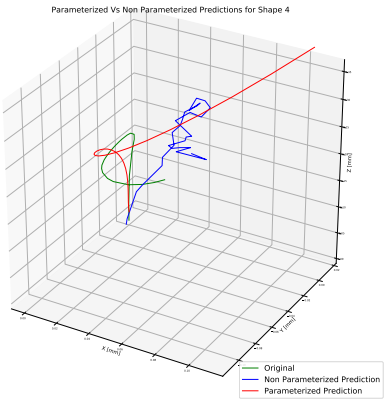

Figure 3.25: Shape 4

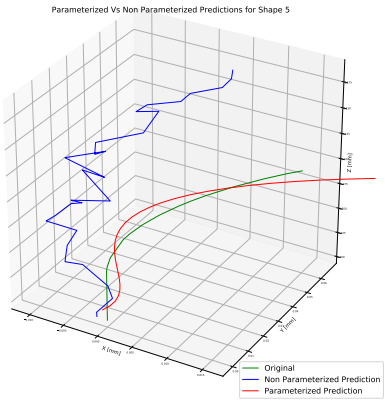

Figure 3.26: Shape 5

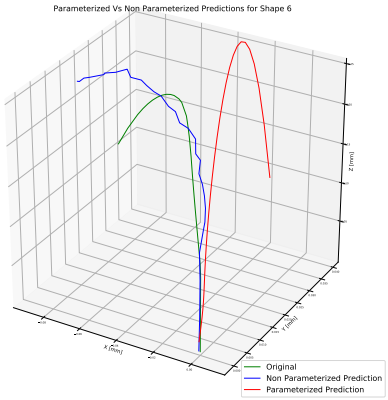

Figure 3.27: Shape 6

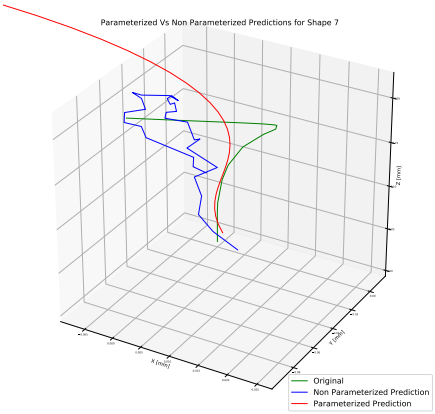

Figure 3.28: Shape 7

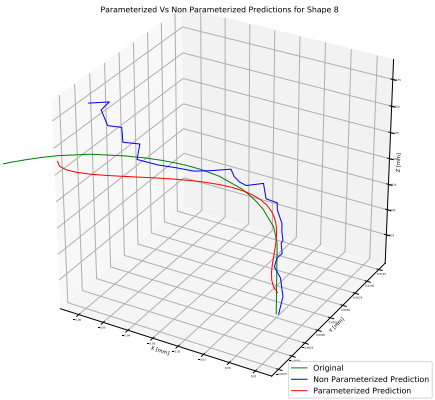

Figure 3.29: Shape 8

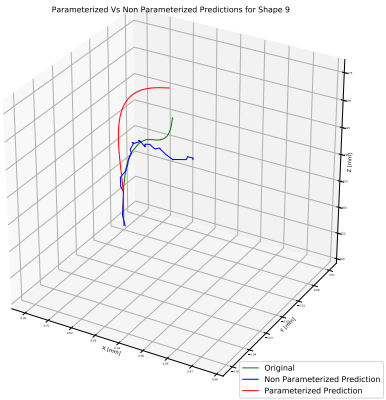

Figure 3.30: Shape 9

Visual inspection of figures 3.21-3.30 show that non-parameterized predictions are all very 'jittery', even when the shape is correct. This is due to 90 different points being predicted, causing the plot of the shape to constantly jump along the line. When the predictions are very inaccurate in the non-parameterized model, the shapes produced appear as a random scribble such as in shape 7. In shape 7, whilst both predicted shapes are not close to the original in this instance, the parameterized model produces a smooth curve, with some of the original properties intact, such as the starting position and the tip pointing left. In contrast, the non-parameterized model appears almost completely random and the shape appears like one that would not even be possible to produce using the original mathematical model.

Shape 1 and shape 6 are the only 2 shapes where we can observe that the non-parameterized model outperforms the parameterized model. The other 8 shapes in this collection show a much better performance by the parameterized model than the non-parameterized model, except for shapes 4 and 6 where both the predicted models are quite far off the original. Interestingly, the non-parameterized model produced a 7.48% lower tip error than the parameterized model at 16.60%. The tip error of the non-parameterized neural network model across the validation set can be observed in Figure 3.31.

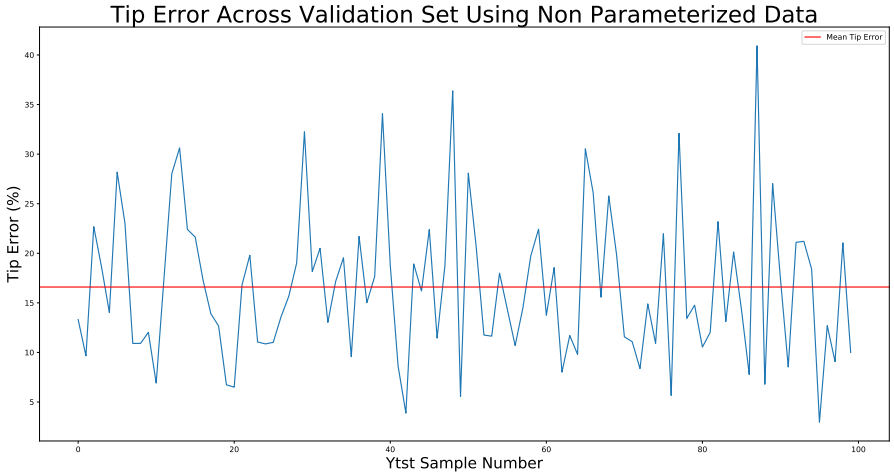

Figure 3.31: Tip Error For Non Parameterized Model

The maximum tip error was found to be 40.92% and the minimum tip error was found to be 2.97%. The mean tip error was 16.60%. Despite this improvement in tip error for the non-parameterized model, the very poor shapes produced when plotted outweigh the benefit of improved tip error, especially when directly compared to the shapes produced by the parameterized model.

3.9 Efficiency Of Neural Network Model Compared To Physics-Based Model

Finally, the efficiency of the neural network model needed to be evaluated and compared with the original physics-based model. 50 inputs were given to the the neural network model. The timer began as soon as the input was given and the timer was stopped as soon as a curve was plotted. The same process was done using the existing physics-based model that had previously been used for data generation. 50 times were recorded for each model and shown in the box plot in Figure 3.32.

The mean time for the neural network model was 0.19 seconds. The minimum time was 0.16 seconds and the maximum time was 0.32 seconds. The standard deviation was 0.03 seconds.

The mean time for the physics-based model was 5.08 seconds. The minimum time was 1.60 seconds and the maximum time was 43.15 second seconds. The standard deviation was 6.21 seconds.

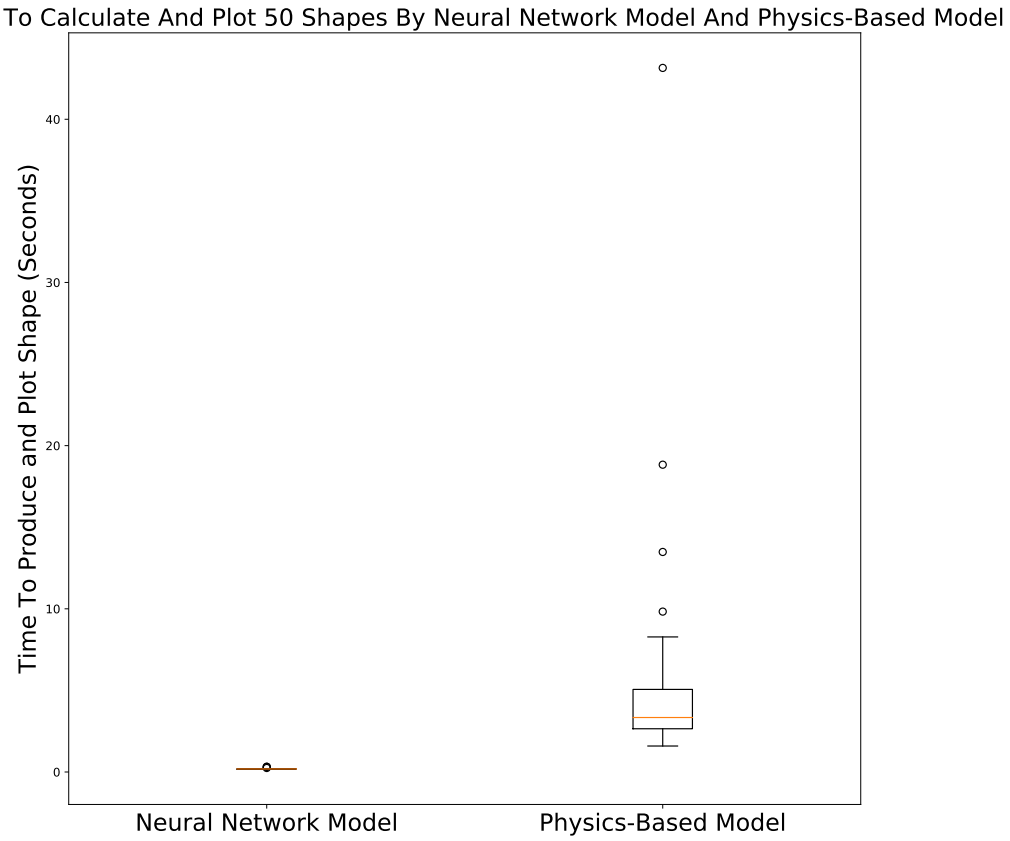

Figure 3.32: Box Plot of Shape Estimation Times Across 2 Models

The neural network model beat the physics-based model for all 50 inputs and had no significant outliers like the 4 observed in the box plot of the physics based model. Figure 3.32 also highlights the consistency of the neural network model's performance, with an extremely low standard deviation, in contrast to the physics based model's larger interquartile range. Overall the neural network models achieves a mean time reduction of 96.85% compared to the physics-based model.

Chapter 4

Conclusion

4.1 Final Results Discussion

Firstly, this report found that representing a curve by its x,y and z coefficients using curve parameterization using the curve's arc-length works exceptionally well and greatly reduces the amount of data required to represent a curve whilst maintaining a highly accurate representation of the shape. Knowing the curve's arc length allows us to plot coordinates of the curves at each point along the journey from the start to the end of the arc-length. These calculations were also found to be very fast to perform, improving computational efficiency of the model.

Secondly, this report found that linear regression is very fast and easy method to use to predict the arc-length of the curves based on the input parameters and highly accurate, achieving an accuracy of 91.74%. Given coefficients that can be predicted accurately and quickly, this means that images of shape of the robot can be reconstructed using the coefficients very quickly using the predicted arc-length and predicted coefficients.

Thirdly, this report found that neural networks are a strong method for predicting coefficients, as they can make fast, accurate predictions, but this report also found that that results can vary greatly depending on how the network is tuned and the quality/size of the data it has been trained on. When plotting the first 10 curves, 70% of the reconstructed images made from predicted coefficients were nearly identical to the original curve, whilst 30% had inaccuracies.

It could be theorised that by training the network on a relatively small number of samples (400 in this case), there are very common, traditional shapes that the neural network can easily predict but also much rarer shapes observed that start, curve and end in strange ways which the model isn't able to predict as accurately. This could possibly be mitigated through further analysis of wrongly predicted shapes, identifying patterns of which unusual shapes are predicted wrong, generating many of these types of shapes using the mathematical model and ensuring the neural network model is trained on these edge cases.

4.2 Discussing Medical Viability

This project's vision for how the machine learned model would ideally be used is as a side monitor for use during bronchoscopy surgery. When the surgeon controls the robot electronically, the input would be automatically sent to the model to predict a shape based on previous training. The surgeon could see the predicted shape of the robot inside the bronchoscope and use the image to help them decide their next control/movement. The prediction would ideally be based off of a large history of shapes the model has trained on, but also give a higher weighting to the previously predicted shape in the prediction in order to predict the new shape continuously based off previous outputs.

The current neural network model is more naive and is trained off simulated data, rather than real-world data using the actual robot. When generating the data set using the simulated model of the robot, these calculations required were computationally too demanding for a 2015 computer used at times, with memory running out constantly. A single image could sometimes take upwards of 40 seconds to produce, which would not be viable for live surgery. In contrast, the neural network model built here could make a prediction in a fraction of a second and output a reconstructed curve in less than one second, making this viable during live-use.

However for use during a medical procedure, whilst time is a very important factor during surgery and the machine learning model could viably keep producing images during live-use, accuracy is most important. A machine learned model would need much further, vigorous testing before being trusted to be part of a medical procedure due to the fact it makes predictions and estimations rather than traditional calculations. This report's results show some promise, with many predicted curves being largely accurate to the original and having a tip error as low as 2% from the original shape, but there are outlier results where the prediction isn't accurate enough. A surgeon could likely easily identify an unexpected output and mitigate this, but it is a risk that must be considered. A more advanced model would need to be thoroughly tested with a physical robot and many rapid inputs. The more advanced model would need to be trained on far more training samples to account for all possibilities. This could be done by generating data from actual usage of the physical robot.

4.3 Future Expansion

If this project were to be continued further, an interesting idea would be to take a deeper look at each outlier prediction in the model and search for a pattern. Perhaps the outliers are unusual shapes and more shapes similar to the outlier could be added to the training data so the neural network model is more prepared. One could study how increasing the number of samples in the training data and testing data would affect accuracy results and use these to tune the existing model further and hopefully discover more outlier and edge-case shapes.

Additionally, one could investigate if the machine learned model could have the error rate of the mathematical model built in so predictions could be made with the possibil-

ity of error taken and in and perhaps more accurate results could be produced. Another way to continue developing the neural network model would be to tune the number of coefficients. Going from 90 points to 15 is a significant reduction and whilst this works well for representing simpler curves, it's interesting to wonder how much impact the coefficients have on more complicated curves and if using 15 coefficients would truly capture all aspects of a shape with many different twists and curves.

It would also be helpful to generate data from the physical concentric tube robot that exists in the laboratory. This data could then be further compared with the mathematical model and tuned appropriately. Data has been generated using cameras on a concentric tube robot before. To make a neural network model using this source of data would be interesting, and it would be good to validate if curve parameterization would work as well with the data produced physically as it does with the simulation.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Hessa Alfalahi, Federico Renda, and Cesare Stefanini. Concentric tube robots for minimally invasive surgery: Current applications and future opportunities. *IEEE Transactions on Medical Robotics and Bionics*, 2(3):410–424, 2020.

[3] Anurag Bhardwaj, Wei Di, and Jianing Wei. *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*. Packt Publishing Ltd, 2018.

[4] C Sidney Burrus, James W Fox, Gary A Sitton, and Sven Treitel. Horner's method for evaluating and deflating polynomials. *DSP Software Notes, Rice University, Nov*, 26, 2003.

[5] Atoosa Davarpanah, Mostafa Khazaee, Amin Moslemi, and SE Khadem. Optimization of concentric tube continuum robot based on accuracy and overall length of the robot via genetic algorithm. *Annals of Robotics and Automation*, 4(1):007–012, 2020.

[6] David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.

[7] Michael S Floater and Tatiana Surazhsky. Parameterization for curve interpolation. In *Studies in Computational Mathematics*, volume 12, pages 39–54. Elsevier, 2006.

[8] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[9] Keshav Iyengar, George Dwyer, and Danail Stoyanov. Investigating exploration for deep reinforcement learning of concentric tube robot control. *International Journal of Computer Assisted Radiology and Surgery*, 15:1157–1165, 2020.

[10] Mohsen Khadem, Lyndon Da Cruz, and Christos Bergeles. Force/velocity manipulability analysis for 3d continuum robots. In *2018 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS), pages 4920–4926. IEEE, 2018.

[11] Alan Kuntz, Armaan Sethi, Robert J Webster, and Ron Alterovitz. Learning the complete shape of concentric tube robots. *IEEE Transactions on Medical Robotics and Bionics*, 2(2):140–147, 2020.

[12] Jesse Lock and Pierre E Dupont. Friction modeling in concentric tube robots. In *2011 IEEE International Conference on Robotics and Automation*, pages 1139–1146. IEEE, 2011.

[13] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[14] Tania K Morimoto, Elliot Wright Hawkes, and Allison M Okamura. Design of a compact actuation and control system for flexible medical robots. *IEEE robotics and automation letters*, 2(3):1579–1585, 2017.

[15] Yoram Reich and SV Barai. Evaluating machine learning models for engineering problems. *Artificial Intelligence in Engineering*, 13(3):257–272, 1999.

[16] D Caleb Rucker, Bryan A Jones, and Robert J Webster III. A geometrically exact model for externally loaded concentric-tube continuum robots. *IEEE transactions on robotics*, 26(5):769–780, 2010.

[17] Alan Saalfeld. Topologically consistent line simplification with the douglas-peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18, 1999.

[18] Philip J Swaney, Arthur W Mahoney, Andria A Remirez, Erik Lamers, Bryan I Hartley, Richard H Feins, Ron Alterovitz, and Robert J Webster. Tendons, concentric tubes, and a bevel tip: Three steerable robots in one transoral lung access system. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5378–5383. IEEE, 2015.

[19] Alessandro Vandini, Christos Bergeles, Ben Glocker, Petros Giataganas, and Guang-Zhong Yang. Unified tracking and shape estimation for concentric tube robots. *IEEE Transactions on Robotics*, 33(4):901–915, 2017.