FORWARD KINEMATICS SOLUTION USING NEURAL NETWORK FOR A STEWART PLATFORM

A Project Presented to the Faculty of San Diego State University

Nakul Narwaria
Nakulnarwaria17@gmail.com

1. Problem Statement

A hexapod (any standard mechanical contraption with 6 legs) has been seen to be related to many real-life problems, especially in the field of robotics and film animation. The study of Hexapod manipulation (Stewart platform) has hence been of considerable importance.

Figure 1: A Hexapod

In this project, we use the Inverse Kinematics approach for attaining the forward kinematic solution for a 6 Degree-of-Freedom Stewart Platform. We've used the ranges of [-1,1], [-1,1] and [5,10] for pose generation as inputs.

A straightforward set of 6 algebraic equations (Kai Lu et al. 1993) is being used for computation of the neural network required for estimating the lengths of legs of the platform, given by:

$$L_1 = \sqrt{\left(X_{T1} - \frac{d}{2\sqrt{3}} - \frac{b}{\sqrt{3}}\right)^2 + \left(Y_{T1} - \frac{d}{2}\right)^2 + Z_{T1}^2}$$

$$L_2 = \sqrt{(X_{T1} - \frac{d}{2\sqrt{3}} + \frac{b}{2\sqrt{3}})^2 + (Y_{T1} - \frac{d}{2} - \frac{b}{2})^2 + Z_{T1}^2}$$

$$L_3 = \sqrt{(X_{T2} + \frac{d}{\sqrt{3}} + \frac{b}{2\sqrt{3}})^2 + (Y_{T2} - \frac{b}{2})^2 + Z_{T2}^2}$$

$$L_4 = \sqrt{(X_{T2} + \frac{d}{\sqrt{3}} + \frac{b}{2\sqrt{3}})^2 + (Y_{T2} + \frac{b}{2})^2 + Z_{T2}^2}$$

$$L_5 = \sqrt{(X_{T3} - \frac{d}{2\sqrt{3}} + \frac{b}{2\sqrt{3}})^2 + (Y_{T3} + \frac{b}{2} + \frac{d}{2})^2 + Z_{T3}^2}$$

$$L_6 = \sqrt{(X_{T3} - \frac{d}{2\sqrt{3}} + \frac{b}{\sqrt{3}})^2 + (Y_{T3} + \frac{d}{2})^2 + Z_{T3}^2}$$

2. Clustering

The positions (P_x, P_y, P_z) and orientation (α, β, γ) of the Stewart platform attained from the input ranges as defined as used. A two-dimensional array holding all possible combinations of position-orientation values is used as a holding platform. The clustering algorithm used here is K-Means. The K-Means is a well-known unsupervised clustering algorithm for unlabeled data. This suits us well as we are just using the (P, Θ) combinations to be inherently grouped into classes.

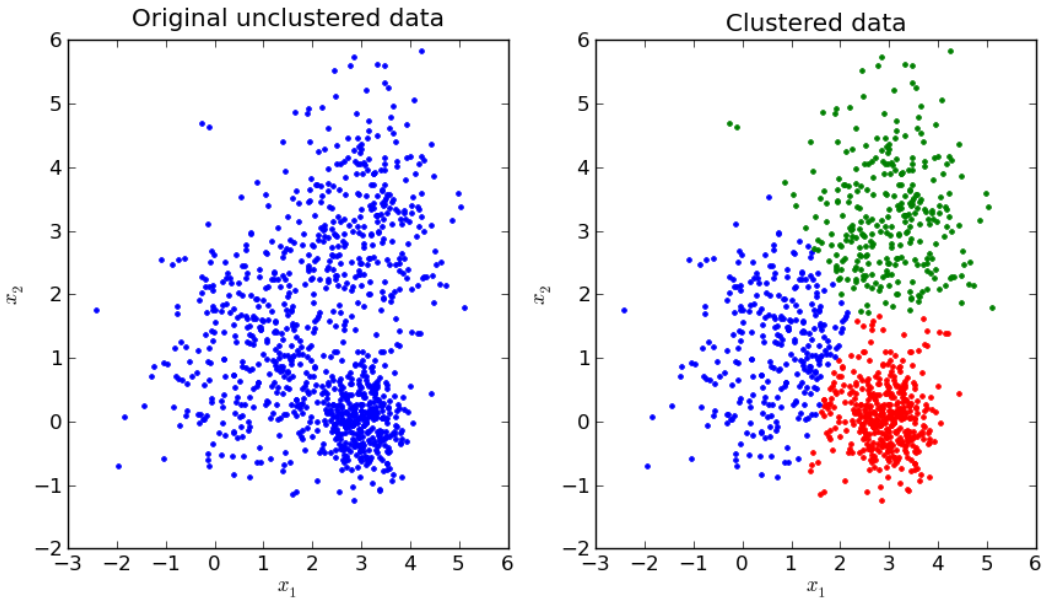

Figure 2: k-means clustering

K-Means takes the required cluster count (k), threshold (τ), and the set of features (X_1, X_2,..,X_N) as input. The general algorithm for this is as follows:

- Use random vectors of means $m_t = \{m_1, m_2, \ldots, m_n\}^T$ for each cluster as initial values of means.
- Calculate minimum distance or distortion D (Eucledean/Mahalanabalis) for each sample point with respect to current means.
- If D_old/D > τ
    - Replace old D with D.
- Re-compute m_t using new value of D.
- Continue until threshold is not reached.

Basic formula:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

The K-Means algorithm receives the 2-D array as input with the number of required clusters (k) set to 4. This produces the position-orientation values as categorized into 4 classes, which were produced by the clustering.

Applying the 2-D array holding all the combination values to the equations succeeds in providing us with the corresponding lengths of each combination. It is important to note here that the index values of the array and the corresponding lengths remain the same. This is stored in another 2-D array.

Once the indices of the length values have been attained by clustering and since the configuration of both the previous 2-dimensional array as well as the classes are the same, the attained values are applied to the combinations held by the combination array.

We aim to create a classification model such that when the length values are provided as input, our classifier should be able to produce the precise position-orientation corresponding to that length value. Note that we are calculating the inverse using the Inverse Kinematic equations, but the pairings should provide us with enough data to create a reasonable classifier.

The means of creating the classifier used is a Feed-Forward Neural Network.

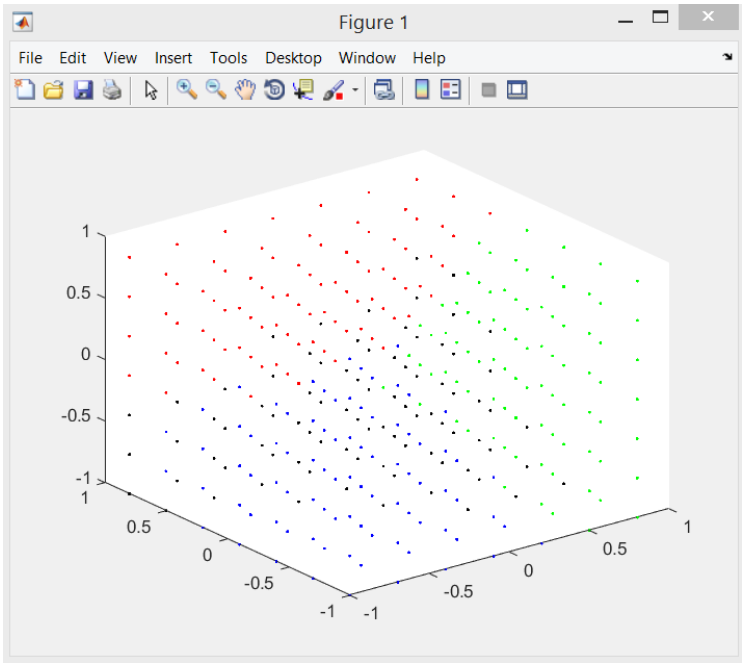

Figure 3.1 : Positions clustered into 4 clusters

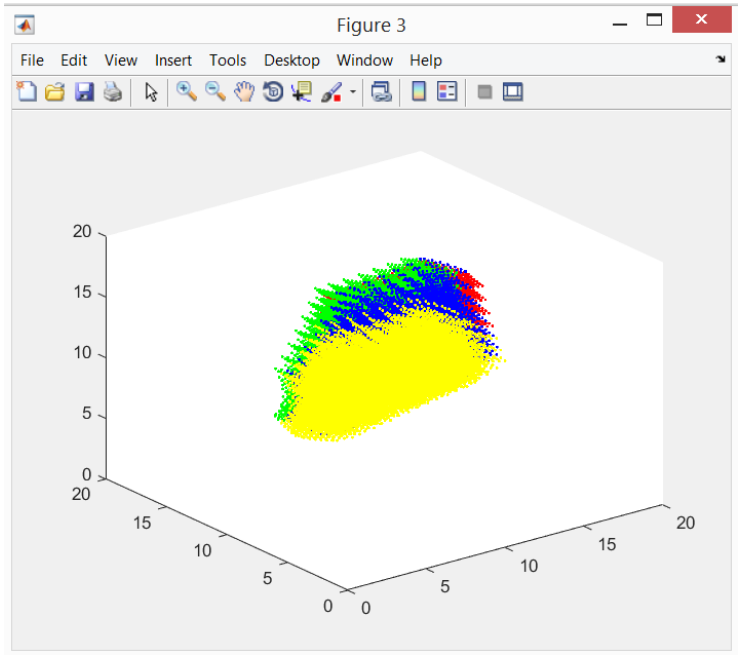

Figure 3.2 : Corresponding first 3 lengths clustered with same index as positions

3. Classification and Training

To completely understand the classification problem at hand, a review of the basic Stewart Platform movements is required, which is given below:

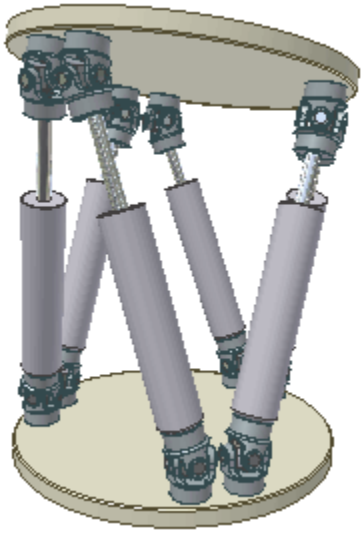

Figure 4: Basic Stewart Platform

This is a type of parallel robot that has six actuators attached to the platform's baseplate in 3 positions, with corresponding three mounting points on the top plate. Any freely-suspended object placed on the platform is allowed to move in six degree of freedoms, which are the three plausible positions along with three possible corresponding rotations. Our positions are denoted by $P(P_x, P_y, P_z)$ and the rotations by $\Theta(\alpha, \beta, \gamma)$.

A neural network is essentially a modeling of the primitive functions of the human brain, namely processing and passing of information signals, while deriving some sort of information from it. A basic model consists of 1 input layer, 1 or more *hidden layers*, and 1 output layer. Each hidden layer has n-nodes, which act as neurons for processing the inputs.

Each node acts as the basic processing unit for the neural network. All layers consist entirely of nodes, and these nodes can be assigned values and descriptions according to the function they are desired to emulate.

Inputs with specific dimensions are passed into the input layer, which in turn pass it to the consecutive hidden layers. The hidden layers are defined with specific parameters and an *activation function*, which define a specific way to gleam information from the inputs. Finally, the output layer has relevant labels to which the inputs are required to be mapped.

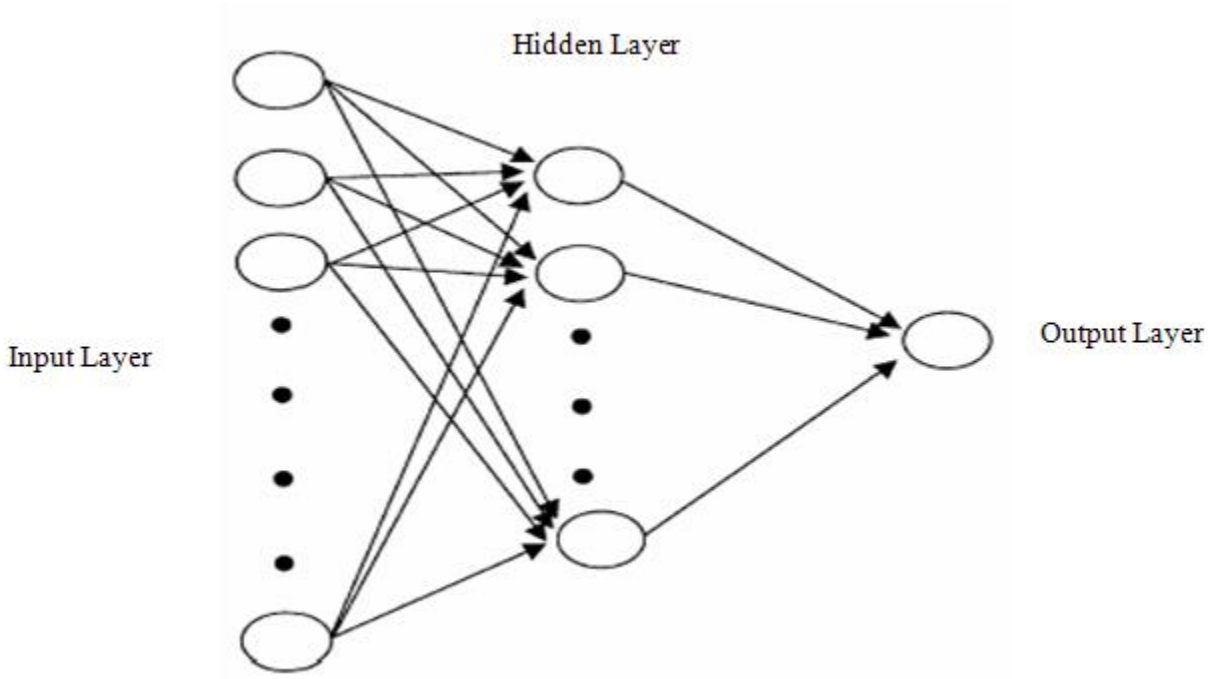

Figure 5: A Neural Network

A feed-forward neural network simply implies that the flow of data in our model is from front to end, i.e. the inputs are being ‘fed’ from input layer, passing through hidden layers, and reaching the output layer.

For our requirements, we are using a feed-forward neural network with 5 hidden layers, each of them having 10 nodes. The activation function being used is ‘Bayesian Regularization’.

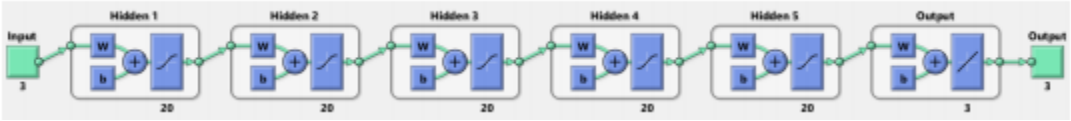

Figure 6: Neural Network Model

For the purpose of our training, we pass the length values attained from the aforementioned equations as inputs, and the (P, Θ) pairings as labels for inputs to be trained over. It is important to note that for each length, there can be 4 possible candidates for the (P, Θ) pairings corresponding to it, due to there being 4 categorized classes of those pairings.

Once the neural network has been trained over and the classifier created, we use it to predict the labels for random positional lengths passed into it and test its accuracy.

4. Results

The solution for forward kinematics for hexapod platform is obtained using the neural network in MATLAB. First, the lengths of all six legs are computed for each combination of position and theta that the input matrix holds, and then the clustered data, using k-means is used to train the neural network for training.

The Input Parameters for the Hexapod are -30>α>30, -30>β>30, -30> γ >30, -1>px>1, -1>py>1, 52>pz>10 (all angles converted to radians), and a=10, b=15, d=1. Using these parameters, we define the inverse kinematics equations. The size of lengths matrix obtained from these equations is 117649x6. Then we use the k-means to partition the data matrix into 4 clusters/classes into data matrix of size 6x29498 , 6x32585 , 6x29841 , 6x25725 respectively. We train all these clusters separately and determine the forward kinematics solution. The input layer and output layer are:

- Class 1 is l1class1, l2class1, l3class1, l4class1, l5class1, l6class1 and px1, py1, pz1, α1, β1, γ1.
- Class 2 is l1class2, l2class2, l3class2, l4class2, l5class2, l6class2 and px2, py2, pz2, α2, β2, γ2.
- Class 3 is l1class3, l2class3, l3class3, l4class3, l5class3, l6class3 and px3, py3, pz3, α3, β3, γ3.
- Class 4 is l1class4, l2class4, l3class4, l4class4, l5class4, l6class4 and px4, py4, pz4, α4, β4, γ4.

The performance obtained are 5.93e-05, 0.0001430, 0.000171, and 0.000189 for 4 neural networks. Since output of the neural networks contains 3 position values and 3 orientation

values, separate mean squared error for both position and orientation is obtained for all 4 neural networks separately. These mean squared error are as follows –

- Class 1 – position error - 0.0391 $meters^2$, orientation error – 0.0017 $radians^2$.
- Class 2 – position error - 0.0112 $meters^2$, orientation error – 0.0016 $radians^2$.
- Class 3 – position error - 0.0056 $meters^2$, orientation error – 0.0012 $radians^2$.
- Class 4 – position error - 0.0343 $meters^2$, orientation error – 0.0042 $radians^2$.

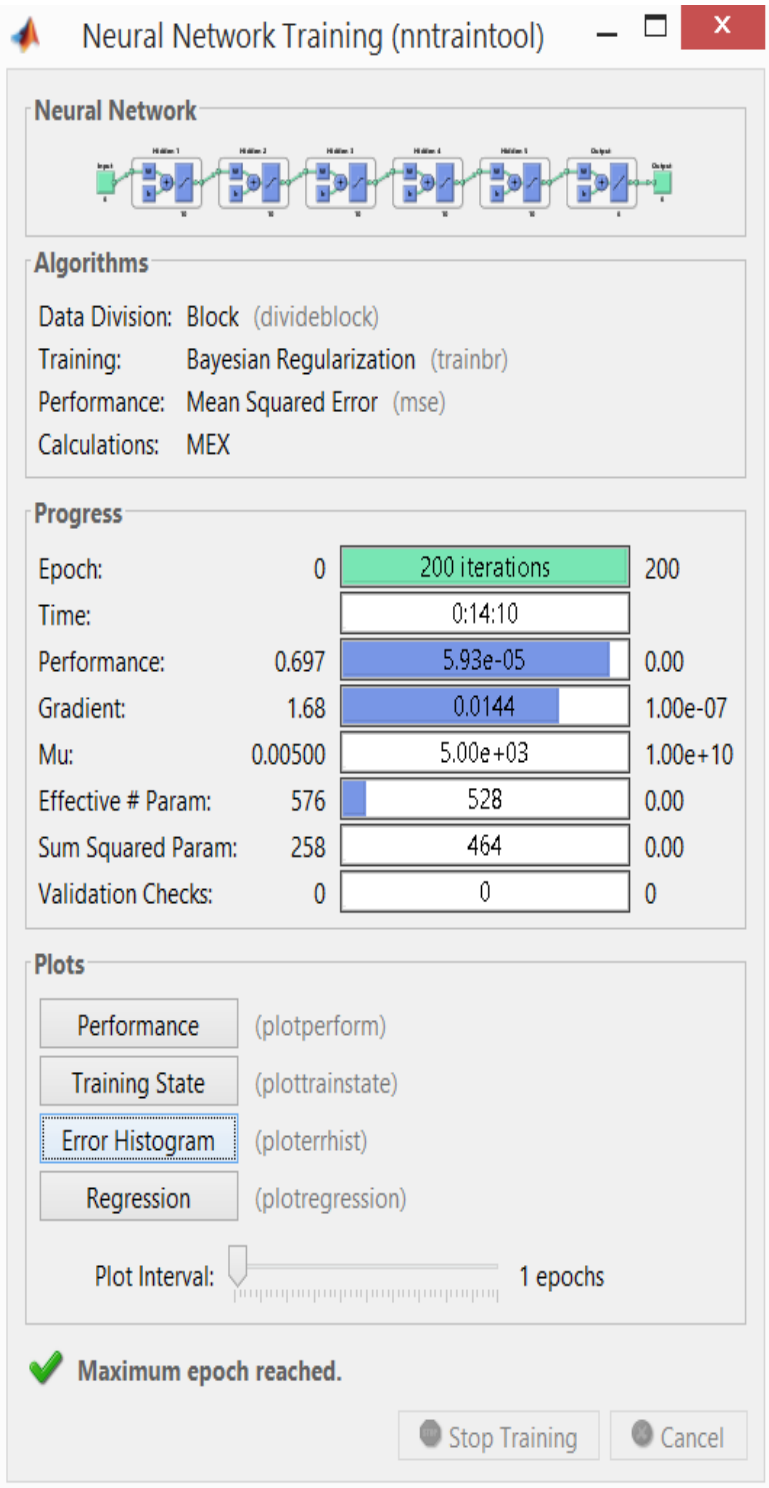

Figure 6.1: Neural Network parameters for class 1

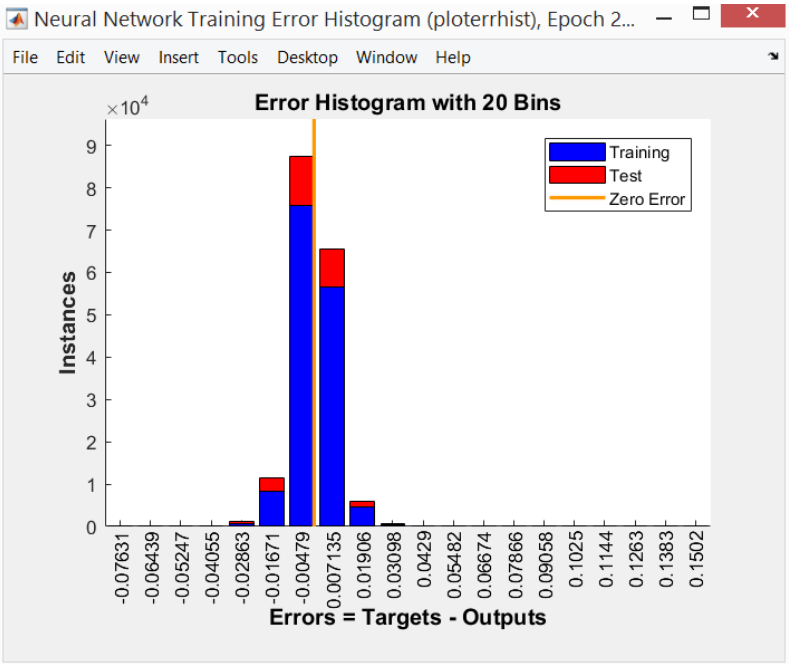

Figure 6.2 : Error histogram for neural network 1

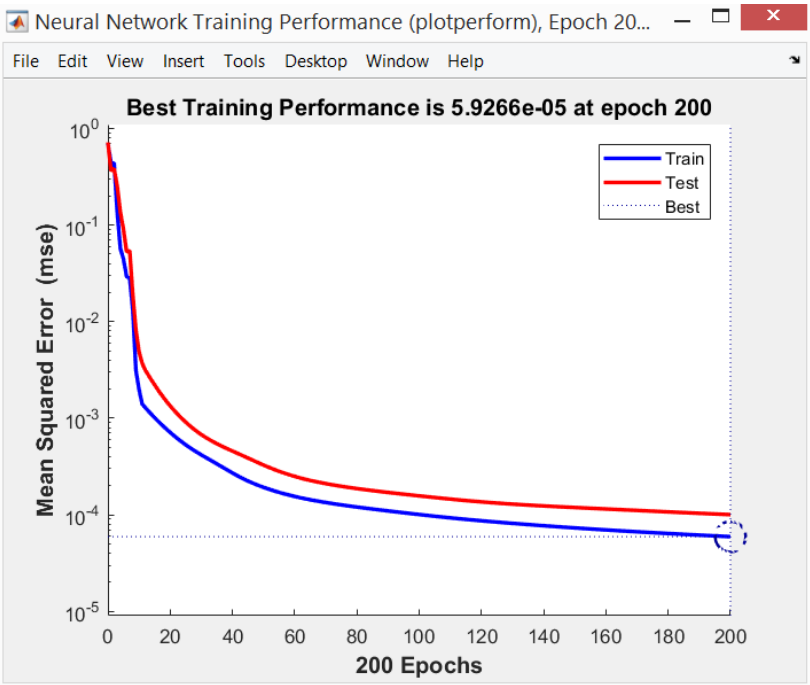

Figure 6.3: Performance of neural network for class 1

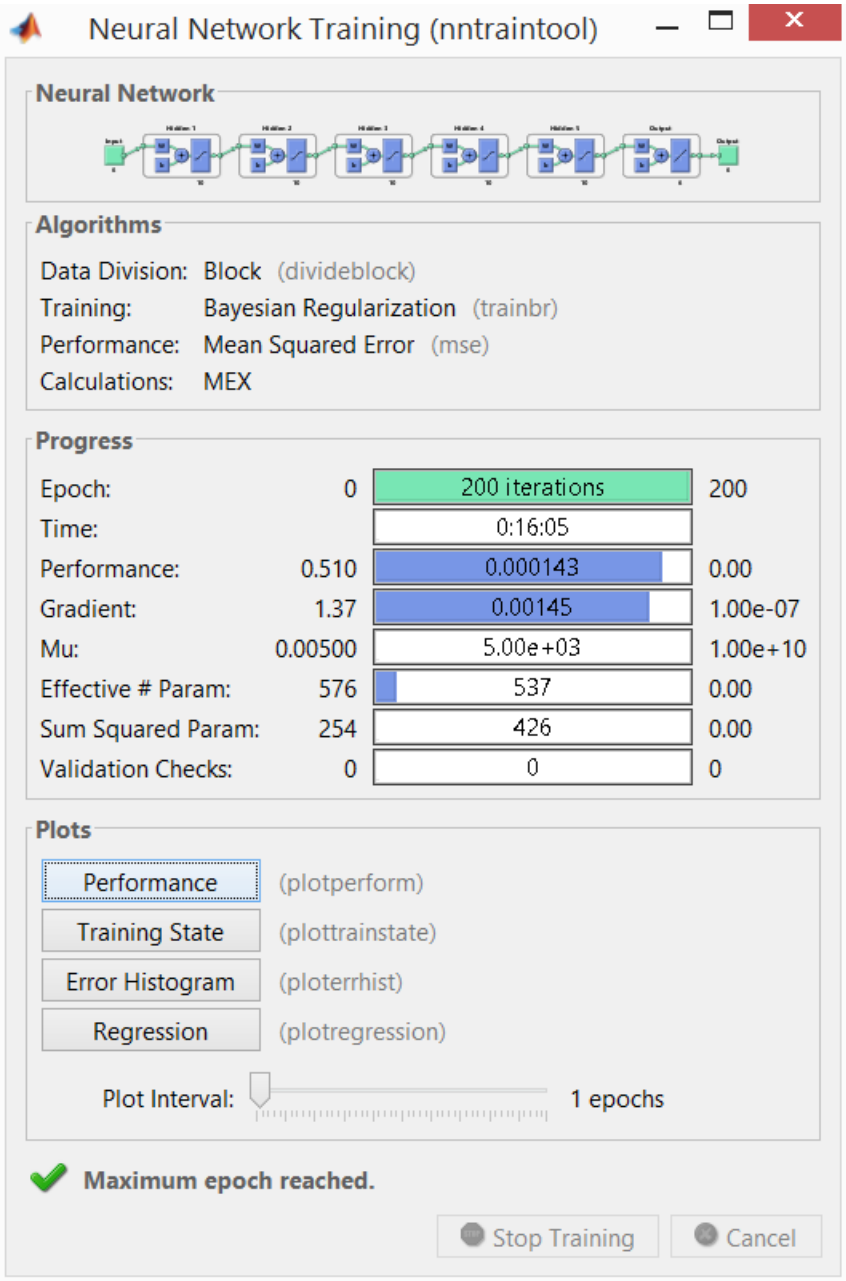

Figure 7.1: Neural Network parameters for class 2

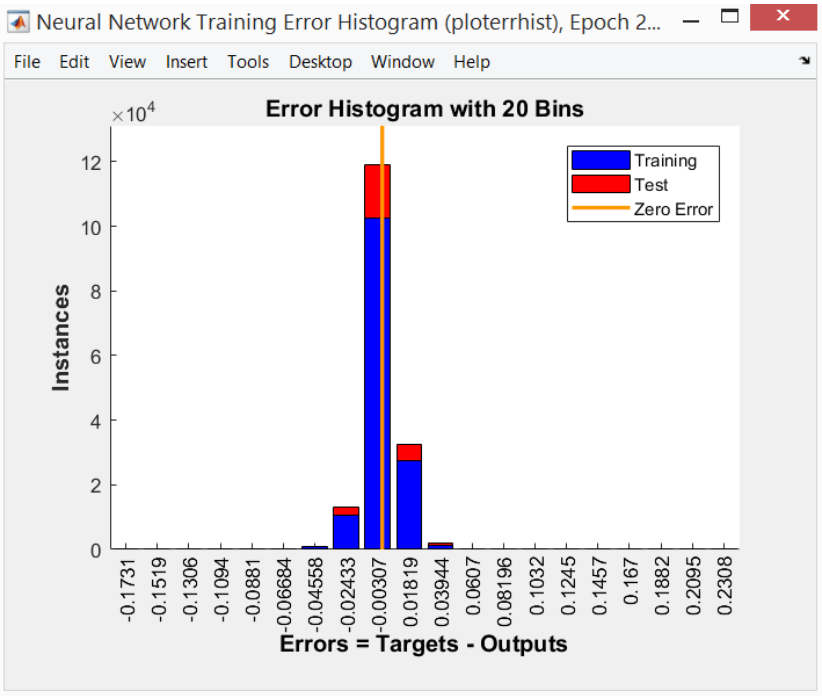

Figure 7.2: Error histogram for neural network 2

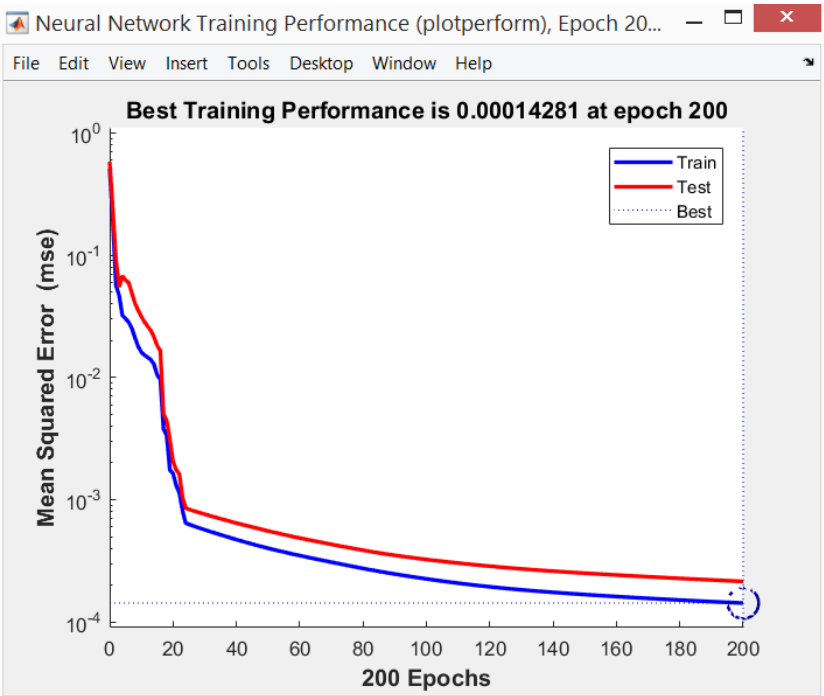

Figure 7.3: Performance of neural network for class 2

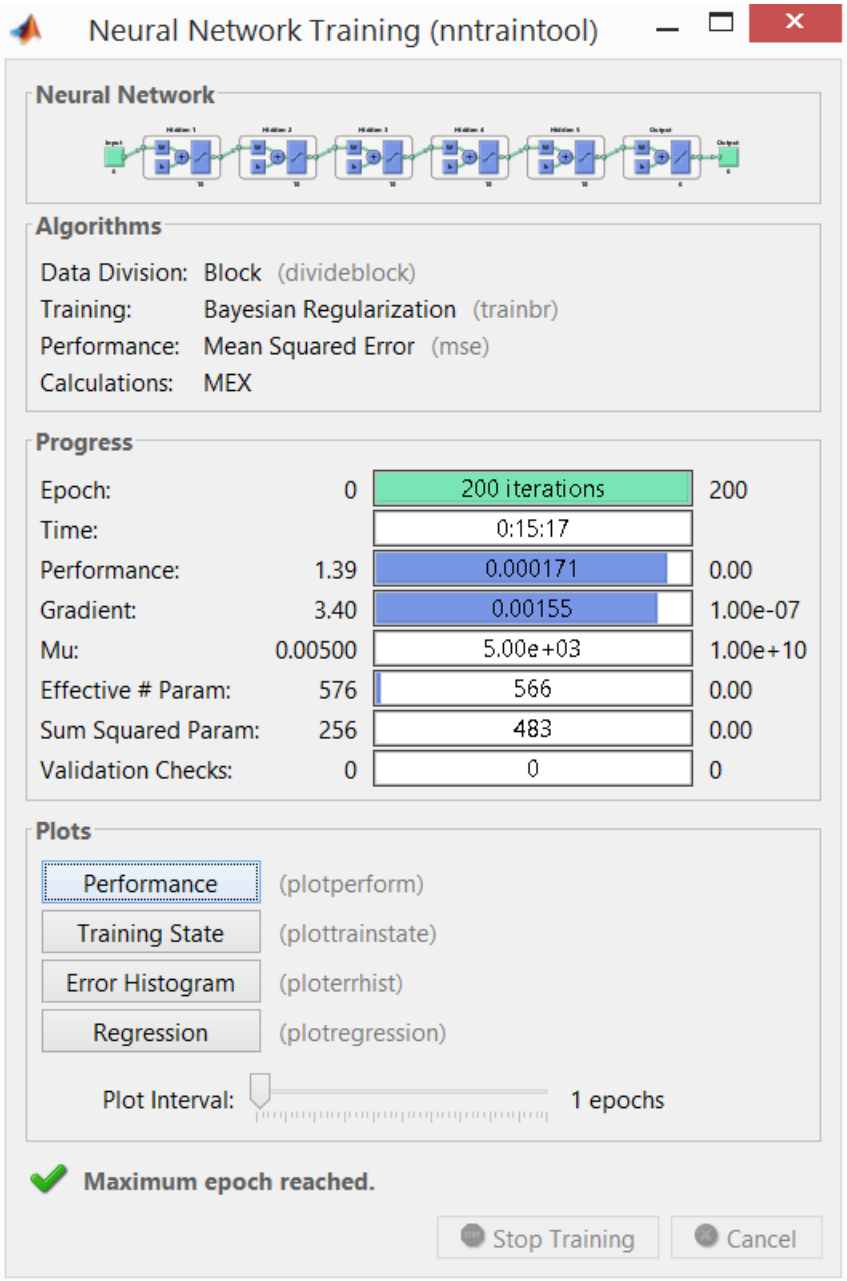

Figure 8.1: Neural Network parameters for class 3

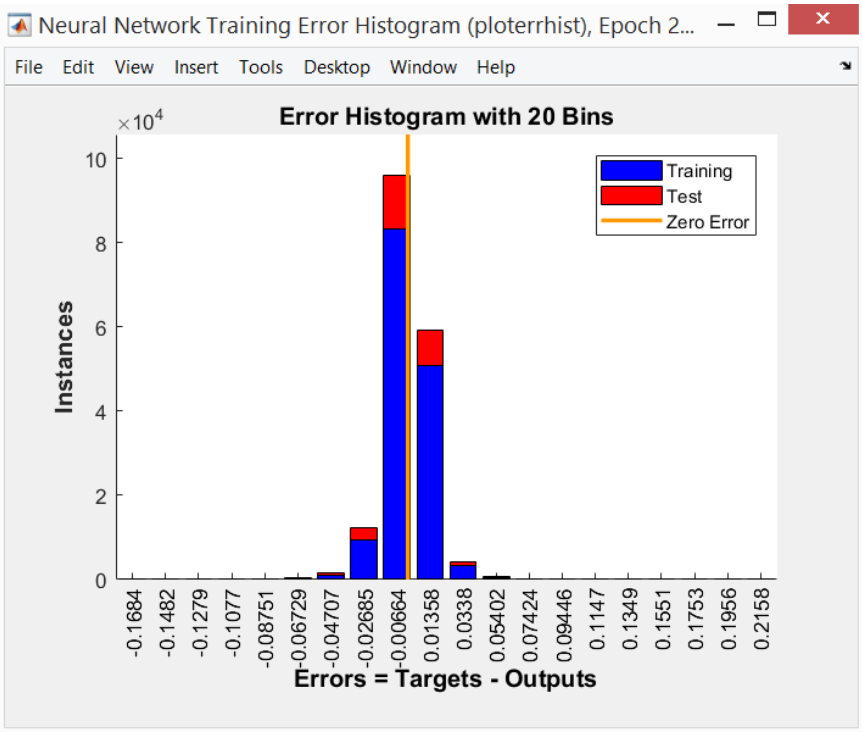

Figure 8.2: Error histogram for neural network 3

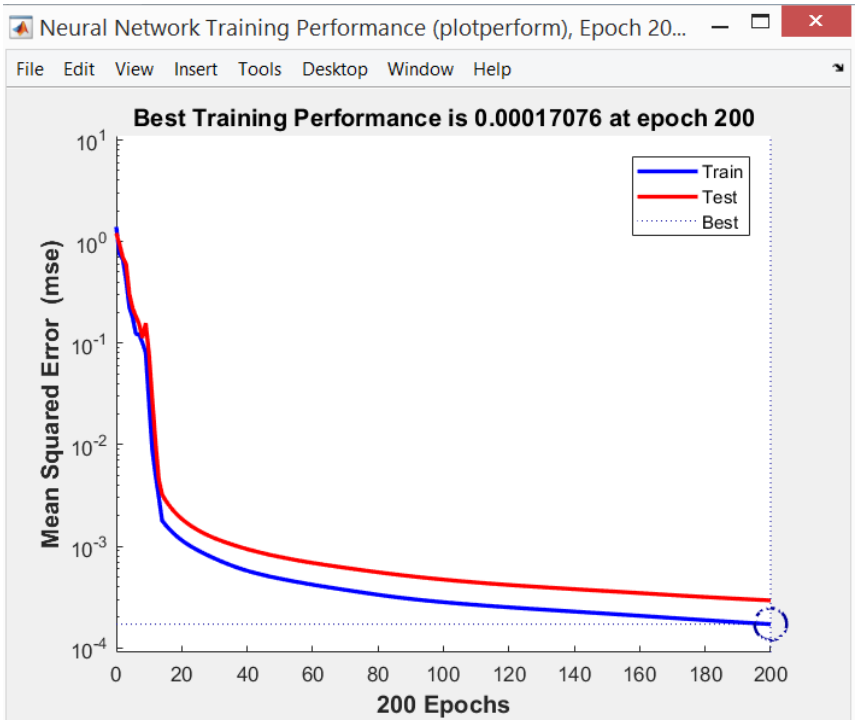

Figure 8.3: Performance of neural network for class 3

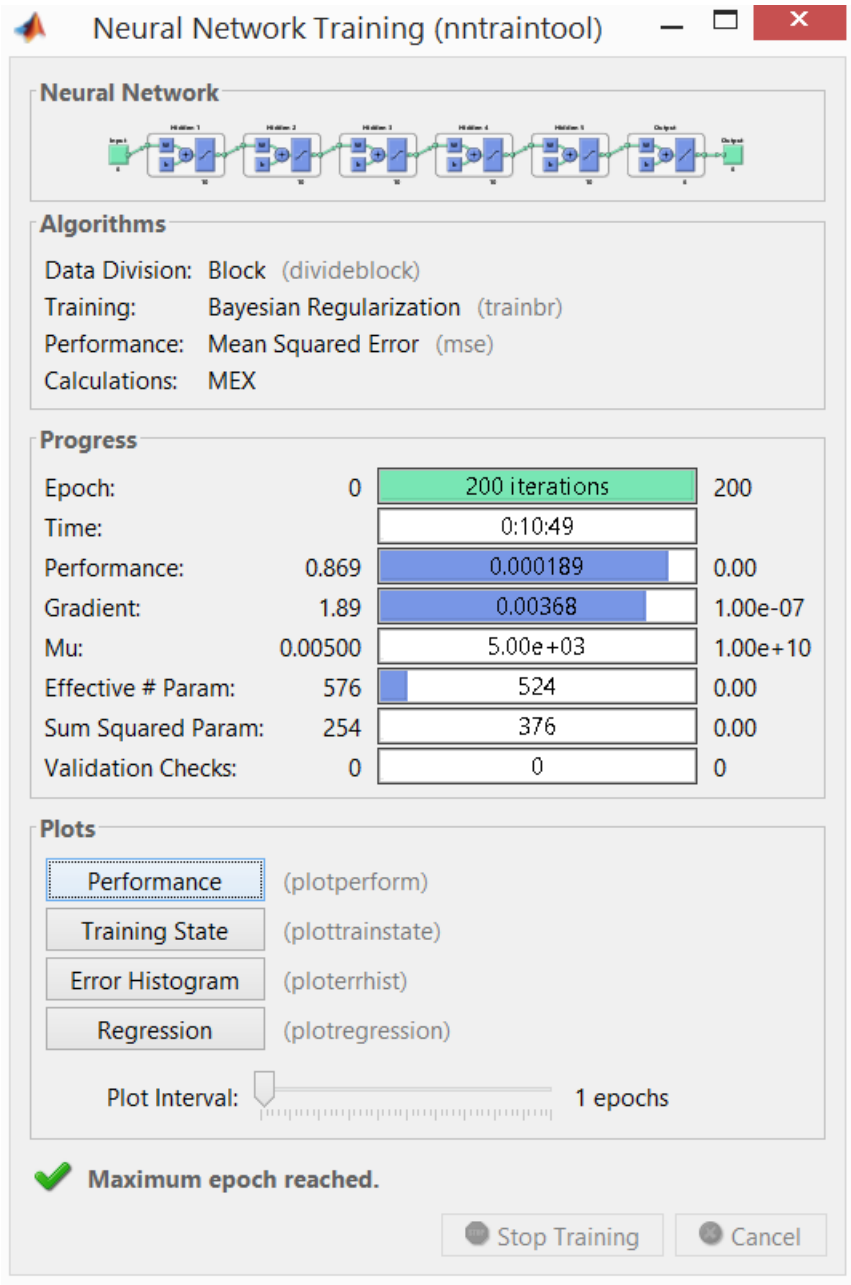

Figure 9.1: Neural Network parameters for class 4

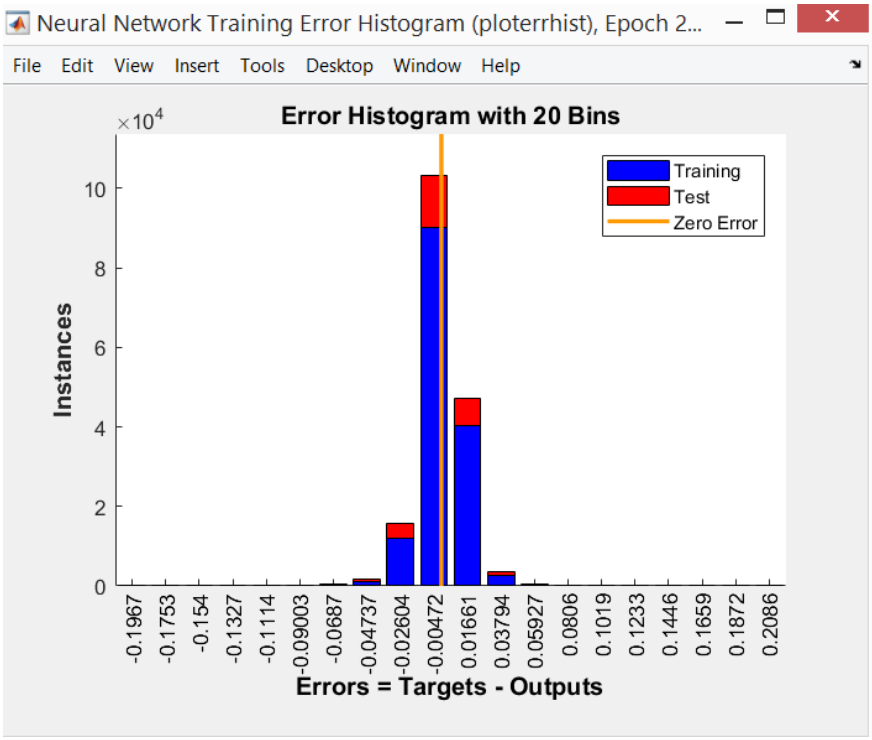

Figure 9.2: Error histogram for neural network 4

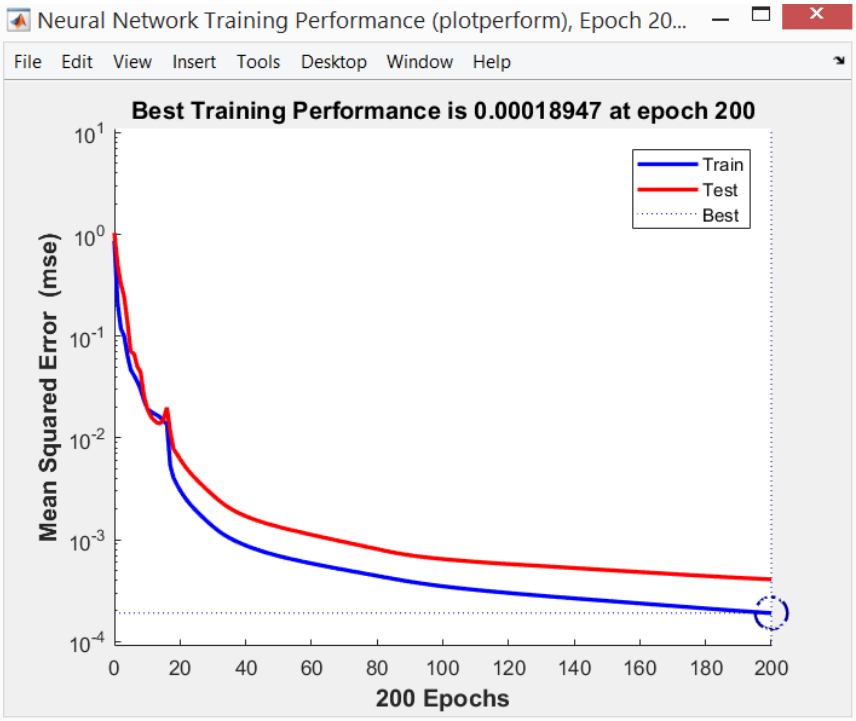

Figure 9.3: Performance of neural network for class 4

5. Conclusion

The neural network utilizing the Bayesian Regularization method was used to create a classifier to recognize (P, Θ) pairings when values of lengths are passed as inputs. The categorical accuracy of our model was seen to be at the order of 10^{-3} or better which turn out to be at max an error of 0.02 radians or 1.15 degrees in orientation and 0.1 meters in position.

An alternative approach to the forward kinematics solution of a general hexapod is introduced that is extremely fast and suitable for real-time applications. Unlike the analytical methods that are restricted to special types of platforms, the proposed approach is applicable to a general platform without placing any restrictions on the geometry or link connection points. Furthermore, it does not suffer from the problems associated with numerical methods of solving a set of nonlinear or polynomial equations.

Many related applications of such kind usually do not have both inverse and forward equations to work on. Such problems can be addressed by the proposed solutions as the memory demands are decent, and thus affordable. Desired accuracy can be obtained via changing parameters of the neural network such as the number of hidden layers, the number of neurons in each layer, activation method etc.

6. References

1. “Real Time Forward Kinematics Solutions of Hexapods”, Mahmoud Tarokh
2. “Kinematic Analysis of a Stewart platform manipulator”, Kai Liu, John M. Fitzgerald, and Frank L. Lewis
3. https://www.mathworks.com/
4. https://www.gnu.org/software/octave/doc/interpreter/
5. https://www.wikipedia.org/
6. https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html
7. https://www.google.com/
8. https://stackexchange.com/