Machine Learning Homework

Olmo Ceriotti

November-December 2024

Contents

Chapter 1

Introduction

1.1 Task

For this homework, the objective is to create three distinct machine learning models to compute the forward kinematics of three different robots. This involved designing and training each model to map the robot's joint parameters, such as angles and lengths, to the corresponding Cartesian coordinates of the end effector. Each robot presented unique challenges, such as variations in degrees of freedom and kinematic configurations. To tackle these, I employed a combination of neural networks, ensuring they could accurately learn the complex, nonlinear relationships inherent in the kinematics. The project required preprocessing the data, tuning hyperparameters, and validating the models to ensure robust performance across varying configurations. Further augmentation and focus on the hyperparameters characterised this work. The optional parts, regarding inverse kinematics, were explored yielding interesting partial results.

1.2 Tools

A variety of tools and frameworks were used to finish this work. Firstly, the simulation is ran using Mujoco and OpenAI Gym. Pandas and NumPy were used to firstly manipulated the datasets. To instantiate the model, TensorFlow was used in conjuction with Keras. Support libraries like SkLearn and SciPy were also employed. Hyperparameter tuning was done using the tuners provided from Keras and SkLearn. To monitor all the data, parameters and performance coming out of the training process, WandB was used. All the assignment was coded in Google Colab and ran on Colab Pro A100 GPUs with extended RAM.

Chapter 2

Dataset

2.1 Generation

To generate the dataset a couple adjustments needed to be made in order to ensure the correct functioning of the simulation. Firstly, I incurred in some issues with setting up the Docker environment, probably given by my Ubuntu installation. To fix these issues a Python Virtual Environment was set up to contain the needed packages. After installing the requirements, everything was ready to go. Then, for every robot in the simulation, 10000 data points were generated using my Student Number as seed. The "alternative" version of the dataset, generated in order to perform a test on a dataset created with a different seed, was assigned as seed the reverse of my Student Number.

2.2 Data manipulation

Not much manipulation was needed to prepare the data for the next steps. The main uncertainty came from the presence of the trigonometric values generated by the simulation. To solve this issue, models were trained both with and without those values. As it is discussed in the next sections, that the use of such values proved ineffective and not of much use to increase the accuracy.
A similar problem was faced with the value to predict. Each robot simulation gave as output the coordinates of the end effector, whether 2D or 3D, and the quaternion representing the orientation. A moderate amount of testing was performed to choose whether to predict both of these values but ultimately only the coordinates were used. This decision was motivated by the usefulness of the predicted data and by the need to ease the calculations for the Jacobian matrix, as seen later.
Ultimately, to test the robustness of the models, the data was enhanced by adding normal noise to the generated data, with results discussed in the next session.

Chapter 3

Forward Kinematics

3.1 Overview

3.1.1 General procedure

To carefully predict the forward kinematics of all the different robots the models chosen were all Artificial Neural Networks. The first approach to tackle the problem was mostly empirical. Through the instantiation of TensorFlow's neural network with arbitrarily chosen hyperparameters a first idea of the issue at hand was obtained.
Then, a manual search for better hyperparameters was done, through exploring different loss functions, optimizer and learning rates.
To better optimize this search for the best performing model, the tuner HyperBand was employed. The parameters to search for each model were the number of hidden layers, the number of units per layer, the optimizer to employ, the loss to compute and the learning rate. Other parameters such as the epoch number and the batch size were successfully searched manually without any big drawback in order to optimise efficiency. The values found through this search will be later discussed in each robot section. HyperBand implements a different approach than most classical hyperparameter tuning methods such as grid search or randomized search. Instead of completing all runs with any possible combinations of parameters, or randomly selecting some combination, it tries a few epochs, namely two, for each possible combination. Than it estimates which of these runs are the most promising and bring those to completion. This methods represent a good trade-off between the computational complexity of grid search and the inaccuracy of randomised search. After this work, in order to compute the Jacobian of the model and compare it with the analytical Jacobian manually computed, the provided FK_Jacobian function was employed and the difference was estimated through a custom loss function defined as follows:

$$\mathrm{MSE}_{\mathrm{jacobian}} = \frac{1}{n}\sum_{i=1}^{n}\left(J_{\mathrm{true},i} - J_{\mathrm{model},i}\right)^2$$

The Jacobian computation was the main factor in the decision of not using the trigonometric value in the training process. Since those values are in function of the joint angles and are not input of the forward kinematic it would have been complicated to compare the model Jacobian to the analytical one.

3.1.2 Other models

To carefully decide the best model to employ to solve this task other options were explored. More specifically, the choice fell on SVMs, Logistic Regression and Linear Regression. As it's easy to predict, Logistic and Linear regression yielded subpar results not worthy of consideration. The SVM, in particular SVR with rbf kernel, provided some interesting results, further explored in comparison to the main models' results. To search the best fitting model, an hyperparameter search was performed using Keras' RandomizedSearchCV tuner. Through establishing a parameter grid with C, epsilon and gamma values inside, the tuner sampled a set number of combinations to evaluate and find the best one.

3.2 R2

3.2.1 Model overview

The model for the R2 robot takes as input the joint angles and produces as output the x and y position of the end effector. To do so the output layer needs to predict two values at the same time. This behavior is implemented through the use of a Dense layer with two neurons and a linear activation function.
To achieve the wanted behavior, the first step was to employ a neural network with 5 hidden layers and 10 unit per layer. The tested optimizer that was used was mostly Adam while the losses function manually explored were mean absolute error and mean square error. For these trial runs the learning rate was set to 0.001.
The situation is definitely different when looking at the parameters outputted by the hyperparameter tuning. Two different tuning procedures were performed: one with the task of deciding the number of layers, learning rate, loss and optimizer and the other with the added responsibility of deciding the number of hidden units for each layer.
The first tuning process returned a model somewhat similar to the one manually instantiated, with 5 hidden layers, mean square error as loss, Adam as the optimizer, 80 hidden units per layer and 0.0025 as learning rate. This resulted in a model with a total of 26,324 parameters.
On the other hand, the tuner, when given the task of deciding how many units per single layer, produced a radically different output with 6 layers, Huber loss, RMSprop optimizer and a learning rate of 0.0006. Each layer had a different number of parameters, listed here in order: 64, 128, 256, 160, 32, 96.

3.2.2 Performance

General performances

The following table highlights the performances of different models in different situations. The best model architecture is composed by 5 hidden layers, 80 units per layer, Adam as optimizer, MSE loss, 0.00025 learning rate. The first model tried was then defined by 4 hidden layers, 10 units per layer, Adam as optimizer, MSE loss, 0.001 learning rate. The model tuned with HyperBand had a completely different structure as said previously with 64, 128, 256, 160, 32, 96] units per layer, RMSprop as optimizer, Huber loss, 0.0006 learning rate. The "different seed" test and the test on noisy data were both performed by the best model architecture found. The SVR was defined as previously stated.

The most important result coming from this table is that all model performed well on the task at hand. There was a sensible improvement between the first and the best model. As shown also in the following sections, the HyperBand tuning didn't yield any particularly performing result.

Trial	Loss	test MAE	training MAE	Elapsed time
Best model	**0.000001**	**0.0009**	**0.0010**	2 min 8 s
First model	0.00001	0.0026	0.0024	1 min 36 s
Tuned model	0.000002	0.0013	N/A	12 min 8 s
Different seed	0.000001	0.0010	0.0010	2 min 25 s
Noisy data	0.000001	0.0031	0.0026	1 min 40 s
SVR	N/A	0.0057	N/A	N/A

Table 3.1: General performances on R2 dataset for different models and different conditions

The test performed on a dataset created with a different seed demonstrated the robustness of the model while the test on noisy data, as expected, yielded fairly worse results. The SVR performed poorly, indicating the ANN as the best suited model for this task.

Hyperparameter study

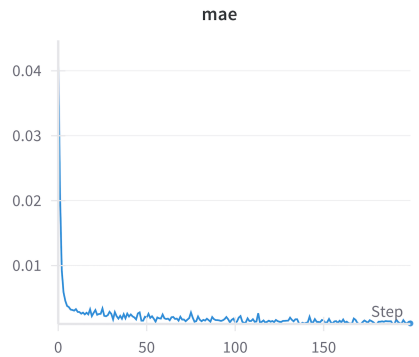
Each model was built on the same architecture of the best model found during the previous training. The models were then trained alternating a combination of Adam and RMSprop as optimizers and MAE and MSE as losses.

Opt/Loss	MAE	MSE
Adam	0.0022	**0.0009**
RMSprop	0.0035	0.0014

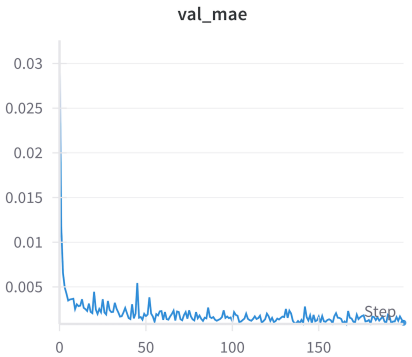
Table 3.2: Test MAE for 4 models trained with a combination of two hyperparameters on the r2 dataset.

The results highlight the superiority of Adam in finding a minimum point for the function at hand and indicate MSE as a better loss function.

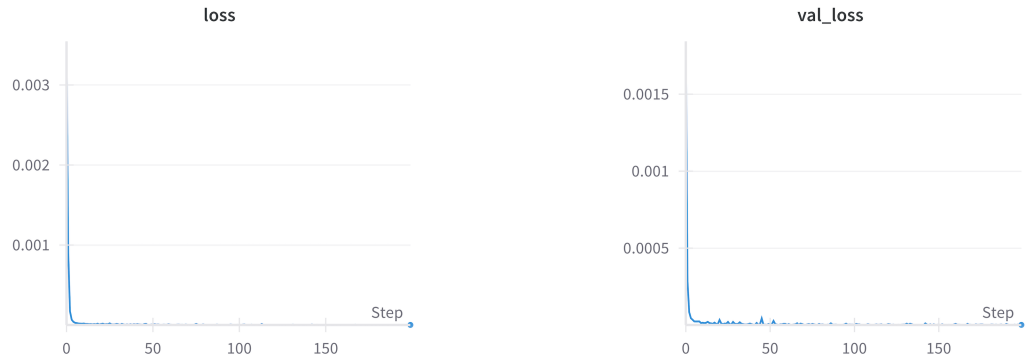
Best model deep dive



(a) Mean Absolute Error (MAE)



(b) Validation MAE (Val_AME)



(a) Loss

(b) Validation Loss (Val_Loss)

Figure 3.2: R2's best model performance Metrics Visualization

Fig. 3.1 highlights the fast convergence of the best model on this task, exposing the efficiency of Adam as an optimizer. These graphs also show that the choice of 200 epochs was probably an exaggeration but the absence of overfitting on the training data demonstrates how the architecture chosen was optimal.

3.2.3 Jacobian comparison

The analytical Jacobian was computed as follows:

$$\mathbf{J} = \begin{bmatrix} -l_1 \sin(\theta_0) - l_2 \sin(\theta_0 + \theta_1) & -l_2 \sin(\theta_0 + \theta_1) \\ l_1 \cos(\theta_0) + l_2 \cos(\theta_0 + \theta_1) & l_2 \cos(\theta_0 + \theta_1) \end{bmatrix}$$

The comparison was ran on the best performing model. The difference between the two Jacobian computed on a random data entry was very small, with a value of 0.000068693.

3.3 R3

3.3.1 Model overview

A similar approach to the one presented for the R2 robot was employed for the R3 robot. The R3 robots takes as input the angles of the three joint $J0$, $J1$ and $J2$ and produces as output the X and Y positions of the end effector.
As it was done with the R2 robot, the first architecture for the neural network was very simple: 5 hidden layers with 10 units each, Adam as the optimizer with a learning rate 0.001 and mean square error loss. Since this model was already overly simple for the R2 robot, a different approach needed to be taken. In order to save computational resources, the same architecture found by the first tuner ran on the previous dataset was tested on the R3 robot. This resulted in a final model with 26,404 parameters.
After testing this configuration, Hyperband tuning with power of deciding the units per layer was run, creating once again a different model. This time the results were the following: a model with 6 layers, RMSprop as optimizer with 0.00083 as learning rate and LogCosH as loss. The number of hidden units for each layer were: 128, 256, 128, 32, 96 224. The total number of parameter after this changes was 95,908.

3.3.2 Performance

General perfomances

As in the previous section, this table highlights the performances of different models in different situations. The best model architecture is composed by 5 hidden layers, 80 units per layer, Adam as optimiser, MSE loss, 0.00025 learning rate, similar to the r2 model. The first model was also very similar to its r2 counterpart as it was defined by 4 hidden layers, 10 units per layer, Adam as optimiser, MSE loss, 0.001 learning rate. The model tuned with HyperBand had a significantly different structure than its manually created counterparts with [128, 256, 128, 32, 96, 224] units per layer, RMSprop as optimiser, LogCosh loss, 0.0008 learning rate. As before, the "different seed" test and the test on noisy data were both performed by the best model architecture found. The results from this tests are easily interpretable. The best model had

Trial	**Loss**	**test MAE**	**training MAE**	**Elapsed time**
Best model	**0.00001**	**0.0019**	**0.0020**	**2 min 22 s**
First model	0.00001	0.0028	0.0030	1 min 46 s
Tuned model	0.000005	0.0023	N/A	10 min 19 s
Different seed	0.000007	0.0020	0.0020	2 min 19 s
Noisy data	0.00003	0.0040	0.0039	1 min 57 s
SVR	N/A	0.0051	N/A	N/A

Table 3.3: General performances on R3 dataset for different models and different conditions

slightly worse performances than the r2 model, but the change wasn't significant. The process of training different models showed a clear improvement. Still, the HyperBand tuning method didn't yield a particularly interesting result. Noisy data still characterized a worse performance and the SVR didn't deliver any good result.

Hyperparameter study

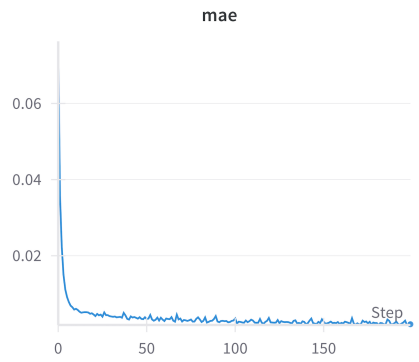
The process was identical as the one used for the r2 models. As seen before Adam delivered the best results, but the difference was less accentuated. MAE delivered worse optimization overall.

Opt/Loss	**MAE**	**MSE**
Adam	0.0028	**0.0019**
RMSprop	0.0056	0.0020

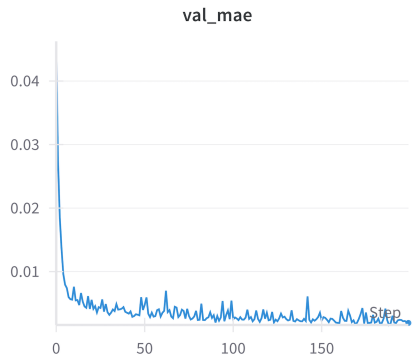
Table 3.4: Test MAE for 4 models trained with a combination of two hyperparameters on the R3 dataset.

Best model deep dive

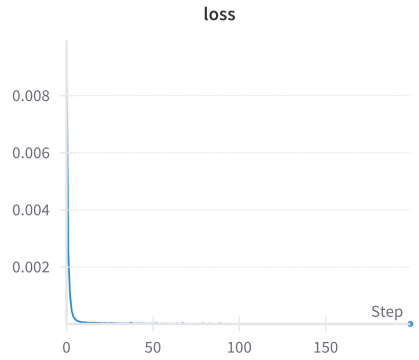
The diagrams of the training process of the best model show that it converged slightly slower than the model trained on the previous dataset. Still, it's possible to see that even if it converged way before the end of the training, no overfitting was present.



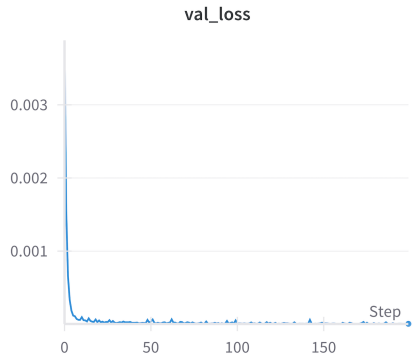
(a) Mean Absolute Error (MAE)



(b) Validation MAE (Val_AME)



(c) Loss



(d) Validation Loss (Val_Loss)

Figure 3.3: R3's best model performance Metrics Visualization

3.3.3 Jacobian comparison

The Jacobian was computed as follows:

$$\begin{bmatrix} -l_1\sin(\theta_0) - l_2\sin(\theta_0+\theta_1) - l_3\sin(\theta_0+\theta_1+\theta_2) & -l_2\sin(\theta_0+\theta_1) - l_3\sin(\theta_0+\theta_1+\theta_2) & -l_3\sin(\theta_0+\theta_1+\theta_2) \\ l_1\cos(\theta_0) + l_2\cos(\theta_0+\theta_1) + l_3\cos(\theta_0+\theta_1+\theta_2) & l_2\cos(\theta_0+\theta_1) + l_3\cos(\theta_0+\theta_1+\theta_2) & l_3\cos(\theta_0+\theta_1+\theta_2) \end{bmatrix}$$

. Once the best model was chosen, the difference between the two Jacobian was still significantly low: 0.00034664.

3.4 R5

3.4.1 Model overview

The procedure was practically identical to the one of the other modules. Even if the task was significantly more complex, the previous chosen way of action proved valid and nice improvements, as presented later, were seen.

3.4.2 Performance

The models chosen for the r5 dataset varied more than in the previous two datasets. The first model trained was composed by only 4 hidden layers, 10 units per layer, Adam as optimizer, LogCosh loss and 0.001 as learning rate. The best model on the other hand has more parameters, with 5 hidden layers, 10 units per layer, Adam as optimizer, MSE loss and 0.001 as learning rate. The tuned model was, again, completely different, with [224, 224, 224, 192, 256, 224, 192, 64, 192, 192] units per layer, Adam as optimizer, LogCosh loss and 0.0002 as learning rate.

General performances

Trial	**Loss**	**test MAE**	**training MAE**	**Elapsed Time**
Best model	0.00001	0.0035	0.0029	3 min 29 s
First model	0.0022	0.036	0.036	4 min 7 s
Tuned model	0.00003	0.0053	N/A	15 min 24 s
Different seed	0.00001	0.0050	0.0029	5 min 15 s
Noisy data	0.00002	0.0066	0.0045	3 min 36 s
SVR	N/A	0.0045	N/A	N/A

Table 3.5: General performances on R5 dataset for different models and different conditions

The results were far than surprising. The model performed slightly than the previous ones, most probably due to the fact that the task were significantly more difficult. The difference between the first model and the best is very marked. One of the most interesting insight is given by Hyperaband's tuned model performance, significantly worse than the hand tuned ones. The test on different seed also was worse than on the previous dataset, probably as a result of the added dimension in this dataset. Surprisingly, the SVR performed better than the tuned model.

Hyperparameter study

The process was fairly similar to the one used on R2 and R3 but this time the losses explored were LogCosh and MSE.

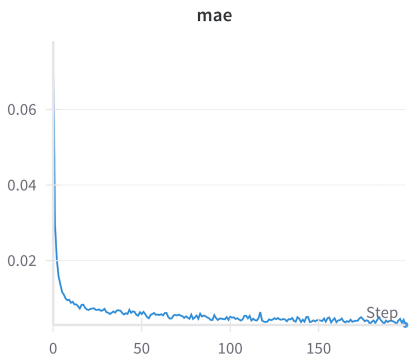
Opt/Loss	**LogCosh**	**MSE**
Adam	0.0035	0.0061
RMSprop	**0.0031**	0.0031

Table 3.6: Test MAE for 4 models trained with a combination of two hyperparameters on the r5 dataset

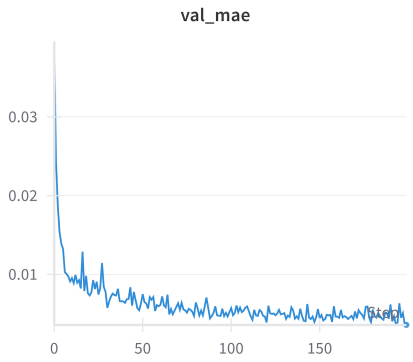
This time, Adam and MSE performed way worse than the alternative, with LogCosh and RMSprop proving to be the best combination for this specific task.

Best model deep dive

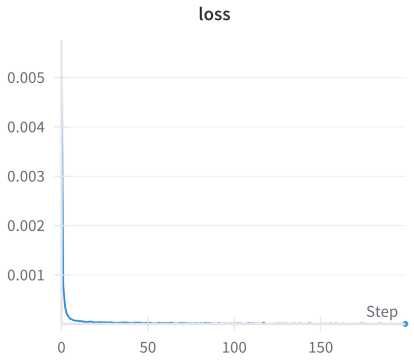
As shown in Fig. 3.3, for this task, the convergence was longer, highlighting the difficulty in learning such a difficult kinematic function. This time the model improved for the whole 200 epochs reaching a point were it performed fairly good.



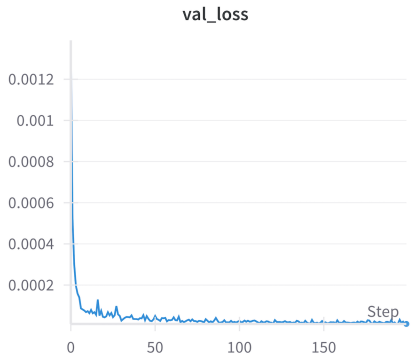
(a) Mean Absolute Error (MAE)



(b) Validation MAE (Val_AME)



(c) Loss



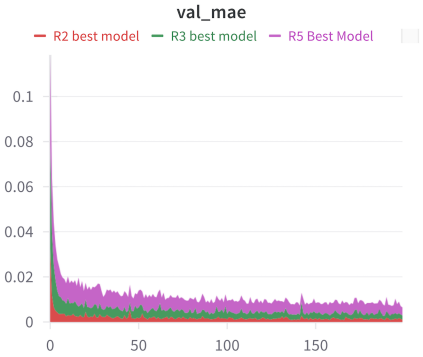
(d) Validation Loss (Val_Loss)

Figure 3.4: R5's best model performance Metrics Visualization

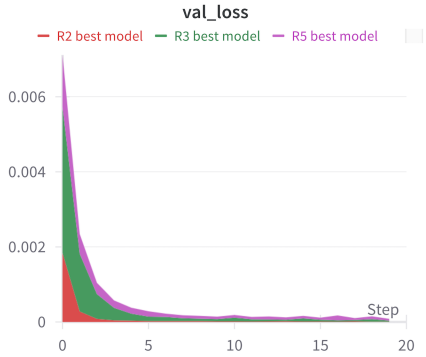
3.4.3 Jacobian comparison

The Jacobian comparison was the most different part in respect to the other models. I incurred in some issues in calculating the DH parameters for this robot, needed to compute the forward kinematic function and then its jacobian. I obtained an analytical Jacobian but I can say with almost certainty that it's not correct. I still computed the difference and, as expected, it was higher then the others, with a value of 0.029926.

3.5 Conclusion



(a) Comparison between the 3 best model validation MAE values during training



(b) Comparison between the 3 best model validation loss values during training

Overall, this analysis provided some interesting results. As it's possible to say from Fig. 3.4, the three models had increasingly inaccurate performance, surely given by the increasing difficulties of each task. The final mean absolute was still low, signifying that the models were doing a good job. Another important insight was the number of epochs each model took to converge, that can be seen from Fig. 3.5. As seen in the error, the complex the model, the longer it took to converge. All the models still reached the minimum loss in these runs.
To further expand this work many possible paths are available. Firstly, the suggested robot control implementation could be done but, given my limited knowledge on the matter, I was not able to complete it. Then, different robots could be used as subjects and transfer learning could be used to have a base for the next models.

Chapter 4

Inverse Kinematics

4.1 General procedure

For this task, the goal was to compute the angles of the joints of the robot given the final position of the end effector. To do so, the computed method proceeded as follows. Firstly, an uneducated guess on the angles of the joints was taken. The initial angles were 0.5 or 0.5 chosen randomly the first time. Then, the following procedure was tried for a given number of time. If the method didn't converge during these iterations, the process would just stop. To reach the target angles, the error between the position to reach and the position of the end effector given by the guessed angles was computed. To improve the guess the Levenberg-Marquardt method was used. Using the Jacobian and a damping factor, a delta was computed and applied to theta. The method continues until the last possible iteration or until the error is smaller than the tolerance. When this happens, the method converged. The metric to evaluate the success of this method is the MAE between the given position and the position indicated by the angles. During the process of establishing this method the difference between the angles found and the "right" angles was also used but proved ineffective since to reach the same end point that may be more than one solution.

4.2 Results

Model	**Position MAE**	**Iterations**
R2	0.000005	38
R3	0.000007	24
R5	0.000004	16

Table 4.1: Performance of the IK algorithm on the three robots

As evident from Table 4.1, the error was very small in all the iterations and the convergence was fast. Not all the solutions returned the "correct" angle but provided the alternative position instead.