Intelligent Adaptive Systems: Assignment

Chanelle Lee

April 21, 2016

1 Introduction

1.1 Adaptive Neuro-Fuzzy Inference System (ANFIS)

ANFIS is a 'fuzzy inference system implemented in the framework of adaptive networks' developed in the early 1990s [1]. It is called a hybrid neuro-fuzzy technique as it uses the learning capabilities of neural networks to 'tune' the membership functions of a Sugeno-type Fuzzy Inference System.

1.2 The Task

The task of this assignment is to use ANFIS to derive a representation of the inverse kinematics of the Lynxmotion robotic arm. In order to investigate this, Matlab's anfis toolbox will be used, with the main questions to be explored:

1. What is the ideal density of training data? A balance between efficiency and effectiveness is needed; as the more training data points the more accurate the results, but the longer the system takes.
2. How to generate the initial fuzzy inference system; should genfis1 or genfis2 be used? And what parameters should they take?
3. For how many epochs should ANFIS run?
4. What should be the spread of the training data? Should the training data points be uniform over the workspace or concentrated near singularities?

Due to time constraints, the simpler problem of a planar RR arm will be used to investigate the first three questions; as the data sets required to train the system are much smaller, and so more exploration of parameter settings of the ANFIS is achievable. Then, the best parameters found will be used to train for a planar RRR arm and as the planar RRR arm gives a representation of the Lynxmotion arm in the XZ-plane, the final question will be investigated. Finally, these findings will be implemented on the Lynxmotion arm and compared against analytical solutions.

2 Planar RR Arm

For this section, the Lynxmotion arm will be reduced the to the planar RR problem as described in Fig.1[1]. The input-output data pairs used to train and

[1] Altered to match findings in Section2.1

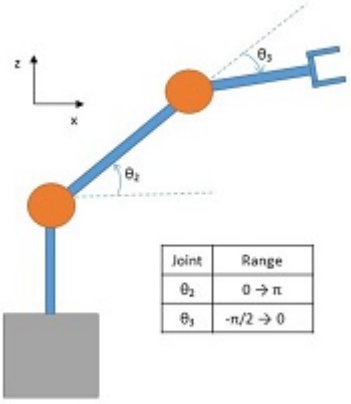

Figure 1: Planar RR Robotic Arm

θ_3 range	trnRMSE2	chkRMSE2	trnRMSE3	chkRMSE3	cartRMSE	cartMin	cartMax
$[-\pi/2, \pi/2]$	0.3792	0.3794	0.7387	0.7389	35.7034	0.0497	99.4345
$[-\pi/2, 0]$	0.0312	0.0311	0.0682	0.0687	5.5590	0.0651	111.7742

Table 1: Results from changing range of θ_3.

validate will be the joint angles θ_2 and θ_3 and their corresponding values in values in the XZ-plane.

2.1 Joint Angle Range Issues

When the experiments were first run there were very large errors in the Cartesian and joint space (Table.1)[2] and this was discovered to be because the range θ_3 was set to $[-\pi/2, \pi/2]$, This meant that for each point in Cartesian space there were two possible solutions in the joint space with either a positive or negative value for θ_3. This corresponded to the fact that each point in Cartesian space can be reached by the end effector in one of two configurations; elbow up or elbow down. This was solved by limiting the range of θ_3 to only negative values, i.e. limiting the robotic arm to only elbow down configurations, which is much more realistic in terms of the Lynxmotion arm's capabilities.

2.2 Number of Epochs

The number of epochs controls the length of time ANFIS spends tuning the membership functions of the Fuzzy Inference System (FIS) and setting this is a matter of comparing the accuracy attained against the time taken. In Fig.2[3] it is clear that if only the first fifty epochs are considered the training and validation errors of the joint angles are still falling and there is little evidence to show how far they might further fall, but when one thousand epochs are run there is clearly little improvement after five hundred epochs. Table.2 also shows

[2]Genfis1 with two membership functions, 100 data points and 500 epochs.
[3]Genfis1 with two membership functions and 100 data points

dataPoints	Epochs	MFs	trnRMSE2	chkRMSE2	trnRMSE3	chkRMSE3	cartRMSE	cartMin	cartMax	time
100	50	2	0.0848	0.0847	0.1474	0.1477	8.9448	0.0728	63.0842	2.38
100	100	2	0.0358	0.0356	0.0795	0.0801	5.8968	0.0425	25.2959	4.38
100	500	2	0.0312	0.0311	0.0682	0.0687	5.5590	0.0651	111.7742	20.17
100	1000	2	0.0311	0.0309	0.0681	0.0687	5.5723	0.0184	111.7176	39.92

Table 2: Number of Epochs Experiment

that the Root-Mean-Squared-Error (RMSE) in the Cartesian space reduces by less than a tenth of a millimetre from five hundred to one thousand epochs despite it taking twice as long to run. Interestingly, there is also little change in the Cartesian RMSE between one hundred epochs and five hundred epochs. In conclusion, the best number of epochs to use for further experiments is five hundred as confidence can be had that training and validation errors in the joint angles have stabilised by this epoch, but there is little point continuing as neither the error in the joint angles or the Cartesian RMSE appear to decrease enough to be beneficial.

2.3 Optimising Genfis1

To investigate the best parameters using Genfis1 to create the FIS two loops were used to explore:

- The number of membership functions between two and ten.
- The number of data points from ten, twenty, fifty, one hundred, two hundred and five hundred.

The results from this can be seen in Appendix 1, Table 5; the last half of the two hundred data

2.3.1 Number of Membership Functions

The number of membership functions is used by genfis1 to set the number of membership functions of each variable in the FIS [2]. This is important to explore as more membership functions can lead to an increase in accuracy due to a greater capacity for complexity, much like increasing the degree of a polynomial line of best fit. However, too great a capacity for complexity can lead to over-fitting, where the system has been trained so well on the training data, that it is unable to generalise in the areas in between. To continue the polynomial line of best fit example, this would be analogous to increasing the degree of the line of best fit until it passes through every given data point, but in doing so losing the overall shape of the data being described. In order to check for over-fitting, validation data is used so that the error at points not in the training set can be monitored. Then over-fitting can be identified as where the error at training points drops to near zero, but the error at validation points begins to climb, see Figure.3. A countermeasure against over-fitting is to increase the density of the training data, as seen in Figure.4, where the behaviour with different numbers of membership functions varies greatly with the density of the training data.

Another repercussion of increasing membership functions is that this also increases the time the systems takes to train and so once again a balance needs to be struck between accuracy and efficiency. Considering the results from Table

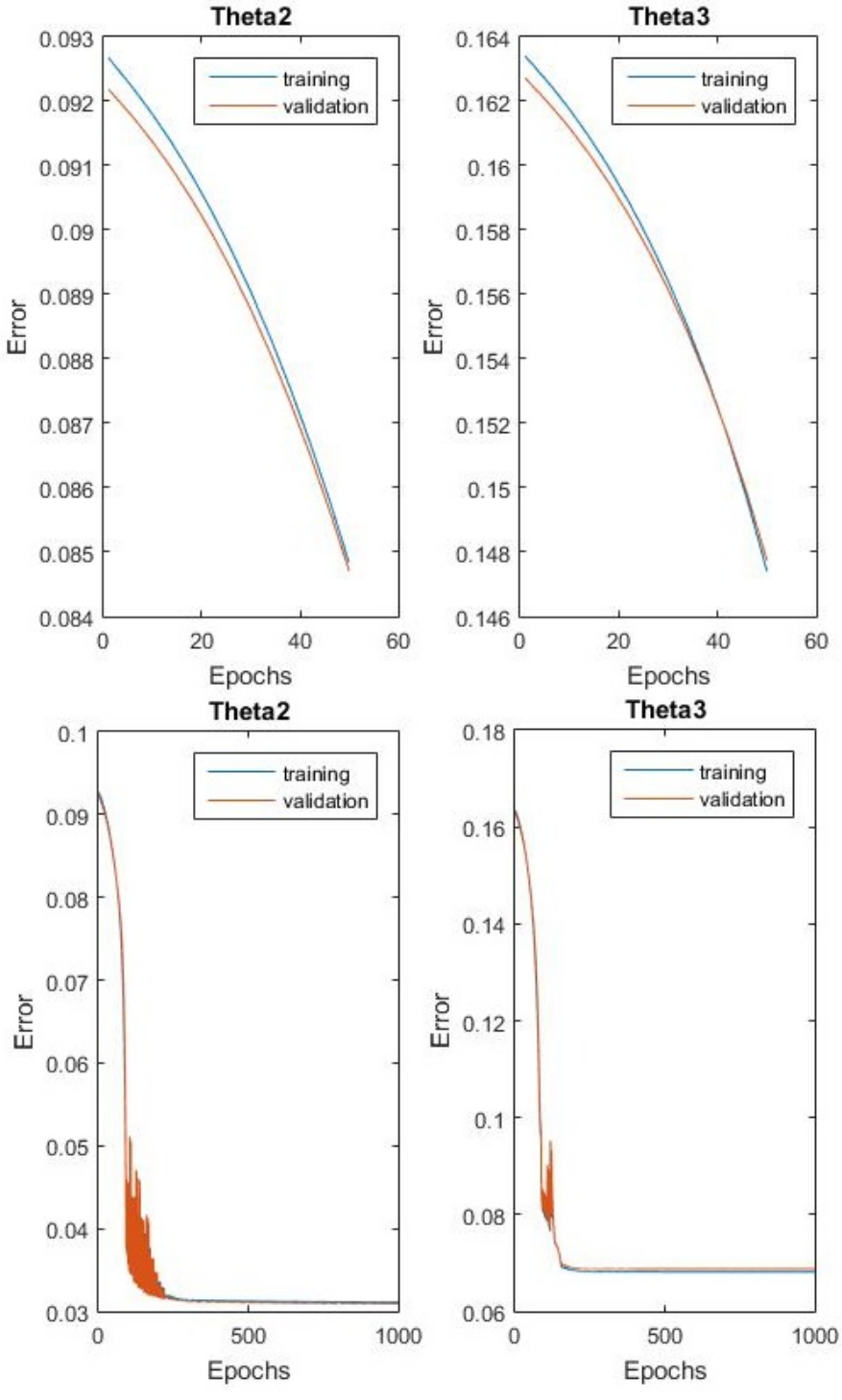

Figure 2: Planar RR Arm with run for 50 and 1000 Epochs

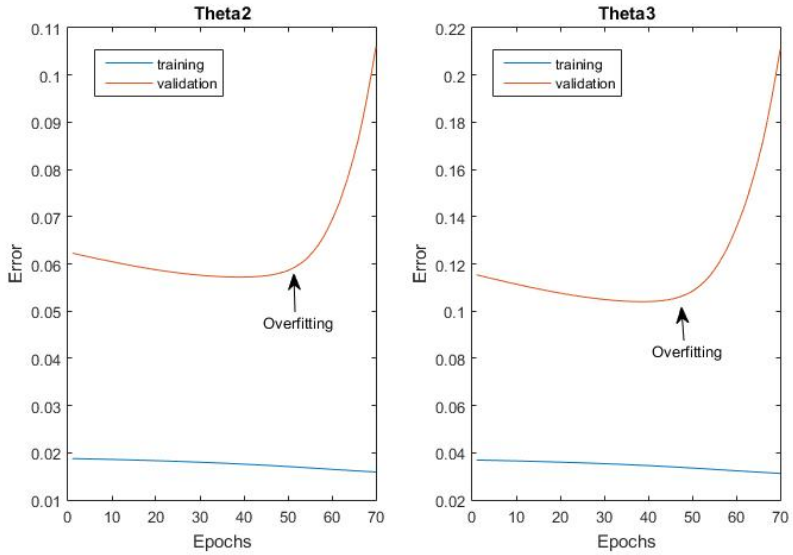

Figure 3: Example of Over-Fitting

5 and displayed in Figure.4, it is clear that the most choosing the most efficient number of membership functions depends greatly on the density of the training data. Perhaps surprisingly, Table 3 shows that, for the training data densities experimented with, two membership functions performed the best in terms of Cartesian RMSE and time. This is mostly likely due to the negative effects of over-fitting at low training data densities.

2.3.2 Density of Training Data

A limitation of ANFIS is that it uses supervised learning and as such needs training input-output data, which can be difficult to source for problems and the additional need for validation data only adds to this. Even with a problem

MFs	Average cartRMSE	Average time
2	15.2	5.9
3	25.8	14.0
4	46.2	31.0
5	65.7	65.5
6	37.7	125.7
7	64.1	227.6
8	52.3	380.2
9	67.0	603.8
10	51.1	904.0

Table 3: The average Cartesian RMSE and time taken for each number of membership functions from Table 5

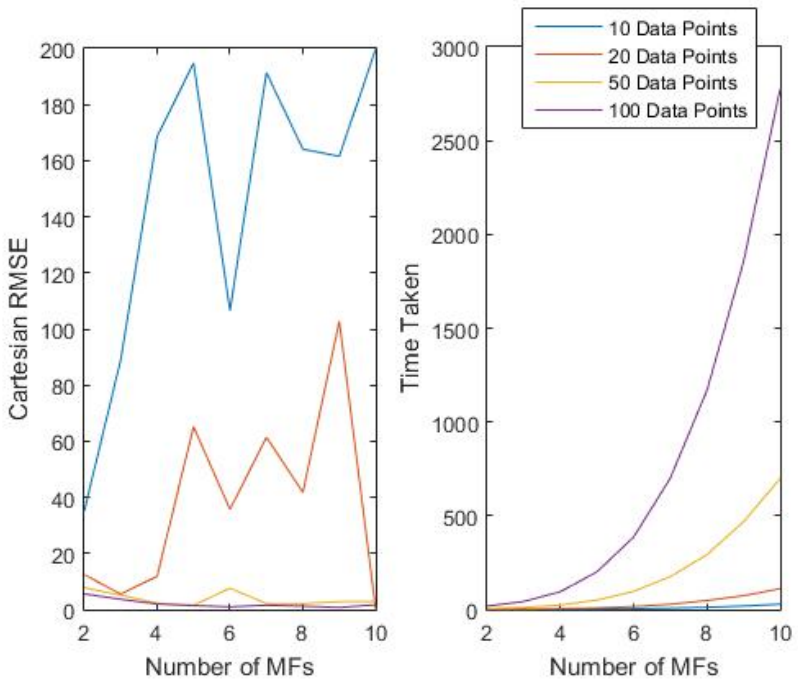

Figure 4: Time and Cartesian RMSE vs Number of Membership Functions for Different Numbers of Data points (from Table 5)

such as this task, where finding input-output pairs is relatively easy using the forward kinematic equations, the density of training data is still an issue as it requires more time and computational power to train. On the other hand, the greater the density of the training data, the greater the accuracy due to less speculation needed to fill in the missing areas of information.

As the planar RR problem is being used as a starting point for the planar RRR one, scalability also have to be considered. ANFIS suffers from the curse of dimensionality, in that adding even just one more variable, or dimension, to the problem can cause massive increase in the computing power and time needed to train the FIS. Considering all this and Figure.4 having fifty data points appears to be a good trade off between accuracy and time.

2.4 Optimising Genfis2

Genfis2 uses subtractive clustering to generate a Sugeno-type FIS structure [3] and in this experiment a genetic algorithm is used to try and determine optimal radii inputs for each joint. A genetic algorithm was chosen here, over any other type of optimisation algorithm, as it was feared that there would be many local optima, e.g. the optimal radius for each cluster. For the sake of convenience, Matlab's built in ga function was used for this experiment; with hard constraints set on the chromosome. However, as genfis2 only accepts radii to one decimal place and the population the ga produces is to four decimal places, the inputs are rounded before genfis2. As the aim of the genetic algorithm is to find the

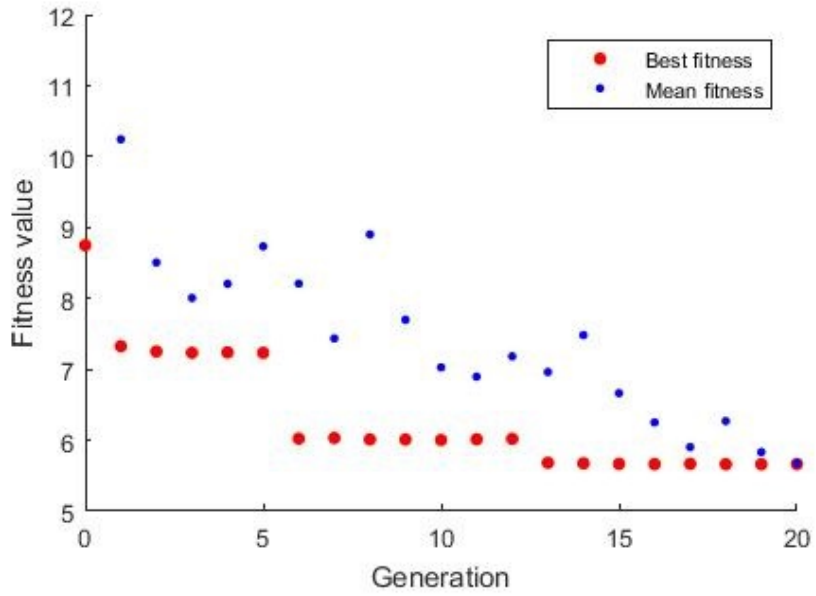

Figure 5: Genetic Algorithm Results for Genfis2 on Planar RR Arm

most efficient set of parameters, as opposed to the most accurate, the original fitness function used to evaluate the performance of the chromosomes is the Cartesian RMSE mitigated by the time taken. Figure.5 displays the results[4] from the genetic algorithm and the best chromosome found corresponded to,

$$\theta_2 \text{ radii } = [0.5, 0.8, 0.2] \quad \& \quad \theta_3 \text{ radii } = [0.4, 0.7, 0.5],$$

which has a Cartesian RMSE of 3.35 and took 980 seconds.

Time problems were experienced using the genetic algorithm, as the time taken for each generation is the length of time for fitness function (in this case genfis2 and anfis) to run, multiplied by the population size. As such only a small population size of ten was used for this experiment and as such the genetic algorithm ended after a short number of generations on a result which may not be globally optimal. In terms of practicality, the length of time necessary to compute the fitness function should in future be taken into account when considering the use of a genetic algorithm. Furthermore, the need to round the chromosomes from Matlab's built-in ga function, meant there was a lot of unnecessary variation within the population.

2.5 Concluding Remarks

Although genfis1 performed much better for accuracy and time for the planar RR arm, [4] states that, because genfis1 relies upon grid partitioning it is more

[4]With fifty data points and five hundred epochs.

susceptible to 'the curse of dimensionality'. This suggests that genfis1 would struggle with the planar RRR arm and even more so with the full Lynxmotion Arm. Also, as discussed in Section 2.4, the results of the genetic algorithm were not ideal and so genfis2 could perform much better than witnessed. Thus, genfis2 is the method to be carried forwards to the planar RRR problem.

3 Planar RRR Arm

This section will deal with the planar RRR arm problem as described in Figure.6. As there is now an additional joint angle to find, a new input variable is needed and this will be the pitch of the arm.

3.1 Experiments with Radii Values

In order to determine suitable radii values for constructing the FIS structure for θ_4, and whether to use the radii values found for θ_2 and θ_3 in Section 2.4 the following options were tested:

1. A control where the radii values were all set to 0.5.
2. A secondary control where the radii values for θ_2 and θ_3 are kept, and for θ_4 are all set to 0.5.
3. Keeping the radii values for θ_2 and θ_3 the same and setting θ_4 as a mix of the two.
4. Keeping the radii values for θ_2, but setting θ_4 to the previous values of θ_3 and θ_3 as a mix of the two. This was based on the thinking that as θ_3 was in fact the new angle to learn, as the planar RR arm had no joint attached to other joints at both ends.

The results of these experiments can be seen in Table 4 with the radius for the pitch set to the default of 0.5. The best performance was from the third set of radii values, and this is most likely due to the significant decrease in the validation RMSE in θ_4. Although surprisingly this was not the case for the reduction in validation RMSE for θ_3 for the fourth set of radii values.

3.2 Spread of Training Data

So far the data points for training and validation have been sampled uniformly across the ranges of the joint angles; however, if the sampling of the data points

Situation	cartRMSE	time	chkRMSE2	chkRMSE3	chkRMSE4
1	24.09	2776	0.0731	0.0661	0.0353
2	13.00	1144	0.0451	0.0781	0.0353
3	12.65	2504	0.0442	0.0750	0.0283
4	13.02	2391	0.0442	0.0603	0.0416

Table 4: Results for Planar RRR Arm in the Four Situations Specified

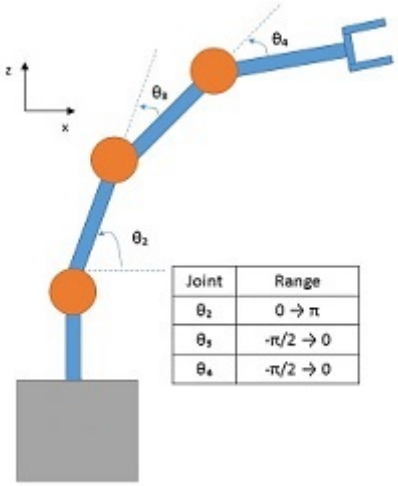

Figure 6: Planar RRR Robotic Arm

was concentrated more heavily in the areas producing the most errors, the accuracy could be increased. The first step is to identify the regions in the joint space causing the greatest errors in the Cartesian space. From Figure.7, these are identified as;

- $\theta_2 = \pi$,
- $\theta_3 = 0$, $\theta_3 = \frac{-pi}{2}$, and,
- $\theta_4 = 0$.

Unfortunately, as can be seen in Figure.8 the accuracy did not improve and did in fact worsen; in the regions with the concentrated sampling the errors do improve, but in the sparser regions the errors grew. Although disappointing, this is believed to be due the already sparseness of the data points and essentially there not being enough to favour any region and it not seriously hamper the effectiveness in another.

3.3 Changing Joint Angle Ranges

Another method to reduce the Cartesian RMSE is to either remove the singularities identified in Section 3.2 or possibly, where they present a boundary to the region, extend the region beyond them in the hopes that performing training either side will reduce the overall error. Figure.9 shows the results of cropping the singularities from the joint space, but, although this method achieved a much better Cartesian RMSE of 4.38mm, it has not truly solved the issue of these troublesome regions, just ignored them. It is a worthy method to consider though, as if the robot arm only needs a limited workspace, then training across singularities when it is unnecessary is pointless.

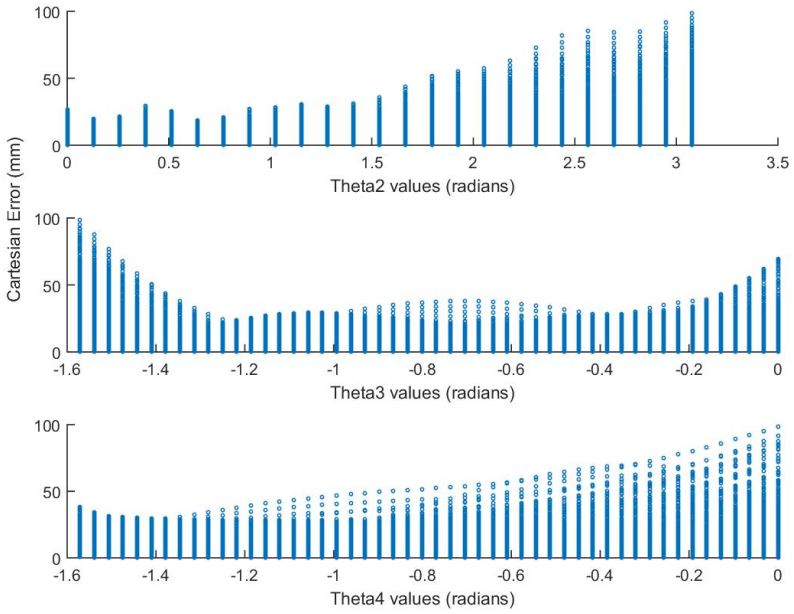

Figure 7: Cartesian RMSE in Joint Space

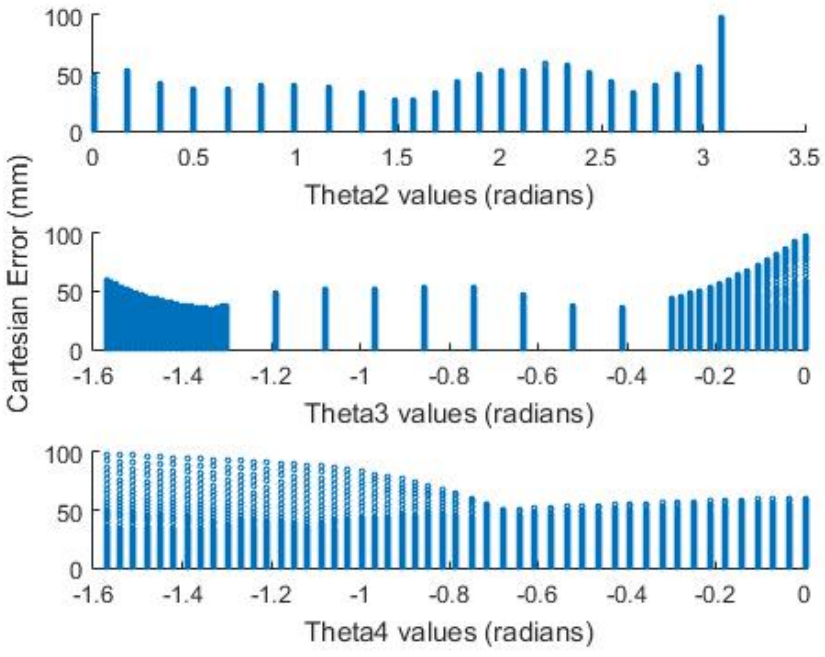

Figure 8: Cartesian RMSE in Joint Space after Change of Spread

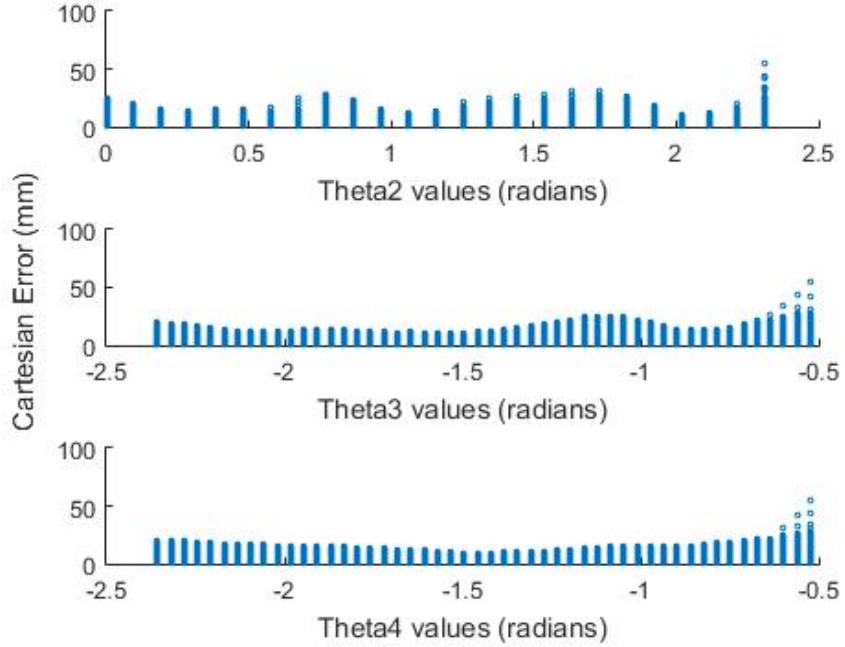

Figure 9: Cartesian RMSE in Joint Space after Reducing Joint Space Ranges

4 Lynxmotion Arm

In order to turn the planar RRR arm to a representation of the Lynxmotion robotic arm another variable θ_1, the rotation around the z-axis, is needed. As θ_1 only affects behaviour on the XY-plane it is unnecessary to train it upon Z and the pitch and similarly, there is no need to train θ_2, θ_3 or θ_4 on Y. As such the Limiting the input for the training of each angle to only what is necessary could help mitigate the scalability issues caused by the additional dimension. Unfortunately, keeping a training data density of fifty data points proved infeasible in the time frame and so this was reduced to just twenty data points.

A control run setting all the radii values to the default resulted in a disappointing Cartesian RMSE of 436.2mm with some errors reaching 800mm, with the validation error for θ_2, θ_3 and θ_4 a magnitude higher than for the planar RRR arm. This is most likely due to the reduction in training data density. For the final run with the radii values for θ_1 set to the default 0.5 and those for the other angles the same as for the planar RRR arm, the Cartesian RMSE error was 436.17mm and once again the validation RMSE is a magnitude higher than for the planar RRR arm.

Considering the membership functions of the FIS before and after training, it is disappointing to see that there is little change apart from for θ_4, implying that little training actually occurred. This is perhaps due to the low density of the training data.

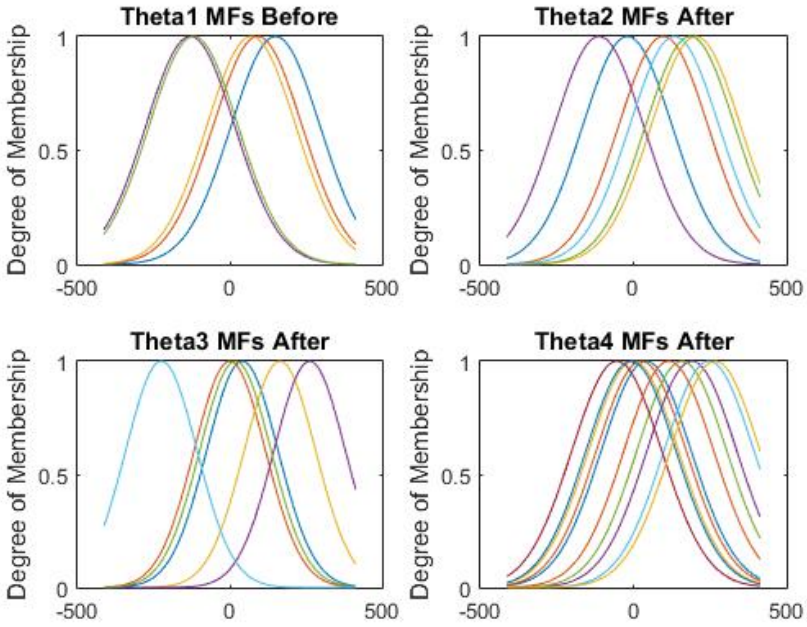

Figure 10: Membership Functions Before Training

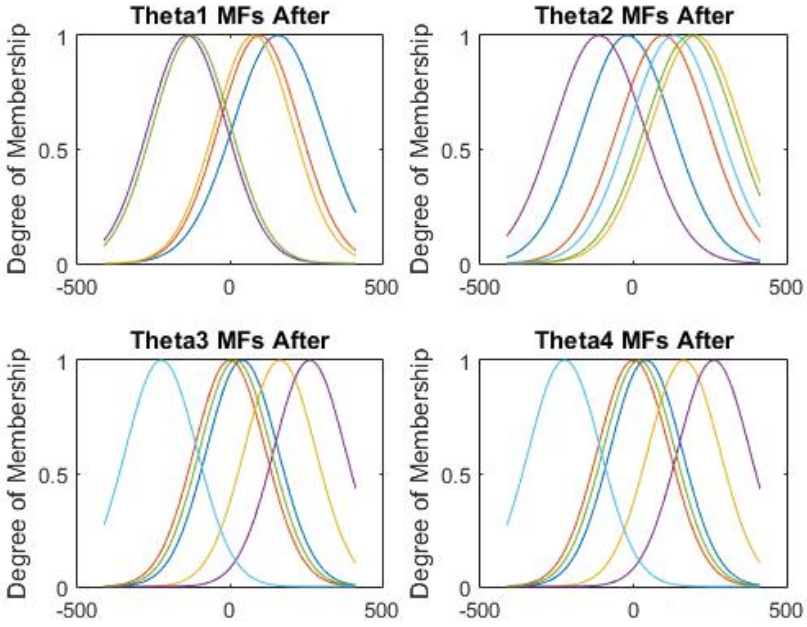

Figure 11: Membership Functions After Training

5 Conclusion

In conclusion, this report has explored the task of using ANFIS to calculate the inverse kinematics of the Lynxmotion Arm. Although the accuracy of the final method used is not idea, it is felt that, if the available time had been used more wisely to fully explore all the methods mentioned in the report, far better results could have been obtained.

6 Appendix 1 - Experiment Results

dataPoints	Epochs	MFs	trnRMSE2	chkRMSE2	trnRMSE3	chkRMSE3	cartRMSE	cartMin	cartMax	time
10	500	2	0.0441	0.1769	0.0595	0.2102	35.0791	2.3860	151.5126	0.32
10	500	3	0.0075	0.6026	0.0186	1.5039	88.9466	1.7816	300.7353	0.44
10	500	4	0.0016	1.5343	0.0029	2.8815	168.6123	3.1412	464.0428	0.99
10	500	5	0.0005	2.3307	0.0001	1.6801	194.8358	8.2084	519.1364	2.03
10	500	6	0.0000	0.7692	0.0000	1.1000	106.5546	1.1282	342.4532	3.91
10	500	7	0.0000	1.0441	0.0000	1.5253	191.3857	6.1158	625.5818	7.09
10	500	8	0.0000	0.8198	0.0000	0.5227	164.0823	2.6072	624.2153	11.81
10	500	9	0.0000	0.8749	0.0000	0.7083	161.5942	7.4095	629.0190	18.56
10	500	10	0.0000	1.0498	0.0000	0.7715	199.8085	4.0177	629.9484	28.28
20	500	2	0.0487	0.0707	0.0747	0.1185	12.4193	0.4108	103.9274	0.74
20	500	3	0.0196	0.0490	0.0395	0.0945	5.5069	0.3025	24.1058	1.74
20	500	4	0.0204	0.0871	0.0407	0.1851	11.8301	0.1035	64.8913	3.95
20	500	5	0.0083	1.0060	0.0151	1.4404	65.1519	0.4279	462.7651	8.43
20	500	6	0.0037	0.1839	0.0074	0.3333	35.6708	0.2279	256.3560	15.52
20	500	7	0.0018	0.3554	0.0035	0.5666	61.4179	0.3654	572.2944	28.16
20	500	8	0.0013	0.2046	0.0025	0.3664	41.7306	0.2340	212.7630	48.13
20	500	9	0.0011	0.3528	0.0015	0.3827	102.7713	0.4082	488.5110	73.61
20	500	10	0.0007	0.3804	0.0015	0.6136	109.5996	1.7098	521.3018	111.39
50	500	2	0.0371	0.0362	0.0694	0.0769	7.7817	0.1222	98.5986	4.46
50	500	3	0.0248	0.0250	0.0451	0.0512	5.1415	0.1326	99.1554	10.80
50	500	4	0.0206	0.0221	0.0411	0.0435	2.2461	0.0248	13.0735	23.72
50	500	5	0.0164	0.0177	0.0326	0.0348	1.5393	0.0175	7.7383	50.42
50	500	6	0.0124	0.0572	0.0253	0.0767	7.5921	0.0401	209.7884	96.92
50	500	7	0.0112	0.0242	0.0220	0.0487	2.1075	0.0208	23.5713	175.22
50	500	8	0.0103	0.0217	0.0195	0.0383	2.1102	0.0155	22.4959	292.86
50	500	9	0.0087	0.0214	0.0171	0.0451	2.8006	0.0122	26.8913	468.32
50	500	10	0.0081	0.0197	0.0159	0.0394	2.8987	0.0078	45.3157	699.17
100	500	2	0.0312	0.0311	0.0682	0.0687	5.5590	0.0651	111.7742	18.10
100	500	3	0.0236	0.0235	0.0446	0.0458	3.6464	0.0521	29.4106	42.97
100	500	4	0.0200	0.0202	0.0396	0.0400	1.9688	0.0156	10.9047	95.06
100	500	5	0.0158	0.0160	0.0314	0.0318	1.4229	0.0046	14.1410	201.28
100	500	6	0.0117	0.0132	0.0230	0.0261	0.9550	0.0016	8.5618	386.25
100	500	7	0.0105	0.0116	0.0199	0.0211	1.5349	0.0052	19.1859	700.12
100	500	8	0.0100	0.0107	0.0196	0.0212	1.1876	0.0100	35.0601	1167.94
100	500	9	0.0084	0.0096	0.0163	0.0189	0.7292	0.0023	7.2050	1854.52
100	500	10	0.0077	0.0107	0.0152	0.0205	1.7005	0.0072	49.1822	2776.96
200	500	2	0.0300	0.0301	0.0672	0.0675	5.6355	0.0679	168.8529	76.53
200	500	3	0.0236	0.0235	0.0440	0.0441	3.2618	0.0141	21.6430	175.64
200	500	4	0.0197	0.0197	0.0387	0.0388	1.8952	0.0042	11.8753	387.51
200	500	5	0.0146	0.0146	0.0295	0.0295	1.6488	0.0123	8.9011	816.60
200	500	6	0.0114	0.0115	0.0224	0.0226	0.8359	0.0016	6.2123	1554.44

Table 5: Planar RR Arm - Genfis1 Parameter Experiment

7 Appendix 2 - Matlab Code

```
clear
clc

%% Training data for ANFIS inverse kinematics system

l1=65; l2=155; l3=160; l5=100;


theta1 = linspace(-pi/2, pi/2, 20); % all possible theta1 values
theta2 = linspace(0, pi, 20); % all possible theta2 values
theta3 = linspace(-pi/2, 0, 20); %all possible theta3 values
theta4 = linspace(-pi/2, 0, 20); %all possible theta4 values


[THETA1, THETA2, THETA3, THETA4] = ndgrid(theta1, theta2, theta3, theta4); % generate a grid of theta va


%Forward Kinematics Equations
X = cos(THETA1).*(l3*cos(THETA2+THETA3)+l2*cos(THETA2)+l5*cos(THETA2+THETA3+THETA4));
Y = sin(THETA1).*(l3*cos(THETA2+THETA3)+l2*cos(THETA2)+l5*cos(THETA2+THETA3+THETA4));
Z = l1 + l3*sin(THETA2+THETA3)+l2*sin(THETA2)+l5*sin(THETA2+THETA3+THETA4);
B = -(THETA2+THETA3+THETA4);

data1 = [X(:) Y(:) THETA1(:)]; % create x-y-theta1 dataset
data2 = [X(:) Z(:) B(:) THETA2(:)]; % create x-z-b-theta2 dataset
data3 = [X(:) Z(:) B(:) THETA3(:)]; % create x-y-b-theta3 dataset
data4 = [X(:) Z(:) B(:) THETA4(:)]; % create x-y-b-theta4 dataset

training_data1 = data1(1:2:end,:);
training_data2 = data2(1:2:end,:);
training_data3 = data3(1:2:end,:);
training_data4 = data4(1:2:end,:);

validation_data1 = data1(2:2:end, :); % create x-y-theta1 dataset
validation_data2 = data2(2:2:end, :); % create x-y-theta2 dataset
validation_data3 = data3(2:2:end, :); % create x-y-theta3 dataset
validation_data4 = data4(2:2:end, :); % create x-y-theta4 dataset


%% Training

theta1_radii = 0.5;
theta2_radii = [0.5, 0.8, 0.5, 0.2];
theta3_radii = [0.4, 0.7, 0.5, 0.5];
theta4_radii = [0.5, 0.7, 0.5, 0.3];

tic;

%Genfis2
input_fismat1 = genfis2(training_data1(:,1:2), training_data1(:,3), theta1_radii);
input_fismat2 = genfis2(training_data2(:,1:3), training_data2(:,4), theta2_radii);
input_fismat3 = genfis2(training_data3(:,1:3), training_data3(:,4), theta3_radii);
input_fismat4 = genfis2(training_data4(:,1:3), training_data4(:,4), theta4_radii);


%ANFIS
numEpochs = 500;
```

```
fprintf('-->%s\n','Start training first ANFIS network.')
[training_fismat1,trnErr1,ss1,validation_fismat1,valErr1]=anfis(training_data1,input_fismat1,

fprintf('-->%s\n','Start training second ANFIS network.')
[training_fismat2,trnErr2,ss2,validation_fismat2,valErr2]=anfis(training_data2,input_fismat2,

fprintf('-->%s\n','Start training third ANFIS network.')
[training_fismat3,trnErr3,ss3,validation_fismat3,valErr3]=anfis(training_data3,input_fismat3,

fprintf('-->%s\n','Start training fourth ANFIS network.')
[training_fismat4,trnErr4,ss4,validation_fismat4,valErr4]=anfis(training_data4,input_fismat4,

t=toc;

%% Plot membership functions before training
figure(1)

subplot(2, 2, 1)
[ x1,mf1 ] = plotmf(input_fismat1,'input',1);
plot(x1,mf1);
xlabel('');
ylabel('Degree of Membership');

subplot(2, 2, 2)
[ x2,mf2 ] = plotmf(input_fismat2,'input',1);
plot(x2,mf2);
xlabel('');
ylabel('Degree of Membership');

subplot(2, 2, 3)
[ x3,mf3 ] = plotmf(input_fismat3,'input',1);
plot(x3,mf3);
xlabel('');
ylabel('Degree of Membership');

subplot(2, 2, 4)
[ x4,mf4 ] = plotmf(input_fismat4,'input',1);
plot(x4,mf4);
xlabel('');
ylabel('Degree of Membership');

% Plot membership functions after training
figure(2)

subplot(2, 2, 1)
[ y1,nf1 ] = plotmf(validation_fismat1,'input',1);
plot(y1,nf1);
xlabel('');
ylabel('Degree of Membership');

subplot(2, 2, 2)
[ y2,nf2 ] = plotmf(validation_fismat2,'input',1);
plot(y2,nf2);
xlabel('');
ylabel('Degree of Membership');

subplot(2, 2, 3)
[ y3,nf3 ] = plotmf(validation_fismat3,'input',1);
plot(y3,nf3);
xlabel('');
ylabel('Degree of Membership');
```

```
subplot(2, 2, 4)
[ y3,nf3 ] = plotmf(validation_fismat3,'input',1);
plot(y3,nf3);
xlabel('');
ylabel('Degree of Membership');
%% Checking Errors for Over/Underfitting

% %% Checking Errors for Over/Underfitting
epochs = 1:numEpochs;

figure(3) % new figure
%theta1
subplot(2,2,1);
plot(epochs, trnErr1, epochs, valErr2)

title('Theta1')
ylabel('Error')
xlabel('Epochs')
legend('training','validation')

%theta2
subplot(2,2,2);
plot(epochs, trnErr2, epochs, valErr2)

title('Theta2')
ylabel('Error')
xlabel('Epochs')
legend('training','validation')

%theta3
subplot(2,2,3);
plot(epochs, trnErr3, epochs, valErr3)

title('Theta3')
ylabel('Error')
xlabel('Epochs')
legend('training','validation')

%theta4
subplot(2,2,4);
plot(epochs, trnErr4, epochs, valErr4)

title('Theta4')
ylabel('Error')
xlabel('Epochs')
legend('training','validation')


%% Calculating the RMSE in Joint Space

%Theta1
trnOut1=evalfis(training_data1(:,1:2),training_fismat1);
trnRMSE1=norm(trnOut1-training_data1(:,3))/sqrt(length(trnOut1));
chkOut1=evalfis(validation_data1(:,1:2),validation_fismat1);
chkRMSE1=norm(chkOut1-validation_data1(:,3))/sqrt(length(chkOut1));

%Theta2
trnOut2=evalfis(training_data2(:,1:3),training_fismat2);
trnRMSE2=norm(trnOut2-training_data2(:,4))/sqrt(length(trnOut2));
chkOut2=evalfis(validation_data2(:,1:3),validation_fismat2);
chkRMSE2=norm(chkOut2-validation_data2(:,4))/sqrt(length(chkOut2));
```

```
%Theta1
trnOut3=evalfis(training_data3(:,1:3),training_fismat3);
trnRMSE3=norm(trnOut3-training_data3(:,4))/sqrt(length(trnOut3));
chkOut3=evalfis(validation_data3(:,1:3),validation_fismat3);
chkRMSE3=norm(chkOut3-validation_data3(:,4))/sqrt(length(chkOut3));

%Theta1
trnOut4=evalfis(training_data4(:,1:3),training_fismat4);
trnRMSE4=norm(trnOut4-training_data4(:,4))/sqrt(length(trnOut4));
chkOut4=evalfis(validation_data4(:,1:3),validation_fismat4);
chkRMSE4=norm(chkOut4-validation_data4(:,4))/sqrt(length(chkOut4));


%% Errors in Cartesian Space

X_out = cos(chkOut1).*(l3*cos(chkOut2+chkOut3)+l2*cos(chkOut2)+l5*cos(chkOut2+chkOut3+chkOut4
Y_out = sin(chkOut1).*(l3*cos(chkOut2+chkOut3)+l2*cos(chkOut2)+l5*cos(chkOut2+chkOut3+chkOut4
Z_out = l1 + l3*sin(chkOut2+chkOut3)+l2*sin(chkOut2)+l5*sin(chkOut2+chkOut3+chkOut4);

X_error = X_out - validation_data2(:,1);
Y_error = Y_out - validation_data2(:,2);
Z_error = Z_out - validation_data2(:,3);

cartesian_error = ((X_error).^2 + (Y_error).^2 + (Z_error).^2).^0.5;

cartesian_errorRMSE = norm(cartesian_error)/sqrt(length(cartesian_error));

figure(4)
subplot(2, 2, 1)
scatter(training_data1(:,3), cartesian_error, 5);
xlabel('Theta1 values (radians)');

subplot(2, 2, 2)
scatter(training_data2(:,4), cartesian_error, 5);
xlabel('Theta2 values (radians)');

subplot(2, 2, 3)
scatter(training_data3(:,4), cartesian_error, 5);
xlabel('Theta3 values (radians)');

subplot(2, 2, 4)
scatter(training_data4(:,4), cartesian_error, 5);
xlabel('Theta4 values (radians)');
ylabel('Cartesian Error (mm)');



%% Three Link Arm - Genfis2

l1=65; l2=155; l3=160; l5=100;

partitionNum = 50;

% theta2 = linspace(0, pi, partitionNum); % all possible theta2 values
% theta3 = linspace(-pi/2, 0, partitionNum); %Restricted theta3 values for elbow up configura
% theta4 = linspace(-pi/2, 0, partitionNum); %Restricted theta4 values for wrist up configura

theta2 = linspace(0, 3*pi/4, partitionNum); % all possible theta2 values
theta3 = linspace(-3*pi/4, -pi/6, partitionNum); %Restricted theta3 values for elbow up confi
theta4 = linspace(-3*pi/4, -pi/6, partitionNum); %Restricted theta4 values for wrist up confi

% theta2 = [linspace(0, pi/2, partitionNum*0.4), linspace(pi/2, pi, partitionNum*0.6)];
% theta3 = [linspace(0, -0.3, partitionNum*0.3), linspace(-0.3, -1.3, partitionNum*0.2), lins
```

```
% theta4 = [linspace(0, -pi/4, partitionNum*0.46), linspace(-pi/4, -pi/2, partitionNum*0.54)];


[THETA2, THETA3, THETA4] = ndgrid(theta2, theta3, theta4); % generate a grid of theta values

%Forward Kinematics Equations
X = l3*cos(THETA2+THETA3)+l2*cos(THETA2)+l5*cos(THETA2+THETA3+THETA4);
Z = l1 + l3*sin(THETA2+THETA3)+l2*sin(THETA2)+l5*sin(THETA2+THETA3+THETA4);
B = -(THETA2+THETA3+THETA4); %Pitch constraint

data2 = [X(:) Z(:) B(:) THETA2(:)];
data3 = [X(:) Z(:) B(:) THETA3(:)];
data4 = [X(:) Z(:) B(:) THETA4(:)];

training_data2 = data2(1:2:end,:);
training_data3 = data3(1:2:end,:);
training_data4 = data4(1:2:end,:);

validation_data2 = data2(2:2:end, :);
validation_data3 = data3(2:2:end, :);
validation_data4 = data4(2:2:end, :);

%% Training

theta2_radii = [0.5, 0.8, 0.5, 0.2];
theta3_radii = [0.4, 0.7, 0.5, 0.5];
theta4_radii = [0.5, 0.7, 0.5, 0.3];

tic;

%Genfis2
input_fismat2 = genfis2(training_data2(:,1:3), training_data2(:,4), theta2_radii);
input_fismat3 = genfis2(training_data3(:,1:3), training_data3(:,4), theta3_radii);
input_fismat4 = genfis2(training_data4(:,1:3), training_data4(:,4), theta4_radii);

%ANFIS

numEpochs = 500;

fprintf('-->%s\n','Start training theta2 ANFIS network.')
[training_fismat2,trnErr2,ss2,validation_fismat2,valErr2]=anfis(training_data2,input_fismat2,numEpochs,[

fprintf('-->%s\n','Start training theta3 ANFIS network.')
[training_fismat3,trnErr3,ss3,validation_fismat3,valErr3]=anfis(training_data3,input_fismat3,numEpochs,[

fprintf('-->%s\n','Start training theta4 ANFIS network.')
[training_fismat4,trnErr4,ss4,validation_fismat4,valErr4]=anfis(training_data4,input_fismat4,numEpochs,[

fprintf('-->%s\n','Finished training networks.')

t=toc;

% %% Checking Errors for Over/Underfitting
epochs = 1:numEpochs;

figure(1) % new figure

%theta2
subplot(1,3,1);
plot(epochs, trnErr2, epochs, valErr2)

title('Theta2')
```

```
ylabel('Error')
xlabel('Epochs')
legend('training','validation')

%theta3
subplot(1,3,2);
plot(epochs, trnErr3, epochs, valErr3)

title('Theta3')
ylabel('Error')
xlabel('Epochs')
legend('training','validation')

%theta4
subplot(1,3,3);
plot(epochs, trnErr4, epochs, valErr4)

title('Theta4')
ylabel('Error')
xlabel('Epochs')
legend('training','validation')


%% Root Mean Square Error in Joint Space

%Theta2
trnOut2=evalfis(training_data2(:,1:3),training_fismat2);
trnRMSE2=norm(trnOut2-training_data2(:,4))/sqrt(length(trnOut2));
chkOut2=evalfis(validation_data2(:,1:3),validation_fismat2);
chkRMSE2=norm(chkOut2-validation_data2(:,4))/sqrt(length(chkOut2));

%Theta3
trnOut3=evalfis(training_data3(:,1:3),training_fismat3);
trnRMSE3=norm(trnOut3-training_data3(:,4))/sqrt(length(trnOut3));
chkOut3=evalfis(validation_data3(:,1:3),validation_fismat3);
chkRMSE3=norm(chkOut3-validation_data3(:,4))/sqrt(length(chkOut3));

%Theta4
trnOut4=evalfis(training_data4(:,1:3),training_fismat4);
trnRMSE4=norm(trnOut4-training_data4(:,4))/sqrt(length(trnOut4));
chkOut4=evalfis(validation_data4(:,1:3),validation_fismat4);
chkRMSE4=norm(chkOut4-validation_data4(:,4))/sqrt(length(chkOut4));

%% Errors in Cartesian Space for Theta2

X_out = l3*cos(chkOut2+chkOut3)+l2*cos(chkOut2)+l5*cos(chkOut2+chkOut3+chkOut4);
Z_out = l1 + l3*sin(chkOut2+chkOut3)+l2*sin(chkOut2)+l5*sin(chkOut2+chkOut3+chkOut4);

X_error = X_out - validation_data2(:,1);
Z_error = Z_out - validation_data2(:,2);

cartesian_error = ((X_error).^2 + (Z_error).^2).^0.5;

cartesian_errorRMSE = norm(cartesian_error)/sqrt(length(cartesian_error));

figure(3)
subplot(3, 1,1)
scatter(training_data2(:,4), cartesian_error, 5);
xlabel('Theta2 values (radians)');

subplot(3,1,2)
scatter(training_data3(:,4), cartesian_error, 5);
```

```
xlabel('Theta3 values (radians)');

subplot(3,1,3)
scatter(training_data4(:,4), cartesian_error, 5);
xlabel('Theta4 values (radians)');
ylabel('Cartesian Error (mm)');



function [trnRMSE2, chkRMSE2, trnRMSE3, chkRMSE3, cartesian_error] = ANFIS_IK_2Link_Genfis1(partitionNum
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here


fprintf('-->%d %d %d\n', partitionNum, numMFs, numEpochs);
%% Training data for ANFIS inverse kinematics system

l1=65; l2=155; l3=160;

theta2 = linspace(0, pi, partitionNum); % all possible theta2 values
theta3 = linspace(-pi/2,0, partitionNum); %all possible theta3 values

[THETA2, THETA3] = ndgrid(theta2, theta3); % generate a grid of theta values

%Forward Kinematics Equations
X = l3*cos(THETA2+THETA3)+l2*cos(THETA2);
Z = l1 + l3*sin(THETA2+THETA3)+l2*sin(THETA2);

data2 = [X(:) Z(:) THETA2(:)];
data3 = [X(:) Z(:) THETA3(:)];

training_data2 = data2(1:2:end,:);
training_data3 = data3(1:2:end,:);

validation_data2 = data2(2:2:end, :);
validation_data3 = data3(2:2:end, :);

%% Training

%Genfis1
input_fismat2 = genfis1(training_data2, numMFs);
input_fismat3 = genfis1(training_data3, numMFs);

%ANFIS

fprintf('-->%s\n','Start training theta2 ANFIS network.')
[training_fismat2,trnErr2,ss2,validation_fismat2,valErr2]=anfis(training_data2,input_fismat2,numEpochs,[

fprintf('-->%s\n','Start training theta3 ANFIS network.')
[training_fismat3,trnErr3,ss3,validation_fismat3,valErr3]=anfis(training_data3,input_fismat3,numEpochs,[

fprintf('-->%s\n','Finished training networks.')
% %% Checking Errors for Over/Underfitting
if show == 1
    epochs = 1:numEpochs;

    figure(1) % new figure

    %theta2
    subplot(1,2,1);
    plot(epochs, trnErr2, epochs, valErr2)

    title('Theta2')
```

```
    ylabel('Error')
    xlabel('Epochs')
    legend('training','validation')

    %theta3
    subplot(1,2,2);
    plot(epochs, trnErr3, epochs, valErr3)

    title('Theta3')
    ylabel('Error')
    xlabel('Epochs')
    legend('training','validation')
end


%% Root Mean Square Error in Joint Space

%Theta2
trnOut2=evalfis(training_data2(:,1:2),training_fismat2);
trnRMSE2=norm(trnOut2-training_data2(:,3))/sqrt(length(trnOut2));
chkOut2=evalfis(validation_data2(:,1:2),validation_fismat2);
chkRMSE2=norm(chkOut2-validation_data2(:,3))/sqrt(length(chkOut2));

%Theta3
trnOut3=evalfis(training_data3(:,1:2),training_fismat3);
trnRMSE3=norm(trnOut3-training_data3(:,3))/sqrt(length(trnOut3));
chkOut3=evalfis(validation_data3(:,1:2),validation_fismat3);
chkRMSE3=norm(chkOut3-validation_data3(:,3))/sqrt(length(chkOut3));

%% Errors in Cartesian Space for Theta2

X_out = l3*cos(chkOut2+chkOut3)+l2*cos(chkOut2);
Z_out = l1 + l3*sin(chkOut2+chkOut3)+l2*sin(chkOut2);

X_error = X_out - validation_data2(:,1);
Z_error = Z_out - validation_data2(:,2);

cartesian_error = ((X_error).^2 + (Z_error).^2).^0.5;

if show==1
    figure(2)
    stem3(training_data2(:,3), training_data3(:,3), cartesian_error);
end

end



function [fitness] = ANFIS_IK_2Link_Genfis2(radii_orig)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here


%% Training data for ANFIS inverse kinematics system
l1=65; l2=155; l3=160;

partitionNum = 200;

theta2 = linspace(0, pi, partitionNum); % all possible theta2 values
theta3 = linspace(-pi/2,0, partitionNum); %all possible theta3 values

[THETA2, THETA3] = ndgrid(theta2, theta3); % generate a grid of theta values
```

```
%Forward Kinematics Equations
X = l3*cos(THETA2+THETA3)+l2*cos(THETA2);
Z = l1 + l3*sin(THETA2+THETA3)+l2*sin(THETA2);

data2 = [X(:) Z(:) THETA2(:)];
data3 = [X(:) Z(:) THETA3(:)];

training_data2 = data2(1:2:end,:);
training_data3 = data3(1:2:end,:);

validation_data2 = data2(2:2:end, :);
validation_data3 = data3(2:2:end, :);

%% Training

radii = abs(round(radii_orig,1)); %Need to ensure the radii is to one dp and positive.

theta2_radii = [radii(1), radii(2), radii(3)];
theta3_radii = [radii(4), radii(5), radii(6)];

tic;

%Genfis2
input_fismat2 = genfis2(training_data2(:,1:2), training_data2(:,3), theta2_radii);
input_fismat3 = genfis2(training_data3(:,1:2), training_data3(:,3), theta3_radii);


%ANFIS

numEpochs = 500;

fprintf('-->%s\n','Start training theta2 ANFIS network.')
[training_fismat2,trnErr2,ss2,validation_fismat2,valErr2]=anfis(training_data2,input_fismat2,numEpochs,[

fprintf('-->%s\n','Start training theta3 ANFIS network.')
[training_fismat3,trnErr3,ss3,validation_fismat3,valErr3]=anfis(training_data3,input_fismat3,numEpochs,[

fprintf('-->%s\n','Finished training networks.')

t=toc;

% %% Checking Errors for Over/Underfitting
% epochs = 1:numEpochs;
%
% figure(1) % new figure
% title('Training vs Validation Error - 3LinkGenfis150Thetas100Epochs3MFs');
%
% %theta2
% subplot(1,2,1);
% plot(epochs, trnErr2, epochs, valErr2)
%
% title('Theta2')
% ylabel('Error')
% xlabel('Epochs')
% legend('training','validation')
%
% %theta3
% subplot(1,2,2);
% plot(epochs, trnErr3, epochs, valErr3)
%
% title('Theta3')
% ylabel('Error')
```

```
% xlabel('Epochs')
% legend('training','validation')


%% Root Mean Square Error in Joint Space

%Theta2
trnOut2=evalfis(training_data2(:,1:2),training_fismat2);
trnRMSE2=norm(trnOut2-training_data2(:,3))/sqrt(length(trnOut2));
chkOut2=evalfis(validation_data2(:,1:2),validation_fismat2);
chkRMSE2=norm(chkOut2-validation_data2(:,3))/sqrt(length(chkOut2));

%Theta3
trnOut3=evalfis(training_data3(:,1:2),training_fismat3);
trnRMSE3=norm(trnOut3-training_data3(:,3))/sqrt(length(trnOut3));
chkOut3=evalfis(validation_data3(:,1:2),validation_fismat3);
chkRMSE3=norm(chkOut3-validation_data3(:,3))/sqrt(length(chkOut3));

%% Errors in Cartesian Space for Theta2

X_out = l3*cos(chkOut2+chkOut3)+l2*cos(chkOut2);
Z_out = l1 + l3*sin(chkOut2+chkOut3)+l2*sin(chkOut2);

X_error = X_out - validation_data2(:,1);
Z_error = Z_out - validation_data2(:,2);

cartesian_error = ((X_error).^2 + (Z_error).^2).^0.5;

cartesian_errorRMSE = norm(cartesian_error)/sqrt(length(cartesian_error));

fitness = cartesian_errorRMSE + 0.1*t;

% stem3(training_data2(:,3), training_data3(:,3), cart_error);

end


%% Genfis2 2Link GA

lb = [0.1; 0.1; 0.1; 0.1; 0.1; 0.1]; %lower constraint as radii values must be positive
ub = [0.9; 0.9; 0.9; 0.9; 0.9; 0.9]; %upper constraint as radii values must be <= 1
opts = gaoptimset('TimeLimit', 25200, 'PopulationSize', 15, 'PlotFcn',{@gaplotbestf,@gaplotma
[x, fval, exitflag, output, population] = ga(@ANFIS_IK_2Link_Genfis2, 6, [],[],[],[],lb,ub, [
```

References

[1] J. S. R. Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, May 1993.

[2] Mathworks - genfis1. urlhttp://uk.mathworks.com/help/fuzzy/genfis1.html?searchHighlight=genfis1 [Accessed: 16/04/2016].

[3] Mathworks - genfis2. `http://uk.mathworks.com/help/fuzzy/genfis2.html?searchHighlight=genfis2`, [Accessed: 16/04/2016].

[4] Frequently asked questions - anfis in the fuzzy logic toolbox. `http://www.cs.nthu.edu.tw/~jang/anfisfaq.htm`, [Accessed: 21/04/2016].