Intelligent and Adaptive Systems Assignment 1

Joshua Cox
UWE Student number: 18044472

3/04/2019

Intelligent and Adaptive systems - UFME7K-15-M - MSc Robotics

1 Examples of IK learning

Various different approaches have been used in inverse kinematics learning such as using Anfis, neural networks and genetic algorithms. By far the most popular are artificial neural networks, these can be broken down into various types. Anfis is a function approximator, however for functions which are harder to map the ruleset can be extremely large. A downside of using anfis is that only the sugeno fis type can be used for and it is not as easy to interpret, but does allow the use of a fuzzy inference system for complex tasks. The goal of these approaches is to map between the joint angles and the corresponding end effector positions and in some cases the end effector pose. This will result in a system which can return the corresponding joint angles for a chosen position, however in some cases where the manipulator is redundant there may be large joint displacements but some methods aim to address this.

One approach taken by Singh & Banga (2012) used Anfis to learn the inverse kinematics of both a 2 joint and 3 joint planar manipulators. The number of Anfis systems is the same as the number of joints, the inputs into the 2 joint system were the X and Y end effector locations, the input into the 3 joint system also consisted of only the X and Y end effector positions. The paper states that they achieved a suitable level of error, however this paper does not describe how many membership functions were used as well as the type.

Another study by Duka (2015) also used anfis for inverse kinematics learning, this study solely focused on a 3-joint manipulator. However the inputs were the X Y and Phi (EE pose) of the end effector position corresponding to the joint angles, but the input dataset was adjusted so that each XY location had a unique pose.. The study looked at testing different numbers of membership functions for the 3 Anfis systems, ranging from 3-6 membership functions for each input and each of these configurations were trained with varying samples sizes, however the paper doesn't state what number of membership functions performed the best; however it matched the chosen trajectories quite well although the scale of the input dataset. The manipulator used in this study has joint limits that cause redundancy, as the q3 limits range from -pi/2 to pi/2, which is potentially why this study uses phi as an input in comparison ot the study by Singh and Banga.

There are quite a few approaches to inverse kinematics learning that use neural networks. Another study by Duka (2014) looks at using a multi-layer perceptron for inverse kinematics, the inputs to the network are the X, Y and phi. The chosen structure had one hidden layer of 100 sigmoid activated neurons with 3 outputs neurons corresponding to the joint angles. The network can match the desired output arbitrary well. This study notes that further improvements to the network could be made, such as looking at the size and number of hidden layers. This paper is used as a reference for the MLP developed in this assignment.

One study by Mao & Hsia (1997) uses a neural network for IK learning. Two are developed, one for position and another which can also do obstacle avoidance. This approach uses the jacobian and the manipulator consists of 4 links and joints, however the inputs into the neural network were just the X and Y positions, the pose was not included. While this does make it lightweight it does mean that this approach is not suitable for certain applications.

A study by Yang et al. (2000) looked at both multi-layer perceptrons and radial basis functions for inverse kinematics of a 3R planar manipulator and compared them. The inputs to both types of network were the X, Y and Phi, it is worth noting that the manipulator used in this study has only positive joint angles which reduces the amount of redundancy which requires less complex networks. The paper describes details of both of the systems developed such as the learning used and the overall structure.

There are some approaches to inverse kinematics which are particularly suitable for redundant manip-

ulators, however these can also be used for those without redundancy. Such as one approach by Salaün et al. (2010) which is suitable for redundant manipulators is locally weighted projection regression. This study looked at a 3 link redundant manipulator, this is a function approximation approach (similar to other learning approaches) and maps between the joint angles and resulting end effector position and pose. The study covers the problems of learning inverse kinematic mappings from the forwards kinematics equations, as there can be an unlimited amount of mappings for redundant manipulators. Instead the study looks at using two approaches the projection method and the extended jacobian method, which instead use the forwards kinematics mappings and invert them to keep the possibilities.

One approach by Momani et al. (2016) was to use genetic algorithms for inverse kinematics. This study compares its approach to genetic algorithms for IK that have not looked at redundancy, which can also be a problem for neural network based approaches. This was tested over manipulators with various numbers of links, for example for a 3R redundant manipulator the joint limits were -180 to 180 and the error for the chosen trajectories is low. The paper covers the amount of generations required for each of the manipulators and what initilization and crossover type result in the most smooth trajectories, while the standard genetic algorithm approach that this was compared to resulted in highly oscillatory movement due to the redundancy.

2 Inverse kinematics with Anfis

2.1 Data/workspace generation

In order to generate a dataset that Anfis could be trained on a workspace was generated using the forward kinematics equations for a 3R planar manipulator with unitless link lengths of 10,7,5 for L1,L2 and L3 respectively. These are:

$$X = L1 \times \cos(q1) + L2 \times \cos(q1 + q2) + L3 \times \cos(q1 + q2 + q3)$$

$$Y = L1 \times \sin(q1) + L2 \times \sin(q1 + q2) + L3 \times \sin(q1 + q2 + q3)$$

$$\phi = q1 + q2 + q3$$

End effector positions were calculated across a range of joint angles, the joint limits for each joint are 0 to pi for q1, from 0 to pi/2 for q2 and from -pi/2 to pi/2 for q3. Note that problems may arise due to the possibility of q3 being either positive or negative, which will result in locations with multiple solutions. The size of the dataset can be adjusted by changing the interval in the q arrays (this can be done for a specific angle or for all). The dataset for the pose (Phi) is also created at this time, as it is the summation of the joint angles. The study by Duka (2015) uses phi as an input due to the manipulator having redundancy as its q3 has the same range as the manipulator for this assignment, this is approach that will be used.

The end result is a dataset with a variety of end effector poses (phi's) and their associated joint angles (which will be the training output). The dataset was adjusted to reduce the density of end effector positions in certain locations (due to the joint limits of the third joint creating redundancy for some points), this was also done by Duka (2015). This approach resulted in a dataset of roughly 2500 samples (2523), the generated workspace is shown in figure 1.

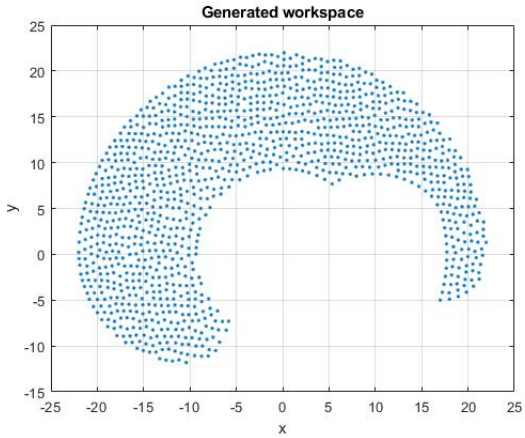
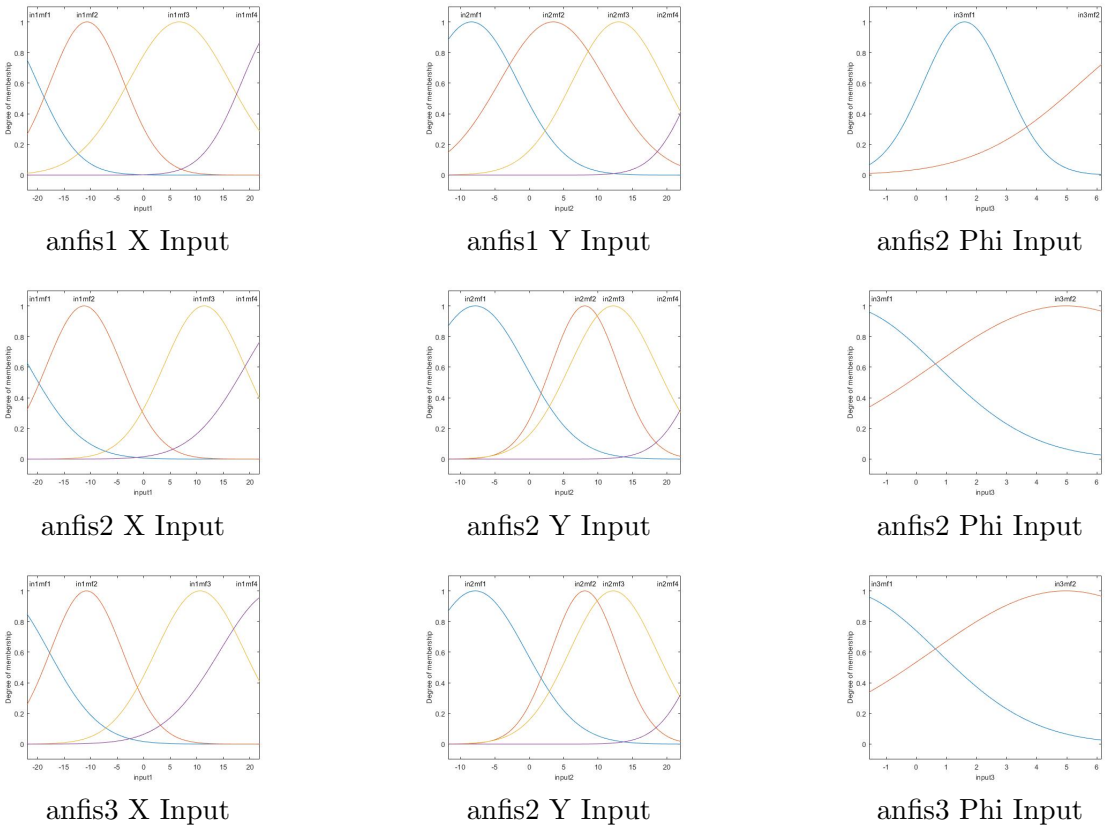


Figure 1: Generated workspace for Anfis training

2.2 Anfis structure and training

For learning inverse kinematics, a set of 3 Anfis systems were used, the inputs to each were the X and Y of the generated end effector locations as well as the corresponding Phi(pose), the training outputs of each system were the corresponding joint angle theta(q), this is similar to the approach taken by Singh & Banga (2012). Genfis was used to generate the initial fis for each of the anfis systems as this allows more customization in comparison to just using anfis, the membership functions are all gaussian. Grid partitioning was used to avoid problems with dimensionality due to the number of inputs, as these generally cause an excessively large ruleset which causes training time to increase exponentially with the number of membership functions and size of the dataset.

The number of membership functions for each of the systems in the study by Duka (2015) ranged from 3-6 for each input, the number of epochs also varied for each of the systems that were trained. The will be the starting parameters for this assignment were based of this approach and were 5,5,5 with 100 epochs. After optimization, the final number of membership functions was chosen to be 4,4,2 for X,Y and Phi respectively, which is shown in figure 2. These values were selected by testing which number of membership functions gave the most accurate inverse kinematics and generalization. The total size of the ruleset for each of these systems is 32 and the number of epochs to train each of the anfis systems was 300 for the systems responsible for q1 and q2 while it was 200 for q3. During optimization the number of epochs was adjusted to ensure generalization across the workspace.



anfis1 X Input | anfis1 Y Input | anfis2 Phi Input

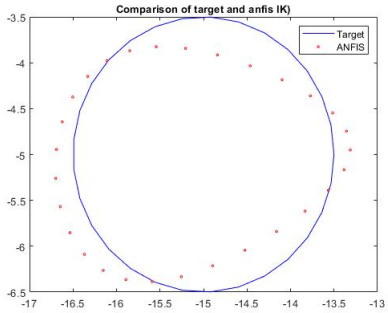
anfis2 X Input | anfis2 Y Input | anfis2 Phi Input

anfis3 X Input | anfis2 Y Input | anfis3 Phi Input

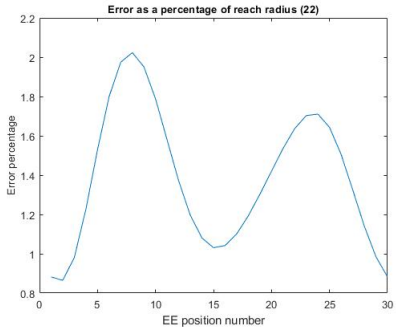
Figure 2: Anfis membership functions after training

2.3 Validation

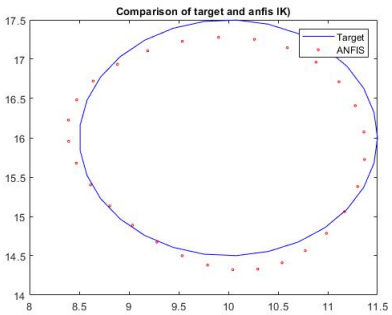
The chosen error metric for validation is the error as a percentage of the reach radius (Which is 22 units). In order to validate the system, several XY positions were input into the system and the theta(q) that is output was then fed into the forwards kinematics equations and plotted. The calculated end effector position by Anfis was then compared to the desired position using the error metric. It was tested over a range of XY locations as well as poses to ensure good generalization. The inverse kinematics of the manipulator were also calculated analytically, this enabled a comparison between the output joint angles and positions of both systems which also meant that test configurations could be evaluated to ensure they were possible. The overall error of the system is suitably low as shown by figure 3 which shows the error across each of the chosen XY validation points at various poses, the curve trajectory changes pose throughout whereas the square and circle trajectories have a constant pose.



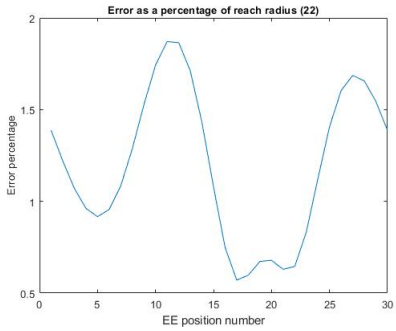
Left side Circle I.K
Pose = 3.6



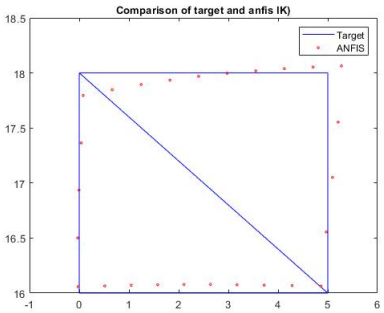
Left side circle I.K error %



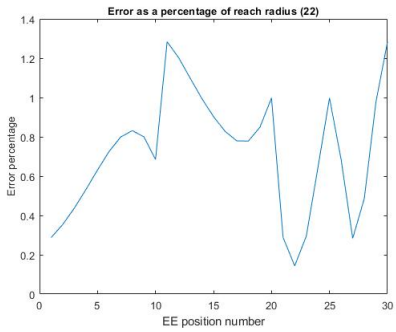
Right side circle I.K
Pose = pi/3



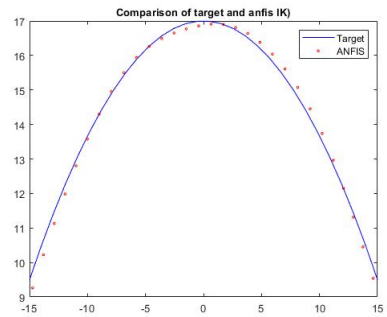
Right side Circle I.K error %



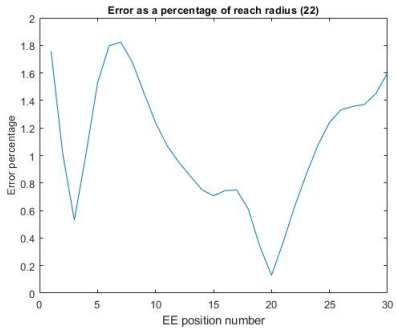
Square I.K
Pose = pi/1.5



Square I.K error %



Curve I.K
Pose = 0.5:3



Curve I.K error %

Figure 3: Comparison of ANFIS I.K to target with pose in radians) (left) and error as a percentage of reach radius (right)

2.4 Evaluation and discussion

The dataset for the training is not that large which is one of the benefits of using anfis, yet the accuracy is suitable for inverse kinematics tasks with errors less than 2.2% of reach radius for the tested positions and poses. While the error shown for inverse kinematics tasks is suitably low, the error is higher at the edges of the dextrous workspace and near pose limits for certain XY locations, which is evident in the circle trajectory in figure 3. So the performance along a trajectory may result in oscillatory behaviour as a result of large joint velocities. This might be due to the fact the 3rd joints limits (-pi/2,pi/2) create redundancy for some locations or could be due to the dataset not having a suitable variety of poses, the quality of the dataset affects the performance immensely. The dataset could be improved by altering the function that decreases the density in certain areas of the workspace, it currently just sorts by XY locations and is indiscriminate towards pose which may result in an uneven spread of poses across the workspace. The accuracy could also be improved with more membership functions, however the training time increase exponentially with each one added. While the membership functions used were gaussian, other types may be more suitable.

While this approach allows the application of fuzzy logic to complex mapping problems, the readability is mostly lost with anfis when the ruleset is so large. The overall approach to the design of this system was to go from the start point suggested by research papers and use trial and error/validation to optimize the number of membership functions, this is not an efficient method to improve inverse kinematics with ANFIS.

3 Inverse kinematics with MLP

3.1 Data/workspace generation

In comparison to Anfis multi-layer perceptron's learn from a much larger dataset. The dataset was generated in a similar fashion, however the intervals were much smaller resulting in a much denser workspace with a larger dataset of 5250 samples as a result, as shown in figure 4. The inputs into the MLP are the X Y and pose of the end effector. Which are calculated using the same FK equations as used previously with anfis.

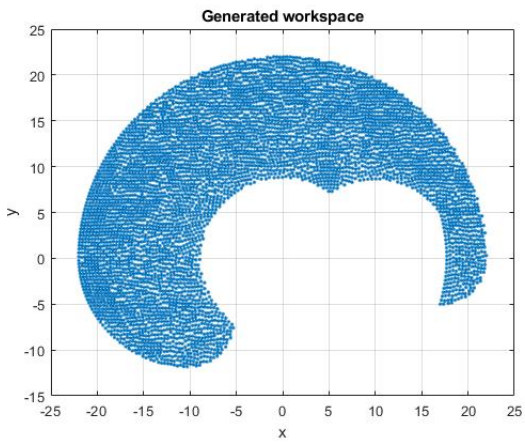


Figure 4: Generated workspace for MLP training

3.2 MLP structure and training

The study by Duka (2014) used one hidden layer with 100 nodes for IK learning. However, it is noted in the study that further work involving optimization of the network is necessary. As such the starting

point for this assignment was with these parameters. The final structure of the MLP is shown in figure 5, it consists of 3 hidden layers with

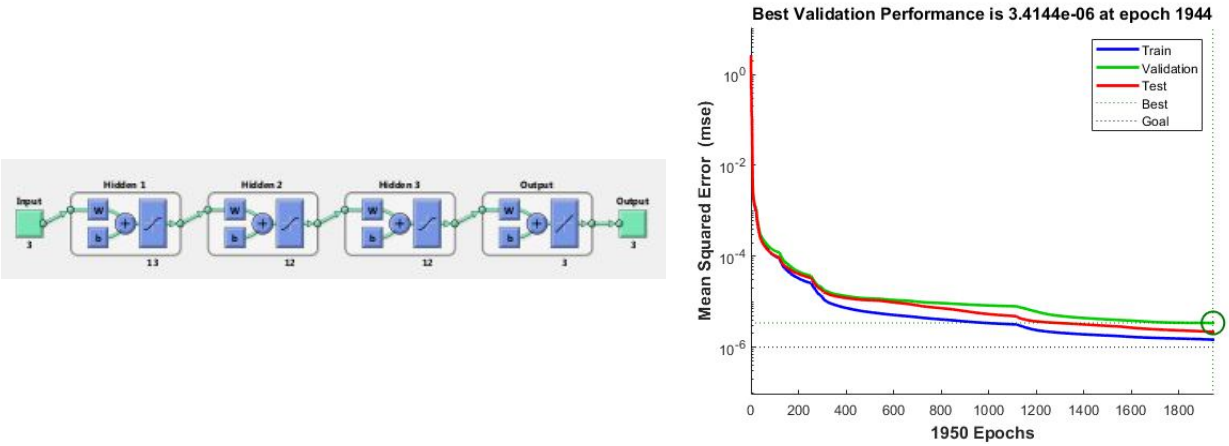
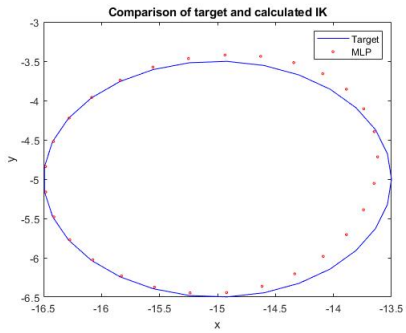


Figure 5: MLP Structure(left) and performance(right)

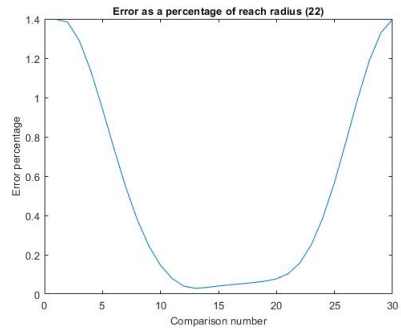
The training performance of the MLP is also shown in 5 The chosen training method for the MLP is trainlm. The generated dataset was partitioned into training, validation and testing, with the ratio 0.7:0.15:0.15 respectively. The number of max epochs was set to 3000, however this was unnecessary as the validations checks stopped the training when the validation error no longer improves. The minimum performance goal was set to 1e-6 and the minimum gradient was set to 1e-6/100, this was to ensure that training continued until the validation accuracy stopped improving.

3.3 Validation

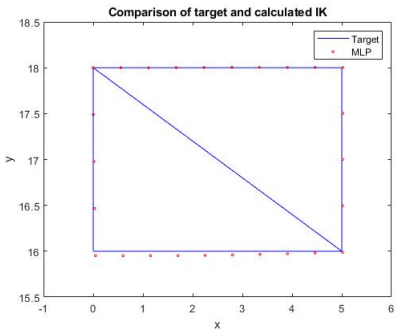
While an advantage of MLP's is that a validation set is made from sampling original dataset. While the networked undergoes validation during training, testing the network on other data is necessary to ensure the inverse kinematics are within a suitable margin of error. The error metric used for this is the same as the one used in the Anfis system. Which is the distance between target position and calculated position as a percentage of the total reach radius. Figure showns 3 sets of validation points, the MLP was tested across various shapes and poses to ensure good generalization. The error for each of these is shown in figure 6.



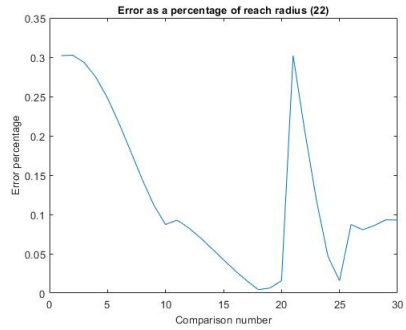
Circle I.K comparison



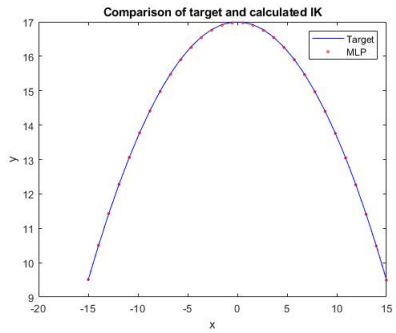
Circle I.K error %



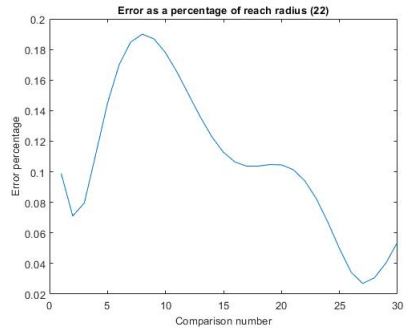
Square I.K comparison



Square I.K error %



Curve I.K comparison



Curve I.K error %

Figure 6: Comparison of MLP I.K to target (left) and associated error (right)

3.4 Evaluation and discussion

The dataset used for this approach was much larger than that of the Anfis system. This may be a factor to take into consideration when using either of these approaches for inverse kinematics, as a large dataset may not always be possible. Neural networks using validation during training is useful for gauging the initial performance of the network design. Another advantage to using a neural network based approach is that it is easier to code and optimize.

The overall accuracy of the network is suitably low for inverse kinematics, however the MLP has greater error with certain configurations, due to the q3 joint angles being from -pi/2 to pi/2. This is due to the training dataset not containing enough poses for the end effector positions on the edges of the workspace (These are still within the dextrous workspace; which is confirmed by analytical inverse kinematics). However, this can be offset by having a larger dataset but that results in a longer training time and perhaps a way to reduce the number of same poses for certain positions would be advantagous.

4 Problems with IK learning

One problem with the learning of inverse kinematics for certain manipulators is redundancy (D'Souza et al. 2001). This means that several end effector positions have various different configurations that could be used, however since there is no difference to the system from one configuration to another(unless the pose is set), this can result in extremely high joint velocities and osciallatory behaviour dependant on the relationship learnt. This is something that some studies have looked to address, the aforementioned studies by Momani et al. (2016) and by Salaün et al. (2010) are a couple of these.

Another problem with IK learning is that inverse kinematics have a non-convexity property which means that direct mapping cannot be done without making converting it to a convex problem by using a small region of theta (D'Souza et al. 2001).

One of the other major problems with IK learning is that as the complexity of the manipulator increases the computational effort increases rapidly which isn't suitable for on-line applications, this has resulted in some approaches that aim to be lightweight or some approaches that deal with small change in theta such as the pseudo-inverse of the jacobian (D'Souza et al. 2001).

Another problem with some approaches to inverse kinematics learning is joint angle drift, this can happen with approaches based on the pseudo-inverse. The pseudo-inverse is also not suitable for velocity or accelration control (Zhang et al. 2009).

One study by da Graça Marcos et al. (2012) uses a genetic algorithm for both a 3 link and 4 link manipulator. The error was remarkbly low and the approach also takes into account the joint displacement. The initial joint configurations are calculated using an extended jacobian matrix. The weighting of the two different optimization critera (position and displacement) greatly affects the behaviour, this is something the user can adjust however depending on the application. A higher position weight results in accuracy but larger displacements which may not be desirable. The final note of the study is that this method requires a lot of computational power and is not currently applicable to real time tasks.

Zhang et al. (2009) compared 3 different sets of dual neural networks to solve the inverse kinematics as a quadratic programming problem. This was in an effort to reduce joint angle drift which is common with approaches that use the inverse jacobian for inverse kinematics. These were a conventional dual network and 2 LVI-based primal-dual neural networks. The overall error for each of these is very low with the LVI-based being the best 2, however the run time for the conventional dual network was lower in comparison.

Another study by Chen & Lau (2016) looks at blending 2 support vector machines. This study compares the approaches by da Graça Marcos et al. (2012) and Zhang et al. (2009) and others. The paper states that none of these are without drawbacks, some are not suitable for certain tasks and some are difficult to implement. The prediction accuracy is extremely high for this method, the only drawbacks is that it is currently incapable of velocity and torque control, but this is stated to be implemented in the future.

5 Search algorithms

Genetic algorithms have been used for inverse kinematics learning, but they can also be used to find optimal parameters for a neural network or anfis network for the same task. For neural networks this is done by having the GA update the weights and thresholds based on the error between the outputs and targets.

The general method for optimisation with a genetic algorithm, is as follows (Yuan et al. 2014) :

1. The system to optimised has an initial pass and trains until satisifed.

2. The output of the system is then converted into a fitness function.
3. This is then fed into the genetic algorithm.
4. Crossover and mutation occur.
5. The new fitness value is calculated and not satisfactory this is re-fed into the GA.
6. When finished the GA outputs the new parameters into the original system.
7. The original system trains until satisified.
8. This cycle contains until a satisfactory overall error is reached.

This is the approach taken by Yuan et al. (2014) , however this study did not involve inverse kinematics. The genetic algorithm was used alongside a back-propogated neural network. For anfis this is done by the GA optimizing its parameters such as the number of membership functions to reduce the error. This approach has been done by Cheng et al. (2002), however this study was also not involved with inverse kinematics, but does show that anfis can be optimised in this way, rather than with trial and error which can be very time consuming.

References

Chen, J. & Lau, H. Y. K. (2016), Inverse kinematics learning for redundant robot manipulators with blending of support vector regression machines, *in* '2016 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)', pp. 267–272.

Cheng, C.-B., Cheng, C.-J. & Lee, E. (2002), 'Neuro-fuzzy and genetic algorithm in multiple response optimization', *Computers & Mathematics with Applications* **44**(12), 1503 – 1514.
URL: *http://www.sciencedirect.com/science/article/pii/S0898122102002742*

da Graça Marcos, M., Machado, J. T. & Azevedo-Perdicoúlis, T.-P. (2012), 'A multi-objective approach for the motion planning of redundant manipulators', *Applied Soft Computing* **12**(2), 589 – 599.
URL: *http://www.sciencedirect.com/science/article/pii/S1568494611004303*

D'Souza, A., Vijayakumar, S. & Schaal, S. (2001), Learning inverse kinematics, *in* 'Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)', Vol. 1, IEEE, pp. 298–303.

Duka, A.-V. (2014), 'Neural network based inverse kinematics solution for trajectory tracking of a robotic arm', *Procedia Technology* **12**, 20 – 27. The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania.
URL: *http://www.sciencedirect.com/science/article/pii/S2212017313006361*

Duka, A.-V. (2015), 'Anfis based solution to the inverse kinematics of a 3dof planar manipulator', *Procedia Technology* **19**, 526 – 533. 8th International Conference Interdisciplinarity in Engineering, INTER-ENG 2014, 9-10 October 2014, Tirgu Mures, Romania.
URL: *http://www.sciencedirect.com/science/article/pii/S2212017315000766*

Mao, Z. & Hsia, T. C. (1997), 'Obstacle avoidance inverse kinematics solution of redundant robots by neural networks', *Robotica* **15**(1), 3–10.

Momani, S., Abo-Hammour, Z. S. & Alsmadi, O. M. (2016), 'Solution of inverse kinematics problem using genetic algorithms', *Applied Mathematics & Information Sciences* **10**(1), 225.

Salaün, C., Padois, V. & Sigaud, O. (2010), Learning forward models for the operational space control of redundant robots, *in* 'From motor learning to interaction learning in robots', Springer, pp. 169–192.

Singh, G. & Banga, D. V. K. (2012), Anfis implementation for robotic arm manipulator.

Yang, S. S., Moghavvemi, M. & Tolman, J. D. (2000), Modeling of robot inverse kinematics using two ann paradigms, *in* '2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No.00CH37119)', Vol. 3, pp. 173–177 vol.3.

Yuan, Z., Wang, L.-N. & Ji, X. (2014), 'Prediction of concrete compressive strength: Research on hybrid models genetic based algorithms and anfis', *Advances in Engineering Software* **67**, 156 – 163.
URL: *http://www.sciencedirect.com/science/article/pii/S0965997813001580*

Zhang, Y., Tan, Z., Chen, K., Yang, Z. & Lv, X. (2009), 'Repetitive motion of redundant robots planned by three kinds of recurrent neural networks and illustrated with a four-link planar manipulator's straight-line example', *Robotics and Autonomous Systems* **57**(6), 645 – 651.
URL: *http://www.sciencedirect.com/science/article/pii/S0921889009000037*

Appendices

A Anfis IK code

```
%Inverse kinematics of a 3 link planar manipulator using Anfis
clear;
clc;

%Set Ik test type, can be leftCircle, rightCircle, leftLine, rightLine, curve,
%leftCurve,square
testType = "rightCircle";

%number of epochs for each system
numEpochs1 = 300;
numEpochs2 = 300;
numEpochs3 = 200;

%workspace resolution, this adjusts density
workSpaceRes = 0.02;%0.02


%RRR planar manipulator IK with Anfis
l1 = 10; % length of first arm
l2 = 7; % length of second arm
l3 = 5; % length of third arm


%%
%workspace generation
theta1 = 0:0.025:pi; % all possible theta1 values
theta2 = 0:0.1:pi/2; % all possible theta2 values
theta3 = -pi/2:0.1:pi/2;% all possible theta2 values

%  theta1 = 0:0.1:pi; % all possible theta1 values
%  theta2 = 0:0.1:pi/2; % all possible theta2 values
%  theta3 = -pi/2:0.1:pi/2;% all possible theta2 values

[THETA1,THETA2,THETA3] = meshgrid(theta1,theta2,theta3);

%%
%FK calculations
FKX = (l1 * cos(THETA1)) + (l2 * cos(THETA1 + THETA2)) + (l3 * cos(THETA1 + THETA2 + THETA3));
FKY = (l1 * sin(THETA1)) + (l2 * sin(THETA1 + THETA2)) + (l3 * sin(THETA1 + THETA2 + THETA3));
PHI = THETA1 + THETA2 + THETA3; %one paper uses this, one does not?


%%
%create an array of all values and uniquetol it for xy and for phi

fullArray = [FKX(:), FKY(:), PHI(:), THETA1(:), THETA2(:), THETA3(:)];

c=fullArray(:,1:2); %sorts by x,y locations that aren't too close together

[~,idx]=uniquetol(c,workSpaceRes,'ByRows',true);
sortedArray=fullArray(idx,:);


%Set net variables since we don't want to change all the code

FKX = sortedArray(:,1);
FKY = sortedArray(:,2);
PHI = sortedArray(:,3);
THETA1 = sortedArray(:,4);
THETA2 = sortedArray(:,5);
THETA3 = sortedArray(:,6);



%%
%Plot the generated workspace?
figure(1);
plot(FKX(:), FKY(:),'.');
title('Generated workspace')
grid on;
xlabel('x')
ylabel('y')
xlim([-25 25]);
ylim([-15 25]);
hold on;



%%
%genfis datasets
data1 = [FKX(:) FKY(:)  PHI(:)]; % create x-y-phi-theta1 dataset
data2 = [FKX(:) FKY(:)  PHI(:)]; % create x-y-phi-theta2 dataset
data3 = [FKX(:) FKY(:)  PHI(:)];% create x-y-phi-theta3 dataset
%anfis datasets
fulldata1 = [FKX(:) FKY(:)  PHI(:) THETA1(:)];
fulldata2 = [FKX(:) FKY(:)  PHI(:) THETA2(:)];
fulldata3 = [FKX(:) FKY(:)  PHI(:) THETA3(:)];
```

```
%%
%Initial genfis gridpartition
genOpt = genfisOptions('GridPartition');
%gridpartition options
genOpt.NumMembershipFunctions = [4 4 2]; %membership function number for inputs x y and phi as suggested by paper 2
genOpt.InputMembershipFunctionType = ["gaussmf" "gaussmf" "gaussmf" ];
genOpt.OutputMembershipFunctionType = "linear";

%%
%set number of epochs for anfis training

%First genfis
disp('--> first GENFIS.')
inFIS1 = genfis(data1,THETA1(:),genOpt);

% %Second genfis
disp('--> second GENFIS.')
inFIS2 = genfis(data2,THETA2(:),genOpt);

% %third genfis
disp('--> third GENFIS.')
inFIS3 = genfis(data3,THETA3(:),genOpt);

%%
%anfis setup & suppress outputs
opt1 = anfisOptions('InitialFIS',inFIS1);
% opt1 = anfisOptions;
opt1.DisplayANFISInformation = 0;
opt1.DisplayErrorValues = 0;
opt1.DisplayStepSize = 0;
opt1.DisplayFinalResults = 0;
opt1.ErrorGoal = 1e-4;
%opt1.InitialFIS = 3;
%opt1.ValidationData = checkdata1;
%anfis2
opt2 = anfisOptions('InitialFIS',inFIS2);
%opt2 = anfisOptions;
opt2.DisplayANFISInformation = 0;
opt2.DisplayErrorValues = 0;
opt2.DisplayStepSize = 0;
opt2.DisplayFinalResults = 0;
opt2.ErrorGoal = 1e-4;
%opt2.InitialFIS = 4;
%opt2.ValidationData = checkdata2;
%anfis3
opt3 = anfisOptions('InitialFIS',inFIS3);
%opt3 = anfisOptions;
opt3.DisplayANFISInformation = 0;
opt3.DisplayErrorValues = 0;
opt3.DisplayStepSize = 0;
opt3.DisplayFinalResults = 0;
opt3.ErrorGoal = 1e-4;
%opt3.InitialFIS = 3; %only for use with just anfis
%opt3.ValidationData = checkdata3;


%%
% Train anfis1.
disp('--> Training first ANFIS network.')
opt1.EpochNumber = numEpochs1;
anfis1 = anfis(fulldata1,opt1);


%Train anfis2
disp('--> Training second ANFIS network.')

opt2.EpochNumber = numEpochs2;
anfis2 = anfis(fulldata2,opt2);

%train anfis3
disp('--> Training third ANFIS network.')
opt3.EpochNumber = numEpochs3;
anfis3 = anfis(fulldata3,opt3);

%% Validation tests
%straight line right side test
if testType == "rightLine"
    X = linspace(12,0,30); % x coordinates for validation
    Y(1:30) = 14; % y coordinates for validation
    phiV(1:30) = pi/3; % phi for validation
end

%straight line left side test
if testType == "leftLine"
    X = linspace(-14,3,30);
    Y(1:30) = 14;
    phiV(1:30) = pi;
end

%Curve through workspace test
if testType == "curve"
    X = linspace(15,-15,30);
    Y = 17 - abs(X.^2)/30;
    phiV = linspace(0.5,3,30);
end

%curve on left side side
```

```
if testType == "leftCurve"
    Y = linspace(0,15,30);
    X = -18 + abs(Y.^2)/40;
    phiV(1:30) = 0;
end

% %circle in workspace test
if testType == "leftCircle"
    angle = linspace(0,pi,30);
    X = -15 + 1.5*cos(2*angle);
    Y = -5 + 1.5*sin(2*angle);
    phiV(1:30) = 3.6;

end

% %circle in workspace test
if testType == "rightCircle"
    angle = linspace(0,pi,30);
    X = 10 + 1.5*cos(2*angle);
    Y = 16 + 1.5*sin(2*angle);
    phiV(1:30) = pi/3;

end

% %circle in workspace test
if testType == "square"
   X = [linspace(0,5,10) linspace(5,0,10)];
   Y(1:10) = 16;
   Y(11:20) = 18;
   X(21:25) = 0;
   X(26:30) = 5;
   Y(21:25) = linspace(16,18,5);
   Y(26:30) = linspace(16,18,5);
   phiV(1:30) = pi/1.5;
end

%%
%plot validation xy coordinates
figure(2);
plot(X,Y,'-b');
title('Comparison of target and anfis IK)','fontsize',10);
%xlim([-25 25]);
%ylim([-15 25]);
hold on;

%%%
%%Actual IK calculations, for comparison with anfis, uncomment to use
%a = Y - (l3*sin(phiV)); % Y of wrist
%b = X - (l3*cos(phiV)); % X of wrist
%%Get theta2
%D = ( ((a.^2)+(b.^2) - l1^2 - l2^2)/(2*l1*l2)); %Costheta2
%THETA2D = real(acos(D));
%%Get theta1
%k1 = l2*sin(THETA2D);
%k2 = l1 + l2*cos(THETA2D);
%THETA1D = atan2((a.*k2)-(k1.*b),(a.*k1)+(b.*k2));
%%get theta 3
%THETA3D = phiV - (THETA1D + THETA2D);

%Plot the calculated IK X&Y positions, this is green, can be used to test
%%if its a reasonable demand of the system
%valX = (l1 * cos(THETA1D)) + (l2 * cos(THETA1D+ THETA2D)) + (l3 * cos(THETA1D+THETA2D+THETA3D));
%valY = (l1 * sin(THETA1D)) + (l2 * sin(THETA1D+ THETA2D)) + (l3 * sin(THETA1D+THETA2D+THETA3D));
%plot(valX(:),valY(:),'-g');

%%
%evaluate anfis for test coordinates

XY = [X' Y' phiV'];

THETA1P = evalfis(XY,anfis1); % theta1 predicted by anfis1
THETA2P = evalfis(XY,anfis2); % theta2 predicted by anfis2
THETA3P = evalfis(XY,anfis3); % theta3 predicted by anfis3


testX = (l1 * cos(THETA1P)) + (l2 * cos(THETA1P+ THETA2P)) + (l3 * cos(THETA1P+THETA2P+THETA3P));
testY = (l1 * sin(THETA1P)) + (l2 * sin(THETA1P+ THETA2P)) + (l3 * sin(THETA1P+THETA2P+THETA3P));
plot(testX,testY,'or','MarkerSize',2);
legend('Target','ANFIS');
hold off;

%%
%Error as a percentage of reach radius,Calc distance between target and calculated
error = sqrt((X(:)-testX(:)).^2 + (Y(:)-testY(:)).^2);
%convert to a percentage of reach radius (22)
percError = error/22 * 100;
%plot the error
figure(3)
plot(percError);
ylabel('Error percentage','fontsize',10);
xlabel('EE position number');
title('Error as a percentage of reach radius (22)','fontsize',10);
```

B MLP IK code

```
%Inverse kinematics of a 3 link planar manipulator using a MLP
clear;
clc;
%Set Ik test type, can be circle, leftLine, rightLine, curve,
%leftCurve,square
testType = "circle";

%%
%Manipulator details
l1 = 10; % length of first arm
l2 = 7; % length of second arm
l3 = 5; % length of third arm

%Workspace generation,
theta1 = 0:0.025:pi;
theta2 = 0:0.025:pi/2;
theta3 = -pi/2:0.025:pi/2;

[THETA1,THETA2,THETA3] = meshgrid(theta1,theta2,theta3);

%%
%FK calculations
FKX = (l1 * cos(THETA1)) + (l2 * cos(THETA1 + THETA2)) + (l3 * cos(THETA1 + THETA2 + THETA3));
FKY = (l1 * sin(THETA1)) + (l2 * sin(THETA1 + THETA2)) + (l3 * sin(THETA1 + THETA2 + THETA3));

phi = THETA1 + THETA2 + THETA3; % Where does this get used?

%%
%create an array of all values and uniquetol it to reduce XY density in
%certain areas
fullArray = [FKX(:), FKY(:), phi(:), THETA1(:), THETA2(:), THETA3(:)];

c=fullArray(:,1:2);
[~,idx]=uniquetol(c,0.015,'ByRows',true);
sortedArray=fullArray(idx,:);


%Update variables with sorted array
FKX = sortedArray(:,1);
FKY = sortedArray(:,2);
phi = sortedArray(:,3);
THETA1 = sortedArray(:,4);
THETA2 = sortedArray(:,5);
THETA3 = sortedArray(:,6);
%%
%plot generated workspace
figure(1);
plot(FKX(:),FKY(:),'.');
title('Generated workspace')
grid on;
xlabel('x')
ylabel('y')
xlim([-25 25]);
ylim([-15 25]);
hold on;


%Join data sets and transpose
Input = [FKX(:), FKY(:), phi(:)]';

%Join the sets & transpose
Output = [THETA1(:), THETA2(:), THETA3(:)]';


%% Network setup & training
net = feedforwardnet([13 12 12],'trainlm'); %My attempt (Way more accurate than one layer)
%net = feedforwardnet([100],'trainlm'); %Paper suggested
net.divideParam.trainRatio = 0.7; % training set ratio
net.divideParam.valRatio = 0.15; % validation set ratio
net.divideParam.testRatio = 0.15; % test set ratio
net.trainParam.goal = 1e-6 ; %Set performance error goal
net.trainParam.min_grad = 1e-6/100; %Set minimum gradient
% train a neural network
net.trainParam.epochs = 4000; %Maximum number of epochs
%Train
net = train(net,Input,Output);


%% Validation tests, can rerun from here and just change testType
%straight line right side test
if testType == "rightLine"
    X = linspace(12,0,30); % x coordinates for validation
    Y(1:30) = 14; % y coordinates for validation
    PHI(1:30) = pi/3; % phi for validation
end

%straight line left side test
if testType == "leftLine"
    X = linspace(-14,3,30);
    Y(1:30) = 14;
```

```
    PHI(1:30) = pi;
end

%Curve through workspace test
if testType == "curve"
    X = linspace(15,-15,30);
    Y = 17 - abs(X.^2)/30;
    PHI = linspace(0.5,2.5,30);
end

%curve on left side side
if testType == "leftCurve"
    Y = linspace(0,15,30);
    X = -18 + abs(Y.^2)/40;
    PHI(1:30) = pi;
end

% %circle in workspace test
if testType == "circle"
    angle = linspace(0,pi,30);
    X = -15 + 1.5*cos(2*angle);
    Y = -5 + 1.5*sin(2*angle);
    PHI(1:30) = 3.6;
end

% %circle in workspace test
if testType == "square"
   X = [linspace(0,5,10) linspace(5,0,10)];
   Y(1:10) = 16;
   Y(11:20) = 18;
   X(21:25) = 0;
   X(26:30) = 5;
   Y(21:25) = linspace(16,18,5);
   Y(26:30) = linspace(16,18,5);
   PHI(1:30) = pi/2;
end


%%
%Plot test ik positions
figure(2);
plot(X(:),Y(:),'-b');
title('Comparison of target and calculated IK')
xlabel('x')
ylabel('y')
hold on;

XYPHI = [X(:) , Y(:), PHI(:)]';
test = net(XYPHI);


%%
% %Actual IK calculations, to check if reachable demand.
% a = Y - (l3*sin(PHI)); % Y of wrist
% b = X - (l3*cos(PHI)); % X of wrist
% %Get theta2
% D = ( ((a.^2)+(b.^2) - l1^2 - l2^2)/(2*l1*l2)); %Costheta2
% THETA2D = real(acos(D));
% %Get theta1
% k1 = l2*sin(THETA2D);
% k2 = l1 + l2*cos(THETA2D);
% THETA1D = atan2((a.*k2)-(k1.*b),(a.*k1)+(b.*k2));
% %get theta 3
% THETA3D = PHI - (THETA1D + THETA2D);
%Plot the analytically calculated IK X&Y
% valX = (l1 * cos(THETA1D(:))) + (l2 * cos(THETA1D(:)+ THETA2D(:))) + (l3 * cos(THETA1D(:)+THETA2D(:)+THETA3D(:)));
% valY = (l1 * sin(THETA1D(:))) + (l2 * sin(THETA1D(:)+ THETA2D(:))) + (l3 * sin(THETA1D(:)+THETA2D(:)+THETA3D(:)));
% plot(valX(:),valY(:),'-g');



%%
%test if the workspace looks alright
testX = (l1 * cos(test(1,:))) + (l2 * cos(test(1,:)+ test(2,:))) + (l3 * cos(test(1,:)+test(2,:)+test(3,:)));
testY = (l1 * sin(test(1,:))) + (l2 * sin(test(1,:)+ test(2,:))) + (l3 * sin(test(1,:)+test(2,:)+test(3,:)));
%phi = THETA1 + THETA2 + THETA3; % Where does this get used?
plot(testX(:),testY(:),'or','MarkerSize',2);
legend('Target','MLP');
hold off;


%%
%Error as a percentage of reach radius,Calc distance between target and calculated
error = sqrt((X(:)-testX(:)).^2 + (Y(:)-testY(:)).^2);
%convert to a percentage of reach radius (22)

percError = error/22 * 100;
figure(3)
plot(percError);
ylabel('Error percentage','fontsize',10)
xlabel('Comparison number','fontsize',10)
title('Error as a percentage of reach radius (22)','fontsize',10)
```