ARTIFICIAL NEURAL NETWORKS FOR LEARNING INVERSE KINEMATICS OF HUMANOID ROBOT ARMS

by
ATIF MAHBOOB

Thesis Supervisor:
Assoc. Prof. Dr. Kemalettin ERBATUR

Submitted to
Graduate School of Engineering and Natural Sciences

in Partial Fulfillment of the Requirements for the Degree of
Master of Science
in
Mechatronics Engineering

Sabanci University
Spring 2015

ARTIFICIAL NEURAL NETWORK FOR LEARNING INVERSE KINEMATICS OF HUMANOID ROBOT ARMS

APPROVED BY:

Assoc. Prof. Dr. Kemalettin Erbatur (Thesis Supervisor) .....

Prof. Dr. Mustafa Ünel .....

Assoc. Prof. Dr. Meltem Elitaş .....

Prof. Dr. Erkay Savaş .....

Asst. Prof. Dr. Murat Yeşiloğlu .....

DATE OF APPROVAL: 04/06/2015

Abstract

Keywords: *Artificial Neural Networks, Bio-inspired Learning, Infants Developments, Inverse Kinematics, Humanoid Robots, URDF Model, Humanoid Robot's Arm Design, Multilayer Perceptron, Goal Babbling, Motor Babbling*

Nowadays, many humanoid teen sized robot platforms have been developed by different research groups. The idea is either to conduct research or to produce a specific task fulfilling machine. This imposes many challenges on the design of algorithms for different actions like walk or reaching some targets. There are many sophisticated humanoid research platforms available, but one crucial aspect to look is the developmental cost associated with the task. As the name describes, the Humanoid robots are the ones that resemble humans in their design as well as their way of performing the task.

In the development of humanoid robots, many design for the arm of a humanoid robot has been studied. We have developed an arm with 5 degrees of freedom using dynamixel servo motors. We used 3D plastic printing for manufacturing the part. This arm with multiple degrees of freedom enables the robot to have free movement around the body. Besides, we also designed a simulator model of a robot that works with the advanced simulators available today.

A great number of approaches and algorithms have been implemented to solve the problem of inverse kinematics. The research carried out in this thesis takes the early learning in human infants as the basis. Human infants in their early age of development move their arm to reach new goals that they have not seen yet and with the help of the visual feedback they learn the limits and possibilities of reaching targets. We have used this idea to develop a learning algorithm that eventually enables the robot to reach goals in 3D space accurately.

This algorithm is advantageous in the sense that it is faster than the parent approach of Rolf [2013] and no prior knowledge of the arm model is required to learn the inverse solution for correct positioning. The algorithm starts with the knowledge of only one goal in the 3D space, explores more goals in the 3D space and the learning enables the algorithm to grasp the solution of inverse positioning of the arm.

The results obtained are comparable to the results generated by Rolf [2013] with the advantage that the learning is fast with our algorithm. In current research in the field of cognitive and developmental robotics, one aim is to develop robots based on biological beings (for example humans and animals) present on our planet. Humanoid robots can be considered as an exemplary development in this sense. Similarly, the researchers are trying to move the mathematically computational solutions more towards bio-inspired computational solutions. Therefore, exploring bio-inspired learning which was achieved by taking advantage of Artificial Neural Networks (ANNs) is another advantage associated with this work.

Özet

Anahtar Kelimeler: *Yapay Sinir Ağları, Bio-ilham Öğrenme, Bebek Gelişmeler, Ters Kinematik, İnsansı Robotlar, URDF Modeli, İnsansı Robot, Kol Tasarımı, Çok Katmanlı Algılayıcı, Hedef gevezelik, Motor Gevezelik.*

Son günlerde, araştırma grupları tarafından birçok genç insan boyutlarında insansı robot platformları tasarlanmıştır. Amaçları ise ya araştırma amaçlı ya da belli bir işi yapabilen makineler yapmaktı. Bu da yürüme ya da belli bir hedefeler ulaşma gibi farklı hareketlerin algoritmalarının tasarımlarında birçok zorluğa sebep olmaktadır. Birçok sofistike insansı araştırma platformları olmakla birlikte, gözden kaçırılmaması gereken önemli bir nokta da bunların geliştirme maliyetleridir. İsimlerinden de anlaşılabileceği gibi, insansı robotlar tasarım ve performansları bakımından insanlara benzerler.

İnsansı robotların tasarımlarında, birçok insansı robot kolu tasarımı incelenmiştir. Biz, dynamixel servo motor kullanan, 5 serbestlik derecesine sahip bir insansı robot kolu geliştirdik. Parçaların üretiminde, 3B plastik baskı kullandık. Birden çok serbestlik derecesine sahip olan bu kol, robotun vücudun etrafında serbestçe hareket etmesine izin verir. Ayrıca, mevcut gelişmiş simülatörler kullanarak bir robot simülatörü modeli tasarladık.

Ters kinematik sorununu çözmek için birçok yaklaşım ve algoritma uygulanmıştır. Bu tezde yapılan araştırma bebeklerdeki erken öğrenimi temel alarak yapılmıştır. Büyüme döneminin başındaki bebekler kollarını daha önce görmedikleri hedeflere ulaşmak üzere hareket ettirirler ve görsel geri bildirimler sayesinde limitleri ve olanakları ulaşırlar. Biz bu fikri kullanarak, sonunda robotun 3B ortamda tam olarak hedeflerine ulaşmasını sağlayacak bir öğrenme algoritması geliştirdik.

Bu algoritma avantajı bir bakıma Rolfün [2013] ebeveyn yaklaşımından daha hızlı olması ve doğru konumlandırma için kolun modelinin önceki bilgilerinin tersine çözümle öğrenilmesini gerek kalmamasıdır. Algoritma 3B ortamda tek amaçla başlar, 3B ortamda daha fazla amaç araştırmak ve öğrenme de algoritmanın kolun tersine konumlandırması çözümünü bulmasının sağlamaktadır.

Elde edilen sonuçlar, yazdığımız algoritmanın daha hızlı öğrenmesi avantajıyla birlikte, Rolf [2013]ün sonuçlarıyla karşılaştırılabilir. Kavramsal ve gelişimsel robotik alanındaki güncel araştırmalarda amaçlardan biri de gezegende mevcut olan (Örnek, insan ve hayvanlar için) biyolojik varlıkları temel alan robotlar geliştirmektir. Bu bakımından insansı robotlar örnek bir gelişim olarak görülebilir. Benzer bir şekilde, araştırmacılar matematiksel hesap sonuçlarını canlılardan ilham alınmış hesap çözümlerine kaydırmaya çalışmaktadırlar. Bu yüzden yapay sinir ağları sayesinde başarılan canlılardan ilham alınmış öğrenmenin araştırılması bu çalışmaya bağlı başka bir avantajdır.

Acknowledgement

The research in this thesis could not have been possible without the help, support, guidance, faith and prayers of many individuals. I "Atif Mahboob" the author of the thesis topic "Artificial Neural Network for Learning Inverse Kinematics of Humanoid Robot Arms" take this opportunity to extend my gratitude to my supervisor Assoc. Prof. Dr. Kemalettin Erbatur at Sabanci University, Istanbul for his continuous support and guidance. Without his support and guidance, I couldn't have been able to accomplish what I have done so far.

I would also like to extend my gratitude and thanks to my jury members Prof. Dr. Mustafa Ünel, Assist. Prof. Dr. Meltem Elitaş, Prof. Dr. Erkay Savaş and Assist. Prof. Dr. Murat Yeşiloğlu.

I also extend my gratitude to my supervisors Prof. Dr. Stefan Wermter and Dr. Sven Magg at the Department of Informatics, University of Hamburg, for their excellent guidance and support. The enthusiasm, encouragement and faith of my supervisors enabled me to accomplish this thesis.

I would like to thank and dedicate my thesis to my mother Zarina Mukhtar, my father Mukhtar Ahmad, my sisters Sajeela Mukhtar and Rimsha Shehzadi for their unconditional love, support, prayers and trust in me. I would also like to thank my brother Kamran Mahboob who has been always there as a mentor and a great support for me.

I also thank Mr. Amir Abbas Davari, Mr. Mert Mehmat Gulhan and Mr. Orhan Ayit for their sincere help during my work at Sabanci University. During my stay at the Department of Informatics I had the opportunity to learn from every single group member of the knowledge technology group (WTM) and excel in a friendly working environment. I would especially like to thank Dr. Cornelius Weber, Mr. Dennis Hamester, Mr. Nicolas Navarro & Mr. Johannes Twiefel for being concerned and helping me in technical difficulties during my thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we will introduce the robot platforms used in this research, the motivation and goal of this research. Later the brief introduction about human arms and Artificial Neural Networks (ANNs) will be presented.

In order to create machines with human friendly appearance and task performance, a considerable amount of research has been carried out on humanoid robots. In the past research, humanoids were produced using different non-linear control techniques and different kinds of actuators like pneumatic, hydraulic, rotational, etc. The produced robots were able to deal with specific problems which they were created for. The limitation was that once constructed, very little modification possibilities were possible.

Recent developments within the field of Artificial Intelligence have made the robots less constrained for specific tasks. Artificial Neural Networks inspired by the biological brain have introduced human brain imitated learning capabilities for robots. Nowadays, robots can learn multiple different tasks e.g. identifying objects, navigating in the environment, human interaction and many more.

The research on humanoids has the long term goal of achieving human friendly interactive robots for safe domestic usage enabling them to do regular tasks in domestic environments as well as interacting with kids and older people in the care centers.

The main idea of this thesis is to develop the arms of the robot NICU and implement a bio-inspired learner to learn the inverse kinematics associated with the arms of a Humanoid Robot. NICU is a humanoid robot that has the walking capabilities of humanoid open platform Nimbro-OP [Schwarz et al., 2012] and interaction capabilities of i-Cub [Metta et al., 2008] head. After developing this algorithm, we will focus on its generalization capabilities by applying it on Sabanci University Robotics Research Laboratory Platform (SURALP - A full human body sized Humanoid Robot, explained in chapter 5).

1.1 Motivation

Humans are very sophisticated being present on this universe. Human body contains hundreds of muscles that enable it to perform tasks for living with the help of nervous system. They need a good coordination between their degrees of freedoms and the environment in order to perform a successful action. However, it has been studied in the research on human infants that they are born without the possession of this coordination and they overcome this limitation by learning it in the early age developments as described in Konczak et al. [1997]. They first try to master reaching for a goal in space and this development further evolves in learning of very complex tasks like walking with stability, balancing with holding weights, running and many more. In modern robotics, this learning provides us with a major goal of research in the field of cogitative and developmental robotics for example Lungarella et al. [2003], Asada et al. [2009].

The main idea of this thesis is inspired from the work of Rolf [2013]. He takes the early age developments in infants [Thelen et al., 1996] to learn the way to reach a goal. Infants start with no previous knowledge about their arms and they start the exploration of space and based on the experience they get by exploration they start to learn. This learning evolves with time from random movements in the space to more precise structured movements. We use the same idea for the learning for the correct positioning of humanoid robots. The robot starts with no prior knowledge about the arm and starts exploration of the space reachable by the arm. A learner on the back of this exploration tries to learn the way in which the data is explored and the points are reached in the space.

The idea of using the learning model of infants is to use it as a basic in the initial development of humanoid robots. The way we are trying to learn the correct positioning in the 3D space is purely bio-inspired (resembles the learning involved in humans). This leads to an idea to use a learner that will also be Bio-inspired. This was a great motivation to use Artificial Neural Networks (ANNs) as the learner in our thesis. ANNs are the computational model of human brain and they try to imitate the learning of human brain [Agatonovic-Kustrin and Beresford, 2000]. ANNs requires no previous knowledge of the model or the source of the data. ANNs need a training data set for learning and they learn by updating the weights of the network based on the learning rule. ANNs after learning the solution provided with the help of the training data, try to generalize the solution of the data point outside the training data. In our work the training data will be the explored points in the space. We have used the idea of goal oriented exploration proposed by Rolf [2013] and have developed it to an approach that will be fast as well as capable of solving a positioning problem spread in 3D space.

1.2 Overview of Nimbro-OP

Nimbro-OP developed in Schwarz et al. [2012], is a ROS based open humanoid platform made by University of Bonn. The prime purpose of construction of this

platform was a soccer demonstration.

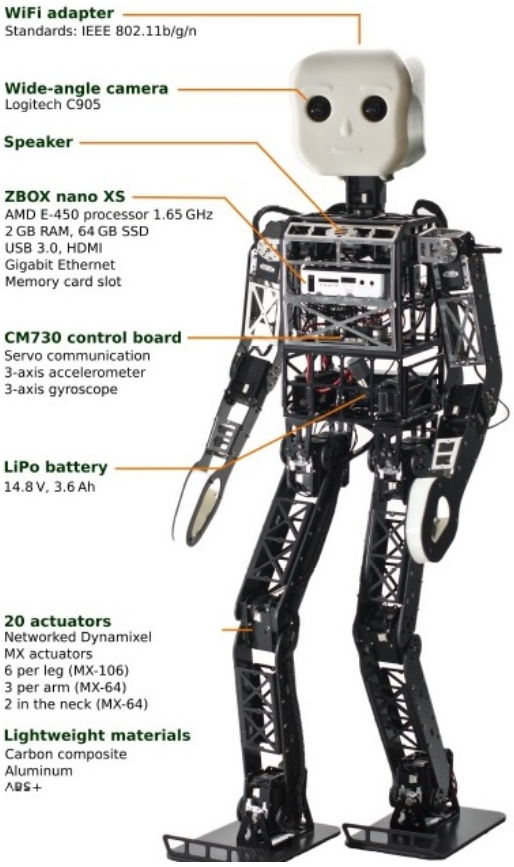


Figure 1.1: Humanoid Open Platform Nimbro-OP

The Nimbro's design is an open source platform and falls in RoboCup KidSize and TeenSize leagues. University of Bonn developed a ROS-based software framework as explained in Allgeuer et al. [2013]. This framework supports the robot by providing functionality for hardware abstraction, visual perception, and behavior generation. The robot can perform walk, ball detection, kick and recover from the fall. It also has two gait stabilization and tilt estimation to detect a fall or instability.

1.3 Thesis Goal

Nimbro can walk, play soccer, detect fall and recover from it. On one side this is a very complex and sophisticated platform as far as its walk and soccer performance are concerned. On the other side, it has a very simple design of arms. According to the design of the arms, Nimbro can use its arms for recovering from a fall on any side i.e. fall on face down, face up and side way falls. In the current configuration of the arms, the inward movement of the arms is not possible due to the extended chest. Due to this limitation, the arms cannot reach any point right

in front of the chest. Nimbro also does not possess a grip in the hand. Instead it has round tip arms. So far, the robot is not capable of demonstrating any kind of grasping or gripping.

Nimbro being constructed primarily for soccer demonstration contains only three degrees of freedom in each arm. The goal of this thesis is therefore to equip Nimbro-OP with arms that will allow it to reach multiple places in the space. The task is further divided into following sub tasks.

1. To build and test multi DOF arms that are available as Open-Hardware for 3D Printing. The arms should also possess a gripping mechanism that can allow the robot to perform grasping/gripping in the future.

2. Implement a learner for the chosen arms using a bio-inspired neural network learning approach to achieve correct positioning in 3D space.

3. Implementation of the same learner on SURALP (A full human body sized Humanoid Robot built by Sabanci University, Turkey). SURALP [Erbatur et al., 2009] will be introduced and explained in chapter 5.

1.4 Foundations

1.4.1 Arms

Humans are composed of a very sophisticated design comprised of arms, legs, eyes, ears etc. This sophisticated and well capable design has helped us to survive all the ages. Out of all these tools the arms are of paramount importance. Arms allow us to physically interact with the environment. We can do physical work like lifting things, grasping, typing etc. Other beings on the planet also have arms but in a different structure. This is shown by the fact that many animals that use their arms for walking cannot perform in the same way as we do.

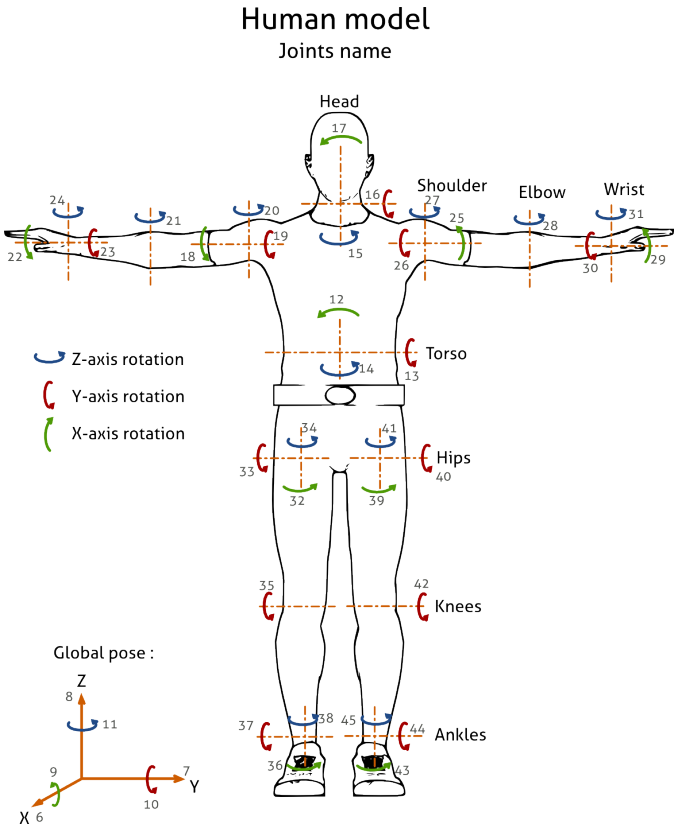


Figure 1.2: Rotations in Human Body[2]

When it comes to research, human arm design is treated as the basic design to derive further sophisticated robotic arms. The human arm has total 7 degrees of freedom i.e. Shoulder pitch, arm yaw, shoulder roll, elbow pitch, wrist pitch, wrist yaw and wrist roll.

In the arm, three of these degrees of freedom are in the shoulder, one is the elbow and three are in the wrist. In practice, three degrees of freedom in the shoulder together with one in the elbow allow humans to reach for a position successfully. The three degrees of freedom at the wrist are used for orientations.

[2]http://www.openrobots.org/morse/doc/1.2/user/code/morse.sensors.html

The hand in the human arm is equivalent to the gripper used in Robots. So the first four degrees of freedom enable the arm to move the hand to a certain position while the three degrees of freedom of the wrist are used to orient the hand in the desired orientation. The axis of rotations in the human arm as well as the rest of the body are shown in figure 1.2

In the early development of industrial robots, robotic arms were made considering the structure of a human arm due to its capabilities. This approach further helped us to develop the complex robotic arms and manipulators that are in use by different industries today.

1.4.2 Neural Networks

Artificial Neural Networks (ANNs) are inspired from neurons in the biological brain. The brain learns simple as well as complex tasks with the help of neurons. ANN uses the mathematical model of a brain neuron and is used for learning complex functions, classification problems and many more.

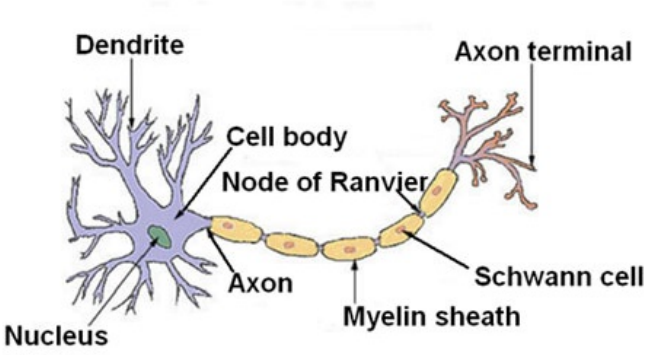


Figure 1.3: Model of a Biological Neuron

Figure 1.3 shows the model of a biological neuron. The model explained by Hebb [1949] and known as hebb's synapses established the basis of neural network algorithms. The work explains that numerous dendrites act as input channels for the electrical signal to the neuron. At a particular contact point, the electrical signal can be initiated from the synapses. The cell body of the neuron accumulates the input signals. If a particular threshold is exceeded, the cell body of a neuron generates a signal which is the output of the cell through the axon.

The model of a biological neuron in artificial intelligence is known as Perceptron and was first proposed by Rosenblatt [1958]. This model was based on the earlier model of neuron presented by McCulloch and Pitts [1943].

Each perceptron in an ANN has its own small sphere of knowledge with which to deal with. Combination of series of such perceptrons that makes an ANN that can learn ranging from simple linear to very complex functions.

Single Layer Perceptrons

In order to understand the basic working principle of a perceptron we need to have a closer look at Rosenblatt [1958]. The output of the neuron is the activation of the weighted sum of all the inputs.

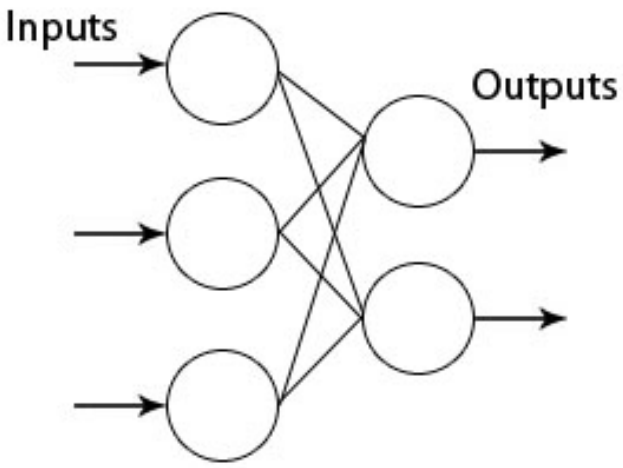


Figure 1.4: Model of Single Layer Perceptron

$$o = f(\sum_{i=1}^{N} W_i x_i) \tag{1.1}$$

Where W_i is the weight for the corresponding input and f is the activation function over the weighted sum. The activation function in case of a simple threshold function can be written as

$$f(n) = \begin{cases} -1 & \text{if } n \leq 0 \\ 1 & \text{if } n > 0 \end{cases}$$

The perceptron learns by adjusting the weights attached with each input. It changes the weights after checking the amount of error generated by the current weights in estimating the target $\hat{t}$. α is the learning rate that is a measure of how fast we want to learn the problem.

$$E = \hat{t} - o \tag{1.2}$$

$$W_i' = W_i + \alpha E x_i \tag{1.3}$$

So far these are the cases that demonstrate when there are multiple inputs and a single output. If there is more than one output then the equation will be as follows

$$o_j = f(\sum_{i=1}^{N} W_i j x_i) \tag{1.4}$$

A perceptron can learn any linearly separable function, for example AND or OR function. However, it cannot approximate a linearly inseparable function like XOR.

Multi Layer Perceptrons(MLP)

Linearly inseparable functions cannot be approximated using Single Layer Perceptron, so the idea of a Multi Layer Perceptron was proposed. Multi Layer Perceptron in comparison to Single Layer perceptron has at least one hidden layer between the output and the input layer (See Figure 1.5)

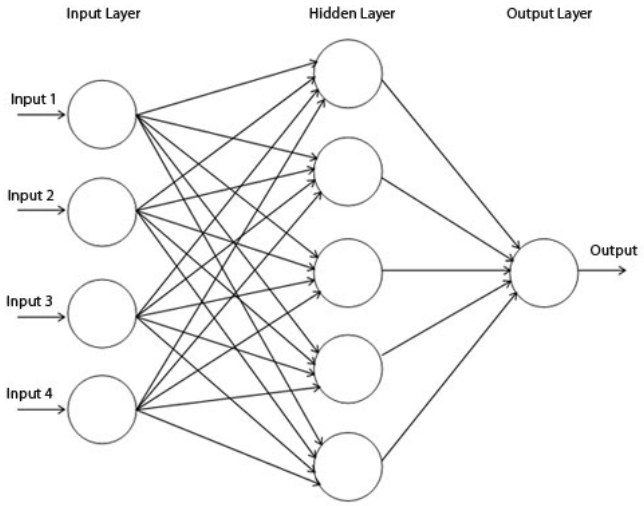


Figure 1.5: Model of Multi Layer Perceptron

The MLP network runs forward by considering the output of the nodes in the previous layer as input for the current layer. As in the figure 1.5, the hidden layer takes the output of the input layer as input and the output layer takes the output of hidden layer as input and then computes the output. The activation function here is applied on the computed values of each node.

In back propagation algorithm, the MLP learns by propagating the error backwards. That means, it moves from output towards input and updates the weights. It incorporates the derivative of the activation function for propagating backwards. So one constraint that we have is that we need to use an activation function that can be differentiated.

One commonly used transfer is the sigmoid unit

$$sgm(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{1.5}$$

This function can be easily differentiated, and the derivative is

$$\frac{dsgm(x)}{dx} = sgm(x)(1 - sgm(x)) \tag{1.6}$$

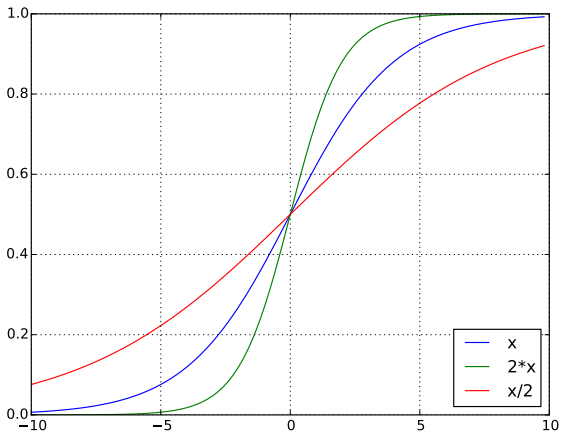


Figure 1.6: Sigmoid Activation Function

Back Propagation/Gradient Decent Algorithm

ANNs have a large area of application and many algorithms have been developed. In this thesis we will focus on "Back-propagation Algorithm" developed in Rumelhart et al. [1986] also known as "Gradient decent Algorithm" in ANNs.

Gradient decent algorithm can be used in batch learning mode or online learning mode. Batch learning mode is referred as "Batch Gradient Decent (BGD)" and Online learning mode is referred to as "Online Gradient Decent (OGD)". The understanding of the difference between these two algorithms is very important. A correct understanding will allow making the correct choice of the algorithm for any particular application. In both of these gradient decent algorithms, each algorithm iteratively tries to minimize an error function and update the parameters. In BGD, the algorithm runs over all the training samples to perform a single update of parameters in a single iteration. On the other hand, OGD only uses the one current training sample to perform the update of the parameters in a single iteration.

Batch gradient decent is a strong algorithm to approximate many of the real world problems. However, its use becomes limited while dealing with huge amounts of data sets. If the data set is huge and batch gradient is used, the learning is very slow and computationally costly. The reason for this limitation is the fact that the BGD tries to approximate the whole data set on every single iteration. On the contrary, OGD only considers the current training example it is looking at and then updates the parameter only according to one example. This makes OGD more efficient for learning problems involving huge training data sets.

Furthermore, very good comparison was made in Wilson and Martinez [2003] to provide a better understanding of batch and online gradient decent. They state that batch learning is a good approximation approach while dealing with small data sets. It deals with a smaller learning rate and a higher learning rate can make batch learning highly unstable. When it comes to large amounts of data sets, batch learning is relatively slower and is not a very effective approach.

On the other hand on-line learning is advantageous because it is fast as compared to batch learning and can deal with higher learning rates and huge data sets. On-line learning also easily avoids the need of storing weight change like it is needed in batch learning. When we compare on-line and batch gradient decent for a very small learning rate, we observe almost equivalent results. The beauty of on-line gradient decent is that it can safely use higher learning rate that makes it faster than the batch gradient decent.

As discussed above, on-line learning is better in performance than batch gradient decent when dealing with huge training sets. Furthermore, it is faster because it can work safely with higher learning rates.

This thesis goal is spread over 3D space and we may face a huge amount of training data in the 3D space. So we will use in our approach online gradient decent as explained in Bottou [1998].

Chapter 2

Related Work

2.1 Robotic Arm

The task of the arm design has some important considerations. We are looking for an arm that can allow the robot to reach different points in 3D space. It should be a low cost production and light weight arm, so that it will not put any stability danger to the robot. Our robot runs on Dynamixel servo motors and we want a design that will use the same actuators. We are developing the first prototype and we want the arm to be printed by the 3D plastic printer.

2.1.1 Seven Degrees of Freedom Robot Manipulator

In the current research for making robots capable of grasping, many arms and arm manipulators have been developed. One of them is made by Quigley et al. [2011]. This is a low cost manipulator developed for Human-scale work space. The arm has 7 degrees of freedom with a pay load of a minimum of 2 kg and repeat ability of 3 mm, with a maximum speed of 1.0 m/s.

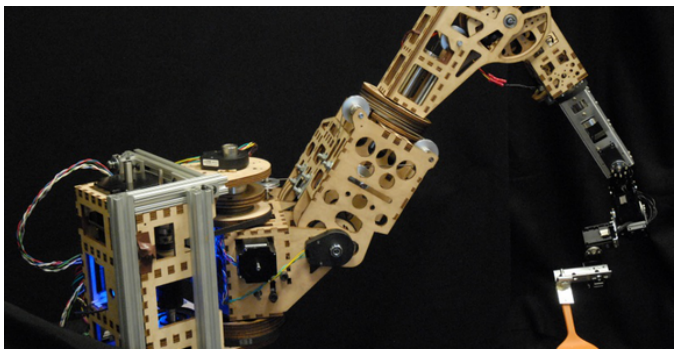


Figure 2.1: A low cost Robot Manupulator with 7 DOF

This arm uses stepper motors to drive the motion with speed reduction accomplished by timing belts and cable circuits. In particular the last three DOFs

are managed by Dynamixel robotics RX-64 servos. The arm can reach considerable space in the Cartesian plane. It can demonstrate the movements involved in playing chess and making pancakes with accuracy.

Although this arm design is very useful, its biggest limitation is that it is an arm manipulator and not a part of humanoid robot. Manipulators always have a fix mounting ground that can support their structure and make their operations stable. On the other hand, the arms made for humanoid robots need to be tested for their stable operations. The reason is absence of a fix mounting support in humanoids, instead we have a structure that is dynamic and can collapse by unstable arm movement. Another consideration is the weight and size of this arm. One can reason that the size can be reduced to fit the size of the robot, but even with this idea the mounting of this arm is similar to the existing arms of Nimbro. This mounting will not allow Nimbro to move the arms in front of the chest. Therefore, this arm design cannot be used.

2.1.2 Coginitive service robots; Dynamaid and Cosero

Nimbro-OP is a platform developed by University of Bonn which uses Dynamixel Motors made by Robotis[3] as actuators. Nimbro was actually designed for soccer demonstration. Meanwhile there are some service robots also developed by University of Bonn which are for grasping demonstration. "Dynamaid and Cosero" as stated in Schwarz et al. [2014] are two platforms designed for indoor navigation, mobile manipulation and for human robot interaction. Cosero and Dynamaid are equipped with omnidirectional wheels that provide flexible locomotion in even restricted spaces. The arms have 7 DOF and are inspired from the human arms. One difference in both robots is that Cosero is capable of lifting higher weights with a single hand as compared to Dynamaid. Both robots have 1 DOF gripper made by Festo FinGripper fingers.

These robots can perform simple household tasks using grasping as cited in Stückler et al. [2011] and Grave et al. [2010], as well as moving things in the kitchen and cleaning places inside the home. These robots were used by the team-NimbRo@Home of Rheinische Friedrich-Wilhelms-Universitat Bonn, Germany at the competition RoboCup@Home league that was held in Joao Pessoa, Brazil in July 2014.

Both of these robots have sophisticated and well performing arms. The arms are fixed on a platform with wheels which eliminate the risk of instability while performing grasping as explained in the limitations of the arm in section 2.1.1. Therefore, this design is not suitable for our application.

However, the idea of utilizing the Dynamixel servos to construct a complete 7 DOF arm can help us in our design.

[3]http://www.robotis.com/

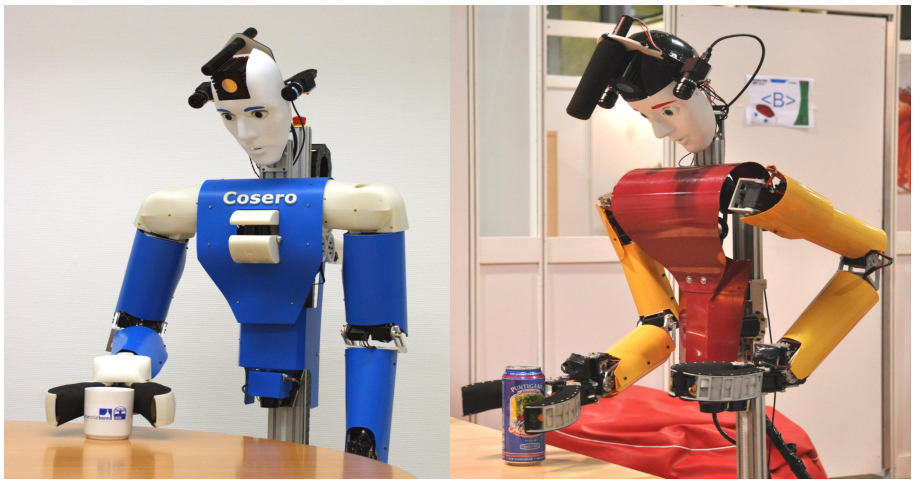


Figure 2.2: Cosero on the Left and Dynmaid [Stückler and Behnke, 2011] on the Right

2.1.3 Poppy, the open source,3D printed robot

Since the last few years the development in 3D printing technology has enabled the developers to produce low cost parts that lead to low cost robots. A part is designed using any of the mechanical design software and the designed part is provided to the printer. The 3D printer creates the exact model using plastic, metallic or polymer materials.

An excellent use of this 3D printer can be observed in Poppy as stated in Lapeyre et al. [2014]. This robot uses 25 Dynamixel Robotis actuators and all the rest of the parts are printed using plastic. As compared to the platforms of its category, the robot is low cost. The platform is open source and designed for research purposes. The robot legs can perform biped locomotion Lapeyre et al. [2013]. The bio-inspired thigh shape makes this biped walk easier. The robot can perform bio-inspired walk but so far it cannot demonstrate any kind of gripping or grasping. The arms have four degrees of freedom with a fixed human looking plastic printed hand (See figure 2.3).

Poppy is a very sophisticated and at the same time low cost robot. Its arms allow its hand to maneuver better in space as compared to Nimbro-Op. However, there is a very big limitation in the arms of this robot. It has a fixed plastic printed human looking hand. This hand improves the human friendly look of this robot and also makes its arm movements more demonstrative. As far as the grasping/gripping is considered, this arm is not usable in our application.

Figure 2.3: Poppy[4], open source, 3D printed robot

2.2 Learning Inverse Kinematics of a Robotic Arm

In the development of the humanoid robots and the advance in artificial intelligence has posed a new challenge on the robot developers to find out the efficient way to explore the joint space of a robotic arm and learn the most efficient way to reach different points in space. Many different algorithms and approaches made in past enabled the robots to learn the inverse kinematics of the arms. A small overview of these approaches will be discussed in this section.

2.2.1 Trivial Approaches

Since the start of robotic age, solving inverse kinematics is a problem of great consideration. In the initial attempts to solve the problem, the approaches were based on creating a mathematical model of the problem and solutions were computed numerically. Mathematical modeling was usually performed to get a set of equations that will fully describe the model of the arm mathematically. These

[4]https://www.poppy-project.org/

equations will link the joint space with the Cartesian space. Figure 2.4 shows an arm with three degrees of freedom.

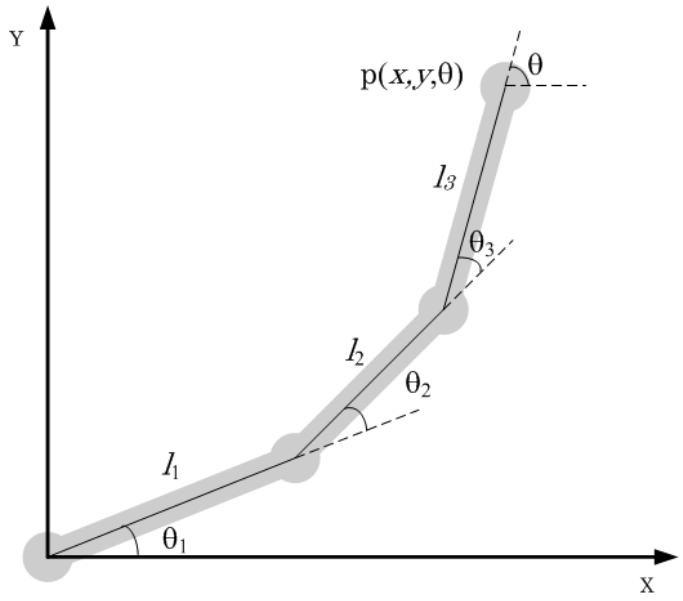


Figure 2.4: An Arm with 3 Degrees of Freedom

The equations for this arm will link the position of the end effector p(x,y) and the angle that represents the orientation with rotation angles of the joints. Correctly knowing all three angles of rotation and computing the equation will lead to the end effector position and orientation. For example the equations below represent the model of the arm in figure 2.4 and can be used to compute all three unknown values of the end effector.

$$x = I_1 \cos\theta_1 + I_2 \cos(\theta_1 + \theta_2) + I_3 \cos(\theta_1 + \theta_2 + \theta_3) \tag{2.1}$$

$$y = I_1 \sin\theta_1 + I_2 \sin(\theta_1 + \theta_2) + I_3 \sin(\theta_1 + \theta_2 + \theta_3) \tag{2.2}$$

$$\theta = \theta_1 + \theta_2 + \theta_3 \tag{2.3}$$

This problem seems easy and straightforward. However, in case of three dimensional position and orientation of an end effactor or the arms with redundancies, the number of these modeling equations multiplies and it becomes computationally very costly to compute the solution. These approaches have problems with complex systems or computationally redundant systems. Secondly, if the modeling equations can be complex it can lead to the requirement of high computation power that is simply not feasible. There can be additional factors of uncertainties that the exact mathematical model cannot be constructed or the initial conditions supposed for finding the solution are not good enough. Furthermore, a huge number of iterations are required for iterative approaches that make the approach less favorable against the modern learning approaches for inverse kinematics. A work by Berka, summarizes the traditional methods of solving inverse kinematics and explains the advantages and disadvantages associated with them.

A comparison of using Neural Networks against the traditional approaches is done by Hasan and Al-Assadi [2010]. This work discusses the closed form solution, iterative approach, numerical approach and pseudo-inverse of Jacobian to solve the inverse model. The work shows that using ANNs is better than these approaches. Also the Neural Network approach is favorable for adapting the changes made to system later after development.

2.2.2 Motor and Goal babbling

Motor babbling and goal babbling are two highly famous approaches to explore and learn the space reachable by the robotic arm. In motor babbling the motors are moved in the motor space in each iteration that causes the end effector to land at a different place each time. A considerable number of these steps can enable a robot to explore and learn the reachable space of a robotic arm. Goal babbling is a technique that uses a similar principle as motor babbling. The difference is that in goal babbling the goal (end effector) is moved in the space and the motor positions are noted. However, in motor babbling, the motor positions are moved and the final position of the end effector is noted. After a series of steps, the robotic arm can explore and learn the reachable space.

The learning works by collecting pairs of motor positions and the position of goal in space. When it comes to learning, motor babbling is not a very realistic approach due to high dimensionality present in the solution. Also due to non-linearity and the presence of redundant solutions, the data is inadequate for learning. In previous research by Rolf et al. [2010], it has been shown that goal babbling is more efficient for learning the inverse model in redundant learning as compared to motor babbling. They propose the idea of moving the goal in space that enables the system to learn the inverse kinematics while exploring. In a work by Moulin-Frier and Oudeyer [2013], four exploration strategies are discussed.

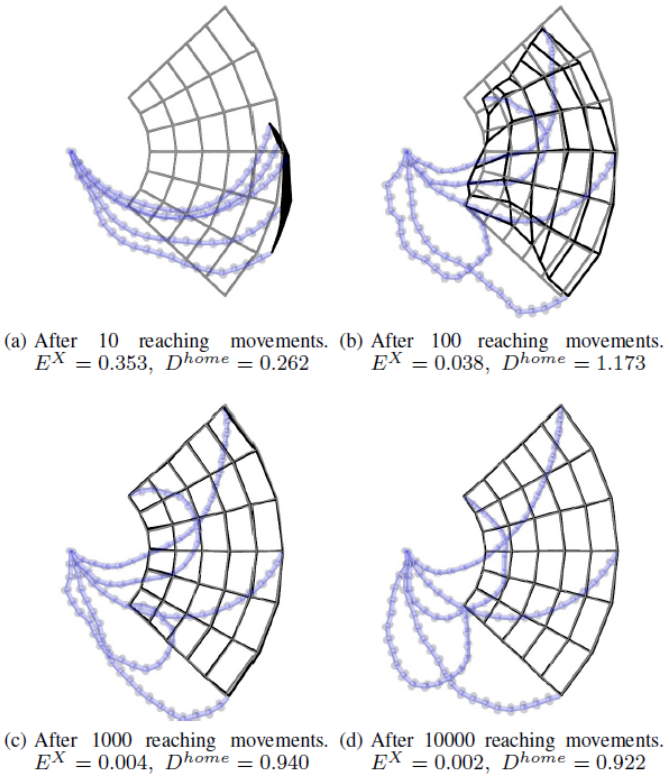
- Random Motor Babbling
- Random Goal Babbling
- Active Motor Exploration
- Active Goal Exploration

In this approach, active exploration incorporates a probabilistic model that maximizes the interest value for the exploration to be made next. The results show that goal exploration based learning performs better then motor exploration based learning. This approach also suggests that active learning is better in performance to random learning because it does more reduction in the error over space.

2.2.3 Online Goal Babbling with direction sampling

As discussed above the performance of goal babbling is better than motor babbling. Further strengthening this approach, a new idea of using online learning

for solving inverse kinematics was proposed in Rolf et al. [2010]. This approach uses the developmental idea in the early age of infants. The infants try to move the arms and then visualize and evaluate the movement to learn the inverse kinematics in the arms. The same way, another work of Rolf et al. [2011] uses online learning for learning the inverse model of the system. A path generation mechanism is used that directs the goal in a specific direction and learning is performed after every single iteration step. This approach does not use any prior knowledge about the system. Instead, the system starts the exploration and is forced to follow a generated path along a line to explore and learn the space. This work uses online goal babbling to learn the inverse estimate. This approach has been tested on a 5 DOF and 20 DOF system and figure 2.5 shows the results for 20 DOF system exploration in two dimensional space.



(a) After 10 reaching movements. $E^X = 0.353,\ D^{home} = 0.262$

(b) After 100 reaching movements. $E^X = 0.038,\ D^{home} = 1.173$

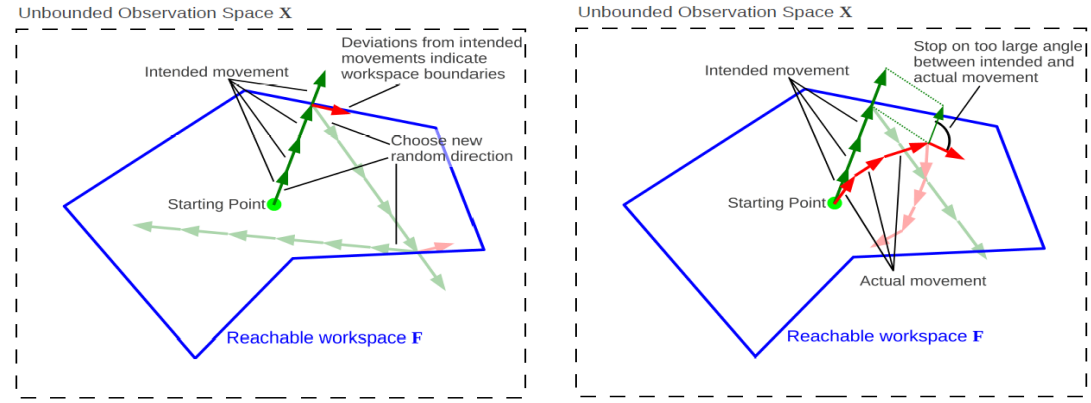
(c) After 1000 reaching movements. $E^X = 0.004,\ D^{home} = 0.940$

(d) After 10000 reaching movements. $E^X = 0.002,\ D^{home} = 0.922$

Figure 2.5: Bootstrapping dynamics for 20 degrees of freedom arm

Rolf [2013] further extends the approach developed in Rolf et al. [2011]. This approach uses the same principle of making the goal to follow a line, such that after seeing one example a small perturbation term is added in the goal position and the system is asked to reach this position. Since the system has not learned the whole space yet, it will not reach the given goal and will land somewhere else. Then the system will be trained on this example and the same process will go on until the arm reaches its boundary limit. A home function is initiated that will always return the home position for any position and configuration of the goal. The goal position will always move in a straight line but the arm cannot exactly

follow this goal line, instead it will start to bend as shown in figure 2.6.



(a) Deviations from intended movements indicate boundaries.

(b) During learning, more deviations need to be accepted.

Figure 2.6: Deviation from the intended movements

Target goal vector is calculated by taking the difference of current and last goal position and real position vector of arm is calculated by taking the difference of the arm's current and last position. When the angle between target goal vector and real position vector is 90 degree (See figure 2.6) it means that the arm has reached the end of its reachable space. At this point the system stops following the line and comes back to its home position that is generated by the home function. From the home position the system will start following another line in another direction generated by the perturbation term. The result of this approach is that after following a few lines the system learns the inverse model without the need of having a predefined set of goal position or any kind of prior knowledge. In this thesis, this algorithm will be further extended to learn the inverse kinematics of an arm with the help Artificial Neural Network as Learner.

Chapter 3

Robotic Arm Design

3.1 Introduction

The main features of Nimbro-OP have already been explained in section 1.2. This introduction will provide a further in depth knowledge about Nimbro-OP. As far as the mechanical design of Nimbro-Op is concerned, 20 actuators for 20 degrees of freedom are present in the robot. Each leg has six actuators and each arm has three actuators. Furthermore, there are two actuators in the neck. All these actuators are made by Robotis[5] and are the Dynamixel motors. The actuators on the legs are Dynamixel MX-106 and the actuators on the arm are Dynamixel MX-64. The neck also uses Dynamixel MX-64 as the actuator. Nimbro-Op is assembled using lightweight materials i.e. Carbon Composite, Aluminum and ABS+.

The robot has only three degrees of freedom per arm that are only used to recover from the fall. The robot has no capability to move the arm right next to its chest. The chest is extended and the current fixing of the arm does not allow it to move inward. The shoulder joint only has two degrees of freedom i.e. Shoulder Pitch and Shoulder Roll. One more degree of freedom is on the elbow of the robot (see figure 1.1). At the end of the arm the robot does not have a gripper, instead it has a round tip that makes it easy for the robot to recover from the fall because the probability of the arm tip to stuck somewhere is minimal. Quite clearly, these arms are not capable to perform any kind of grasping. So in the design of the arm we have several major tasks to be done as mentioned below

1. To provide more than three degrees of freedom, to build an arm that will allow more points in 3D space to be traced.

2. An arm design that should allow it to reach the points right next to the chest of the robot and that can avoid the restriction on the inward movement due to an extended chest of the robot.

3. A grip at the end of the arm that can allow the robot to grasp different objects. This is very important for physical interaction of the arm with its environment.

[5]http://www.robotis.com/xe/dynamixel_en

4. The arm should be bio inspired which means it should give a human arm like movements at performance.

5. The walk should be stable with the new arms. This can be done by keeping the weight of the newly designed arm close to the old one. The position of the center of mass of the robot will not be affect by introducing the arms.

6. The URDF model (explained in Section 3.5) should be created. This model will help in experimental simulation for the new arm design.

3.2 Initial Design Attempt

The first design attempt was made considering the fact that the modification should be as minimum as possible. The shoulder roll motor in the Nimbro is placed inside the chest. The first idea was to put this motor out and add more motors in the arms to have an initial design idea. So, the shoulder roll motor was mounted outside the chest and then the upper arm containing shoulder pitch and elbow motor was joined to this motor. The lower arm of the Nimbro with the round tip was removed and instead of that a gripper was introduced that could orient itself by the use of a wrist motor that was placed in between the elbow and the gripper. The gripper used in the structure is the gripper designed by Robotis[6] for Darwin-Op robot. The first attempt of the design as explained in this section can be viewed in the figure 3.1.

This design could avoid the problem of extended chest and also had a gripper with four degrees of freedom i.e. shoulder pitch, shoulder roll, elbow and wrist. However, it was not suitable for our application due to the following problems

- It could reach some points right next to the chest but these points were a few in quantity and the arm still restricted considerably.

- Though the arm joints were inspired from the human arm, the structure did not give a human friendly look.

- The arm was now mounted far from the chest that endangered the safe and stable walk of the robot. Moreover, mounting of the arms was not strong enough to provide the grasping torque.

3.3 New Design of the Arm

After considering the problems in the initial design attempt, it was clear that the possibility of using the previous structure of the arm to develop the new one was almost impossible. Therefore, as a result, a new design of the arm was needed. In the new design we thought about different possibilities of having different number

[6]http://www.robotis.com/

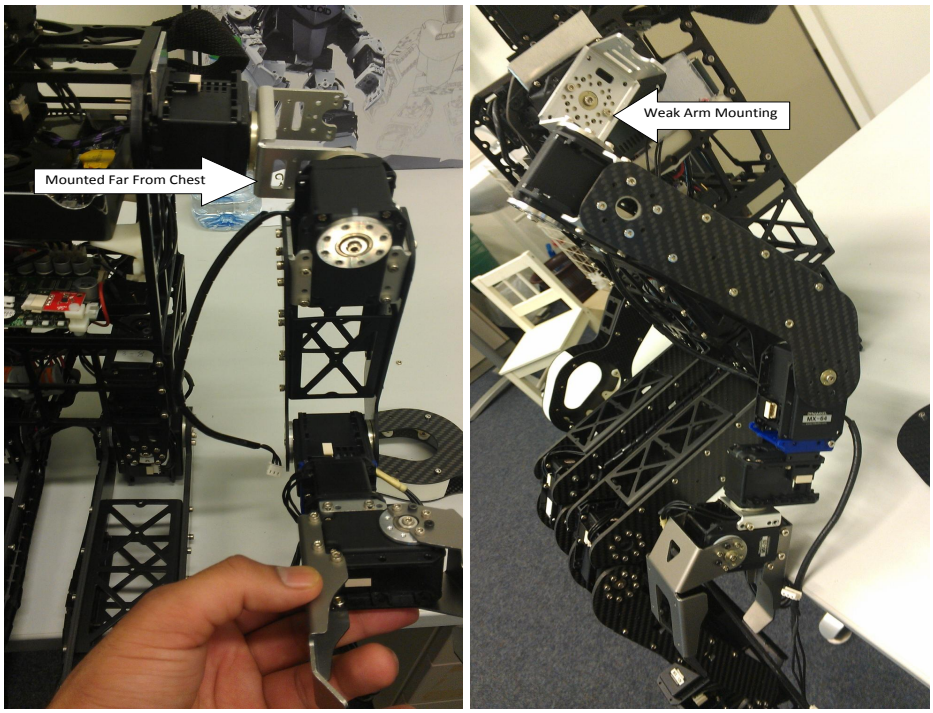


Figure 3.1: Initial Design Attempt (Left Arm)

of joints and different mounting possibilities. We decided to put 5 degrees of freedom to the new arm. The main idea was to have 3 degrees of freedom at the shoulder, exactly as the human arm has. How these joints would be placed on the shoulder was still to be decided. The fourth degree of freedom was the elbow. These four degrees of freedom were important for the movement of the arm in 3D space. The idea was inspired by the human arm that also uses these four degrees of freedom to reach different points in the space. The designed arm can be seen in figure 3.2

In order to decide the design for the placement of first four DOF, different parts were designed to check the arm structure. The main software used for the part design and the assembly was AutoDesk Inventor (Education version). The designed parts were assembled inside AutoDesk Inventor to check the final structure outcome. The design for the shoulder was very important as it would have a direct effect on the stability and the strength of the arms. This is the reason why decided to put one motor (one degree of freedom) of the shoulder inside the chest. We had some mounting possibilities available at the shoulder of the existing structure of the robot. Now, the next question was out of the 3 DOF of shoulder which DOF of the shoulder would be placed inside the chest. It should provide the strong mounting for the arm, be stable and provide unrestricted movement of the arm around the chest. Therefore, we decided to place the Arm Yaw joint inside the chest with the help of a part that would fit the motor with the existing structure of the robot. This part assembled with the motor is shown in the figure 3.3

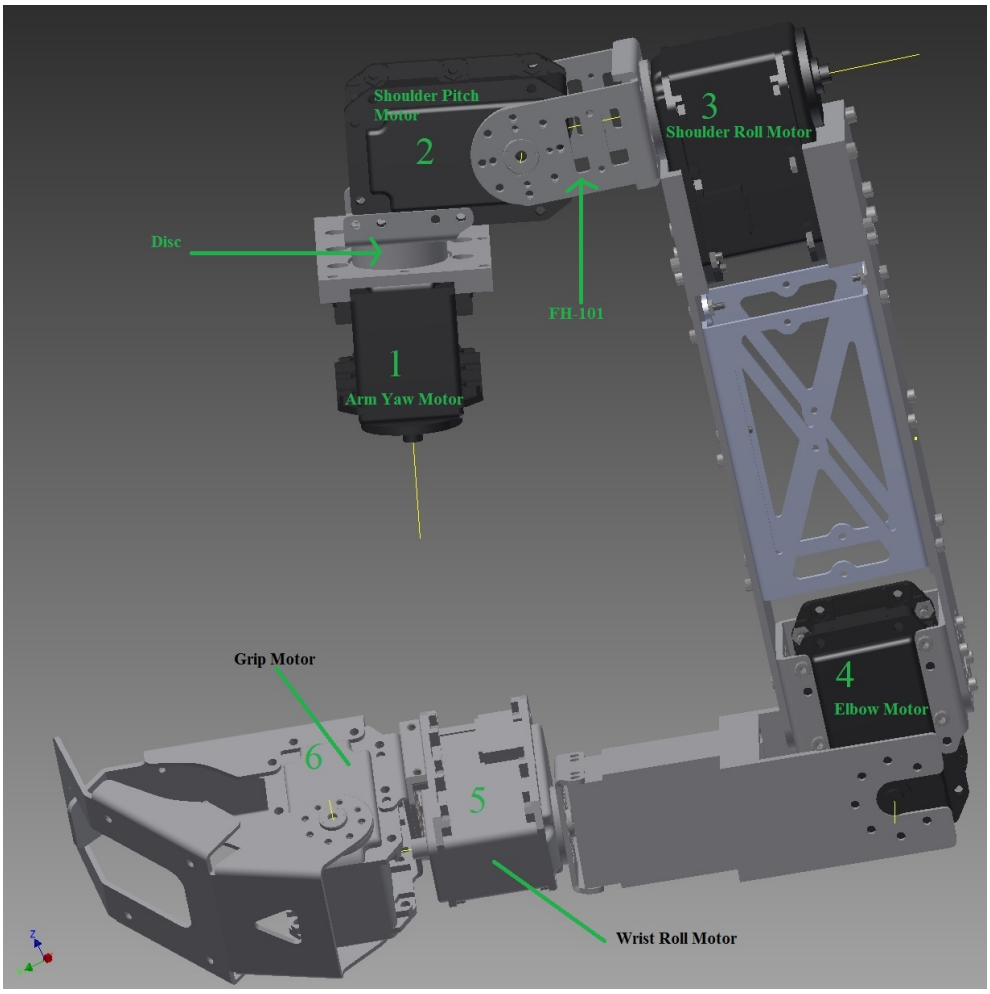


Figure 3.2: New Design of the arm (Left arm). The thin yellow lines show the axis of rotations of their respective motor

As it can be seen in the figure 3.3, a disc is placed right on the gear of the motor. The disc was needed to provide a proper mounting for the next structure that would not collide with the existing structure of the robot. This disc provides a mounting to the next structure as well as a clearance for a free rotation around the yaw joint by avoiding the collision with the mounting place.

The next motor to be mounted on was the shoulder pitch motor. This motor was mounted with the help of standard fitting available from Robotis[7]. It can be seen in figure 3.4

The next joint to be placed was the shoulder roll joint which was placed on the upper arm structure. It was not possible to mount the arm directly on the shoulder pitch motor. Again a standard fitting (FH-101(see figure 3.2)) from Robotis was used. This structure and mounting completed three degrees of freedom of the shoulder with the Torso of the robot and the upper arm of the robot. For the

[7]http://www.robotis.com/

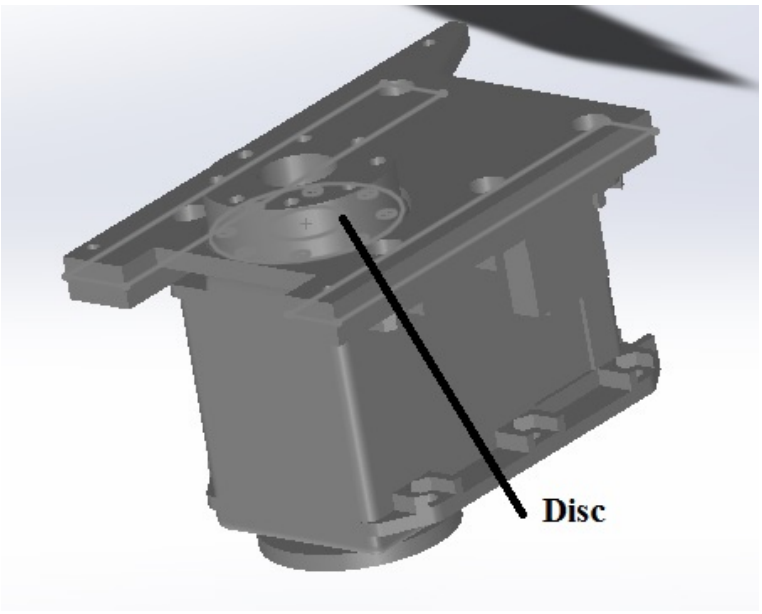


Figure 3.3: Right Arm Yaw Joint

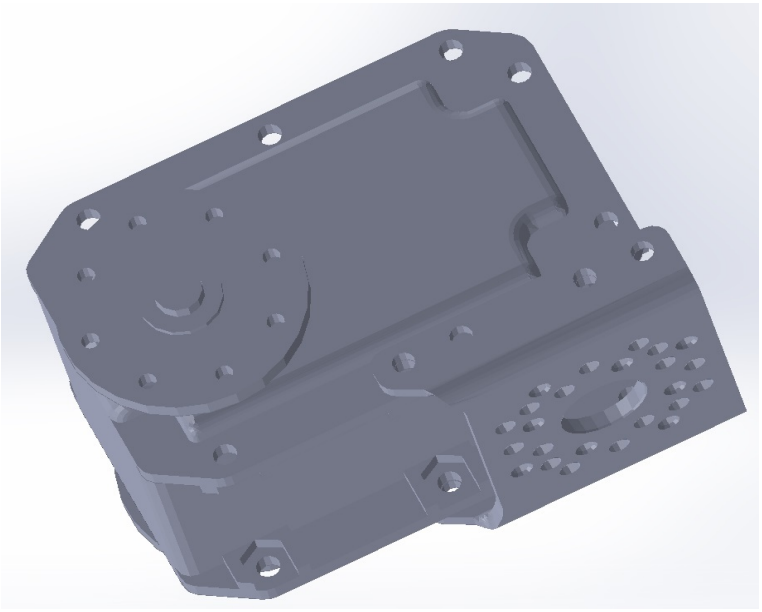


Figure 3.4: Right Arm-Shoulder Pitch

upper arm, the old parts were not usable due to the bend just before the elbow joint in the Nimbro. So the upper arm structure was also modified but the spacers of the original Nimbro arm were used. The structure of the upper arm assembly along with the elbow motor is shown in the figure 3.5

The next task was to create a design that would mount the elbow motor with the upper arm and also support the lower arm. The lower arm in the end should connect to the wrist motor. The first four motors (Arm Yaw, Shoulder Pitch,

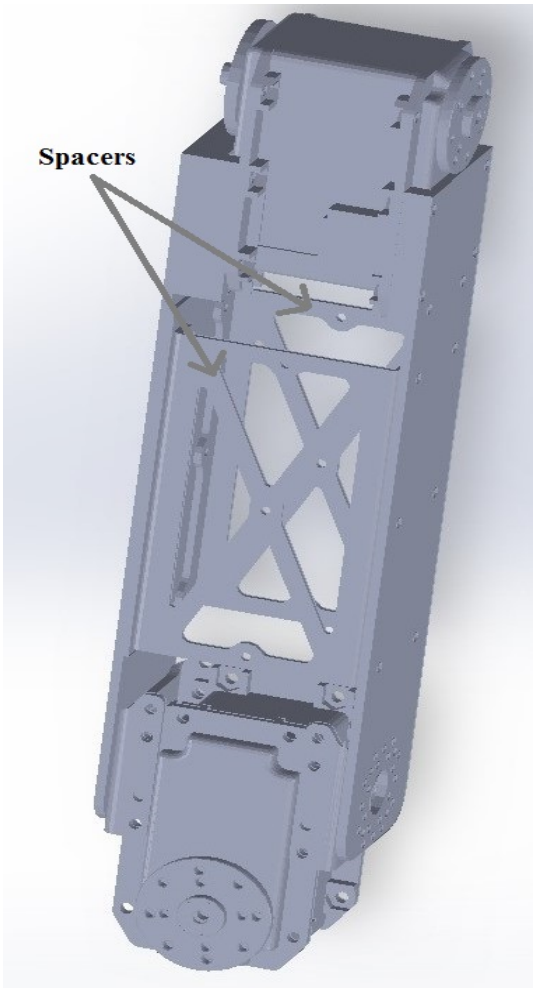


Figure 3.5: Right Upper Arm and Shoulder moutning with Torso

Shoulder roll and Elbow) are used to reach different positions with the help of the lower arm structure. However, their help in orienting the hand is minimal. The human arm has three joints at the wrist that help us to orient our hand. So we needed a structure for the lower arm that would further mount the orientation motors for the wrist, if any would be included in the structure at all. The lower arm structure is shown in the figure 3.6.

The next step was the design for the wrist of the robot. It was not possible to put three motors as present in the human arm for the orientation of the gripper i.e. Wrist Roll, Wrist Pitch and Wrist Yaw. Only one motor (wrist roll) for the orientation was added due to following considerations

- Mounting all three motors at the wrist was very difficult. Three motors at the wrist increase the weight of the arm to the level that can make the robot unstable for walk and for performing other tasks
- If we put three degrees of freedom at the wrist then we have a total of seven degrees of freedom along with one additional of the grip. Eight motors control

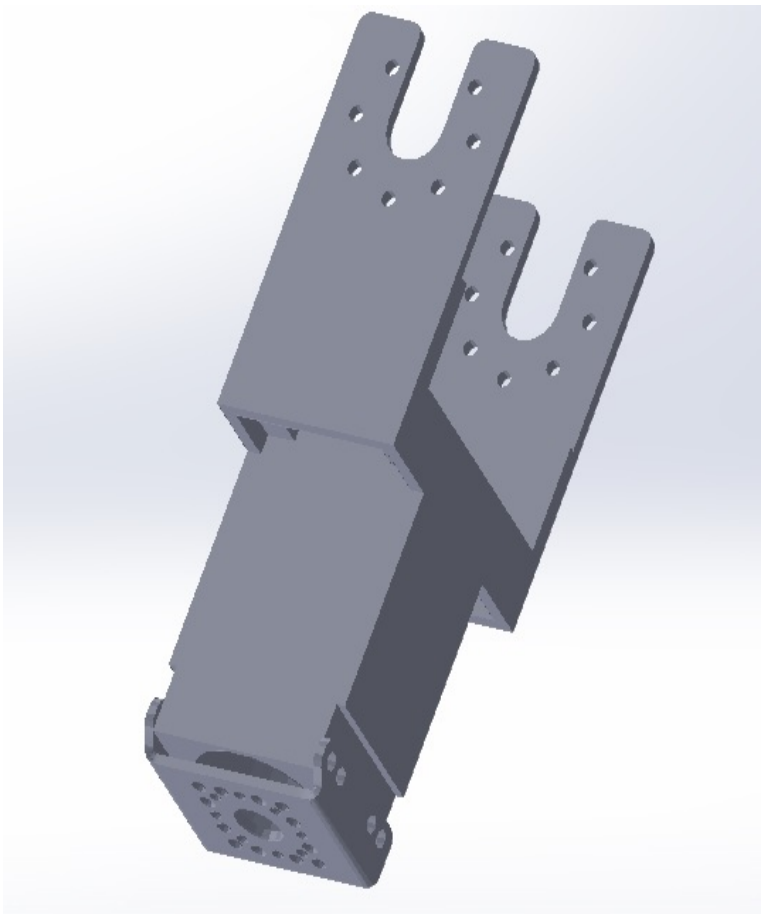


Figure 3.6: Right Lower arm

algorithm is more complex and difficult because it has more redundancies than six motor controls.

One motor for orientation can adjust the final orientation of the gripper by rolling the wrist. Finally the gripper was connected to the wrist Roll motor and we had the complete arm design. The same steps were repeated for the left arm and the same structure was obtained. The Autodest Inventor assembly for the left arm can be viewed in the figure 3.2

3.4 Printing of the Arm parts

As explained in the previous section, the parts were designed using Autodesk Inventor and the assemblies were also made using it. Once the arm design was ready inside the Inventor and was approved, the next step was to produce the solid parts and put them on the real robot. The conventional way of doing it is to produce the metallic parts or create the parts using polymers with high strength. This process can be expensive and can take a lot of machining. Latest advances in the field of 3D printing is a great relief for creating cheap parts that are just a

little bit less accurate as the ones obtained after using the machining approaches. Islam et al. [2013] investigate the dimensional accuracy of the parts created by a 3D printer. They conclude that 3D printing almost gives the same accuracy as Wire-cut discharge machining (WEDM) but the accuracy is less then CNC end milling process.

As we are in the stage of developing the first prototype of the robot, we did not need high precision. Therefore, it was convenient to print the parts using the 3D plastic printer available at the Knowledge Technology Group (WTM). The 3D plastic printer works on a gcode that is generated by the part designer using the part mesh file (STL file). This code is actually a language of instruction to the printer advising the shape, features and the information about printing layers. It also contains the information regarding printing density, speed of printing etc. The printer works using the plastic as input and uses high temperature nozzle to melt the plastic and print the part layer by layer. The layer thickness can also be defined in the gcode. The parameters density, speed and thickness are set according to the requirements and the printed parts are accurate and immediately ready to use.

The head of robot Nimbro was replaced by an iCup shaped head which was also produced using 3D printer. The iCup robot can be viewed in Metta et al. [2008]. Moreover, as one of the shoulder motor was right above the torso the length of the neck was needed to be increased. This is why the robot's neck was extended to give the head more movement span. The printed parts were assembled on the Nimbro. The final outcome after mechanical design of the arm and modification of the head was named as robot NICU. NICU can be seen in figure 3.7.

3.5 Robot Model for the Simulator

URDF is the robot definition file used by different simulators like Vrep, Rviz and Gazebo to create the exact model of the actual robot. The original model of Nimbro-Op has a URDF model that runs on Rviz and provides the real time data on the behavior of the robot that is expected after providing controller command. In the development of a prototype it is important to have the simulator model also ready as it can be very useful to conduct experiments that cannot be performed on the real robot, for example, repetitive learning tasks or checking the stability of the robot.

3.5.1 Robot Model from AutoDesk Inventor

Up to this point the designed model of the robot was assembled in Autodesk Inventor. To create the robot simulation model from inventor, multiple attempts were made. We found out that the only solution was to create the XML file from Inventor. Afterwards it could be brought to the SimMechanis plugin of MATLAB and from there it could be exported as URDF. Although many attempts were made, the model could not be created. When we created the XML file for the robot definition; it created every single part, screw or a nut as a separate STL file.

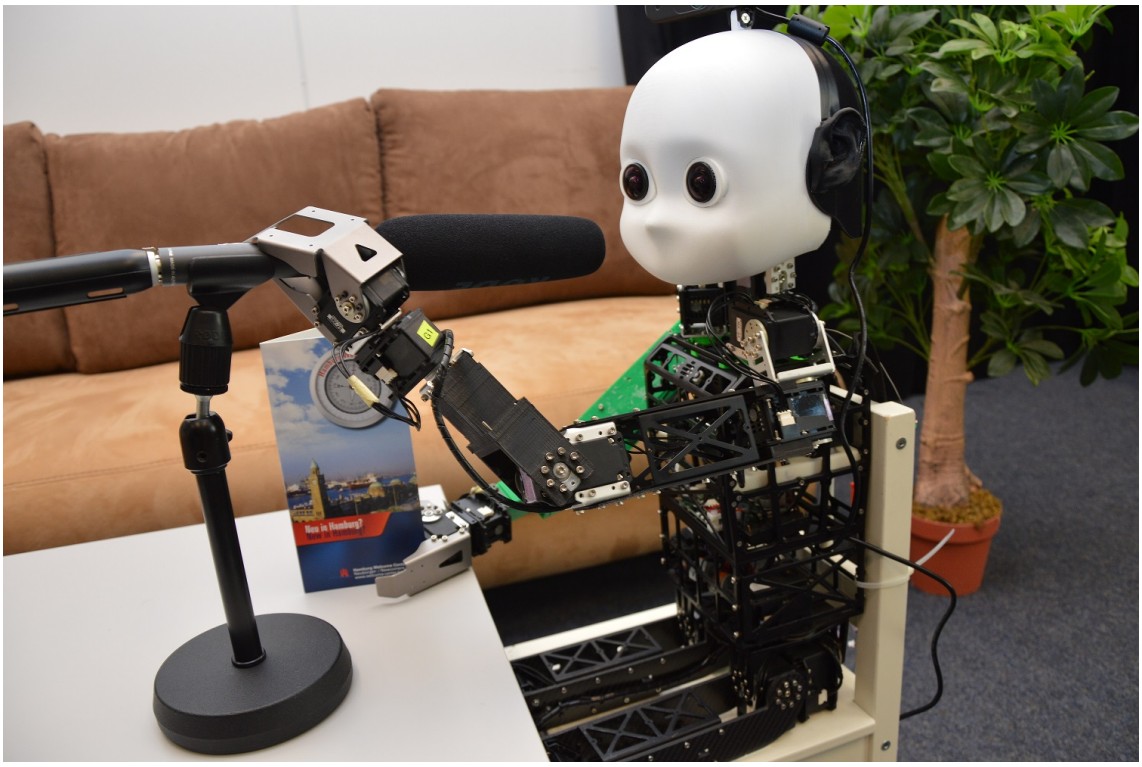


Figure 3.7: Final Assembled NICU Robot

The result was more than 200 STL files. When this XML file was imported inside V-rep, it created the robot model but it was too heavy for the simulation and also the axis of rotations of the joints were not clear to V-rep.

3.5.2 Robot Model from Solid Works

After doing further research for finding the solution to create the URDF model of the robot, it was found out that an open source Solid-Works-to-URDF plugin has been made by Stephen Brawner[8] for ROS community. The plugin works with Solid Works and exports a part or a solid works' assembly into URDF directly. The next challenge was to create the robot assembly inside Solid Works. It was tried to directly import the AutoDesk Inventor assembly in Solid Works. During the import we could only get parts and the assembly was broken due to the removal of assembly constraints. So the new assembly was made inside Solid Works. Creating the whole Solid Works assembly of NICU required a lot of time so only the torso and the arm assembly along with the head was created as shown in figure 3.8

3.5.3 Procedure for creating Robot definition File (URDF)

Now from this assembly, the URDF definition file was created that would provide us with the exact model of the torso, new arms and the head inside V-rep

[8]http://wiki.ros.org/sw_urdf_exporter

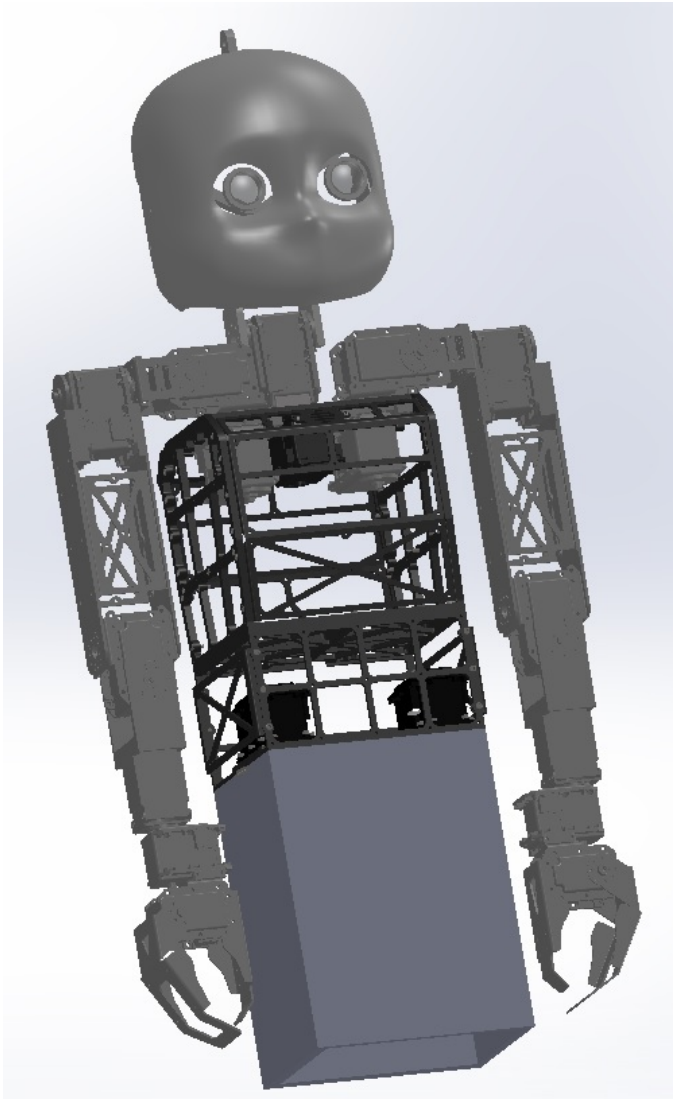


Figure 3.8: Nicu Torso Assembly - Solid Works

Simulator. There are some difficulties in following this approach since all the constraints should be at the right place and all the joint axis of rotation should be free to rotate. If these conditions are not achieved, the URDF export plugin crashes. Also in order to export the URDF model the assembly should be created in a special way. This means that the final assembly file will have the sub-assemblies that should be assembled separately. Sub-assembly is referred to here as an assembly file that will contain smaller sections of the complete arm. For example, the figures 3.3, 3.4, 3.5 & 3.6 are the sub-assemblies.

The plugin works by calculating the axis of rotation with respect to the origin of the body defined inside the plugin. Once it calculates the axis of rotation, it is extremely important that all the parts present in between two axis of rotation should be assembled in one sub assembly. This is extremely important while exporting the URDF. If two assemblies are selected as a link to an axis of rotation,

the plugin gets confused and crashes. This is why all the links in between two axis of rotation should be assembled inside one sub assembly. A combination of these sub-assemblies makes the final assembly file that will be used to export the URDF model.

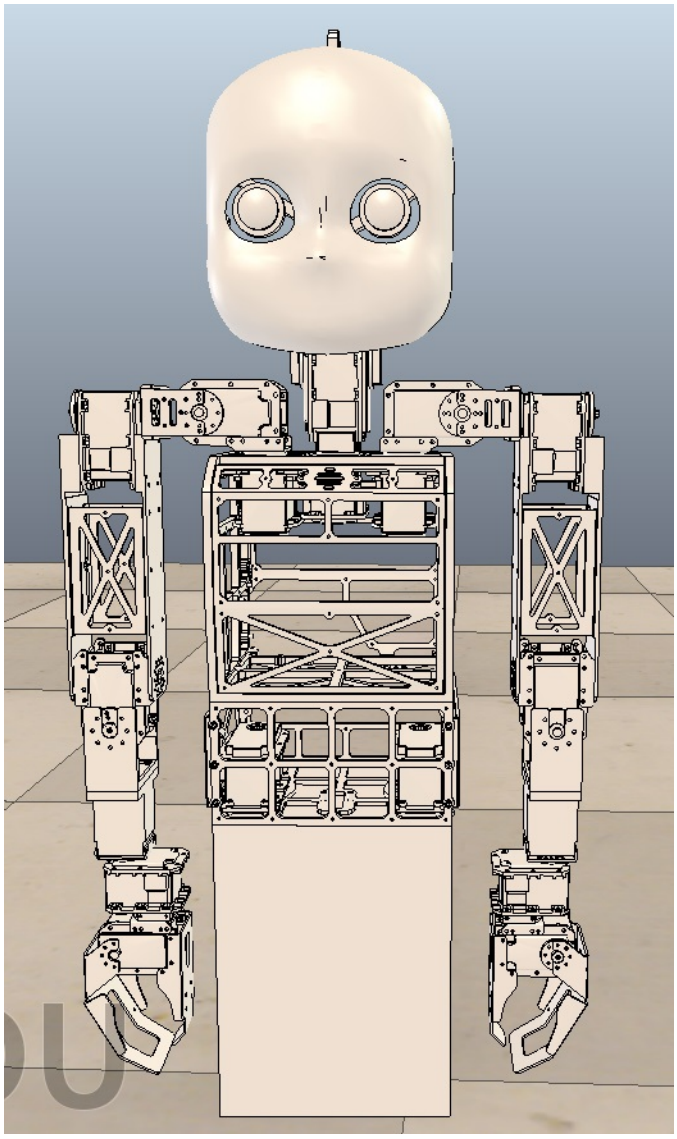


Figure 3.9: NICU Torso Model in V-rep

When we start exporting the URDF model, the plugin automatically picks all the axis of rotation from solid works. It also picks if the link to the axis of rotation is movable or fixed. All the names of the joints and the links associated with these joints are declared while requesting an export from the plugin. These will appear as the names in the robot definition file and will also be appearing inside the simulator when this URDF will be imported.

If after completing these steps the assembly is as expected by the plugin, it will produce the URDF package for the assembly or otherwise it will crash again.

In case of crash, fixing of the assembly is required and the same process will be repeated again.

3.5.4 Robot Model (URDF) Package

The final package exported by the URDF model contains the sub-assemblies that are links between the axis of rotations as mesh files (STL files). A URDF file inside the robots folder, textures and a manifest file will be created. The manifest file will enable this package to be useful inside ROS system. The URDF file that contains the definition of the robot can be directly imported inside V-rep and the V-rep scene obtained for NICU can be seen in figure 3.9.

3.6 Conclusion

This chapter discussed in depth the design of the arms of robot NICU. The approaches used and the ideas considered have been explained. Now this design will be discussed keeping in view the tasks set in section 3.1.

1. The new arm has five degrees of freedom that allow it to reach a large amount of points in 3D space. The space covered by the new arm will be shown in section 4.4.

2. The new design of the shoulder allows the arm to avoid the restriction from the extended chest of the robot and allows it to reach the points in front of the chest.

3. The gripper at the end of the arm will allow the robot to physically interact with its environment. The wrist and the grip use Robotis Dynamixel MX-28 motors.

4. The arm design is bio-inspired but with the limitation that the arm has two degrees of freedom less than the human arm.

5. To keep the robot stable, we put one motor inside the chest and one up on the chest (see design in figure 3.3 & 3.4). These will least affect the center of mass of robot as compared to the one with old arms. Also the plastic printing of the parts allowed us to have light parts and keep the weight of the new arm close to the old one.

3.7 Future Work

There are some ideas that can be considered for future improvements and are mentioned below

1. The wrist can be further extended to contain one or two more degrees of freedom and the design can be analyzed for feasibility and its stability. This extension will also provide more orientation possibilities for the gripper.

2. The most interesting idea can be to replace the gripper with a 3D printed plastic hand developed by Joel Gibbard[9]. This hand is open source and the parts are available for printing. The parts can be printed in smaller size to fit NICU and the lower arm can be modified to accommodate small actuators for the hand's finger movements.

[9]http://www.openhandproject.org/

Chapter 4

Neural Learner Design

4.1 Introduction

After having mentioned briefly the work of Rolf [2013] in section 2.2.3, this section will provide in depth details of this approach. Initially the space reachable by the learning agent is unknown. The agent starts with a position x^{home} that is the result of some action q^{home}. Until now the agent knows only the position x^{home} which is the first explored goal x_0^*. This position is used as the starting point and the agent tries to move in a randomly selected direction Δx. In the next time-step t, the goals are chosen along a direction according to the following equations

$$x_t^* = x_{t-1}^* + \frac{\epsilon x}{\parallel \Delta x \parallel} \Delta x \tag{4.1}$$

In the above equation ϵx is the distance between the current and the next explored position. Consider a situation where the agent knows how to achieve the goals in the space, it will choose correct actions for the corresponding position. In this case the observed and the desired movement will be the same, unless the desired goal is outside the reachable space of the agent. While the agent is trying to follow a line, the trend will be smooth. As it reaches its limit a sudden variation will occur that can be detected. This detection can lead to bringing the agent on the previous position and starting to follow another direction. A series of these lines will finally enable the agent to cover the whole space.

The important question here is how to detect this variation at the boundary of reachable space. As the agent has not yet learned the whole space, it will not be able to follow the line. Instead it will deviate from the line. This phenomenon can be seen in figure 2.6. In order to stop following a line, the boundary limit needs to be detected. As the agent is deviating from the desired position, a criterion on the base of the angle can be defined. The angle between the vector of the desired exploration step and the intended one will be monitored all the time. This can provide us a clue about when the boundary limit has reached by giving more than a 90° angle. This approach can be explained in terms of negative scalar product as in the following equation.

$$(x_t^* - x_{t-1}^*)^T.(x_t - x_{t-1}) < 0 \tag{4.2}$$

As soon as this condition will be detected, the agent will come back to the previous position and will start following another line. It can happen that the goal steps can be small at the start which means that the line will have more goals along the path. As the agent will learn, it will make the path through the entire space and with a comparable goal step size as expected by the algorithm.

The goal babbling algorithm tries to learn the inverse estimate of a goal g(x^*) of the forward function that is f. The model that is to be learned suggests an action $\hat{q} = g(x^*)$ for any goal that is presented. If the learning is successful then it can be said that the model is the inverse function of f. Such that it can achieve all possible outcomes

$$f(g(x^*)) = x^* \quad \forall \quad x^* \quad \in \quad \mathbf{F} \tag{4.3}$$

The algorithm starts with the initial action q^{home}. The inverse estimate g has parameter θ (motor or joint angle) to be adapted during learning. θ is initialized such that the suggested action is always the home position (see equation 4.4)

$$g(x^*\,,\,\theta_0) = q^{\text{home}} \quad \forall \quad x^* \tag{4.4}$$

Starting from this point, goals are chosen with the help of direction sampling. The algorithm tries to learn these goal. After every step of exploration a perturbation term $E_t(x_t^*)$ is added as stated in equation 4.5. This term acts as a noise and helps to explore novel outcomes.

$$q_t = g(x_t^*, \theta_t) + E_t(x_t^*) \tag{4.5}$$

The learner for the agent can be any learner. Rolf [2013] uses Locally Linear Learner (LLM) and the parameters and meta-parameters of this experimental setup are identical to Rolf et al. [2011]. The results show that the approach works well enough for 5 DOF as well as 50 DOF setup for workspace discovery. Network converges after seeing almost 1.000.000 examples of explorations.

4.2 Approach

The design approach of the neural learner for this thesis is based on the idea presented in 4.1 with some modifications. In the work mentioned above only goal babbling is used to explore the reachable space. We modified in to include motor babbling also that will help to cover the space faster then only goal babbling. If the system will cover the space faster it will also try to approximate the whole space faster. The problem is spread in 3D Cartesian space **S** with no prior knowledge about the arm of the robot. The position **P** in 3D space is a forward function f of the motor sequence **M** and the motor sequence **M** is an inverse function I of the goal Position **P**. The inverse estimate I has the motor sequence **M** to be

adapted during exploration. The initial motor sequence $\mathbf{M}^{\text{home}}$ knows the outcome position $\mathbf{P}^{\text{home}}$ and is used as the first goal in the exploration course ($I(\mathbf{P}^{\text{home}}) = \mathbf{M}^{\mathbf{home}}$). The agent is trained on this first sample of **(P,M)**. If Position **P** is always a reachable position, equation 4.3 can be rewritten as

$$f(I(\mathbf{P})) = \mathbf{P} \quad \forall \quad \mathbf{P} \in \mathbf{S} \tag{4.6}$$

A random variable is initialized that will generate a direction of exploration by choosing a random perturbation term $\boldsymbol{E}$. This term is a function of space variables x,y,z and will be added in the position **P** to get a new $\mathbf{P}^*$.

$$\mathbf{P}^* = \mathbf{P} + \boldsymbol{E}(x, y, z) \tag{4.7}$$

The new position $\mathbf{P}^*$ obtained after adding the perturbation term is unknown to the inverse estimate I as it has not seen it up to this point. Now the agent is asked to calculate the motor sequence $\mathbf{M}^*$ against the new position $\mathbf{P}^*$. As the inverse estimate has not seen this example, it will provide a wrong estimate of the motor sequence $\mathbf{M}^*$. At this point we add the perturbation term in the motor values to see another example

$$\mathbf{M}^{**} = \mathbf{M}^* + \boldsymbol{E}(m1, m2, m3, m4) \tag{4.8}$$

This sequence will lead the arm to a new position $\mathbf{P}^{**}$ along the direction of exploration. Addition of this term not only increases the exploration speed but also make it easy to detect the boundary condition at the end of a line. Now the learner will be trained on the new correctly known example as shown below

$$I(\mathbf{P}^{**}) = \mathbf{M}^{**} \tag{4.9}$$

The perturbation term will remain the same until the exploration is following a direction without hitting the boundary limit. Therefore, the boundary condition will depend on the saturation of the motor values. Once we see the saturation of all the motors in any direction (positive or negative), we will stop the line. The process will repeat several times using the same perturbation term $\boldsymbol{E}$ unless it hits the boundary of the reachable area. When this condition is reached, a new random perturbation term $\boldsymbol{E}$ is generated and added in the last position of the inverse estimate. This will allow the exploration to go to another direction within the reachable space. The choice of the perturbation term is purely random and can produce the same perturbation term as the previous one. In this case the boundary condition will be reached again and a new perturbation term will be chosen. In the experiments, Multilayer perceptron with online back propagation algorithm will be used as the estimator for the inverse model.

One important question is how the performance of the inverse estimate will be checked. For checking the performance of the inverse estimate, a test set with fixed points will be used. The test set is created in such a way that it contains the points inside the reachable space that are a minimum of 10 cm away from each other. The test set contains a total of 110 points in the reachable space. The

inverse estimate is asked to explore through a certain number of lines and then test on this test set. If the inverse model performs to certain accuracy then the exploration will be stopped, otherwise it will be asked to explore 200 lines more.

4.3 Experimental Setup

The arm design of NICU has already been explained in section 3.3. The arm has a total of five degrees of freedom including one for the orientation of the gripper. In our experiments we have not included the orientation of the explored goal. Therefore, in the learning of inverse estimate only four degrees of freedom are used (Shoulder Pitch, Shoulder Roll, Arm Yaw and Elbow see figure 3.2). These four degrees of freedom will mainly contribute to reach a certain goal in the reachable space. In order to get a real time position of the goal reached by inverse estimate, V-rep simulator is used. The robot model design for V-rep has already been explained in section 3.5 and the final scene can be viewed in figure 3.8.

Python is used as the language of programming and remoteAPI function of vrep is used to control the robot model from python console. RemoteAPI connects the program with the simulator to translate the program commands into actions. We have the goal coordinates in 3D space (x,y,z) as the input and four motor values as the output of the learner. All the joints were used in a restricted space to avoid the collisions with the robot body. The arm yaw motor (motor 1 in figure 3.2) had 60° , shoulder pitch(motor 2) 20°, shoulder roll(motor 3) and elbow(motor 4) had 90° allowed rotations. The zero positions for the joints can be viewed in figures 3.7 and 3.8.

In the approach by Reinhart and Rolf [2013], a home function is initialized that will always result in the home position posture. This means that after completing a line, the system will always start from the home position. We modified this approach in Artificial Neural Network such that the network was largely over fitted on the home position. This over-fitting gives two advantages. First, we no longer have to worry about the weight initialization for the network. It does not matter what are the initial weights. The reason is that when the system over-fits a single position, it always has the weights that exactly produce that position. So this approach solved the problem of hyper parameter optimization for network weights initialization. Secondly, we implemented an alternative to the home function approach of the above author.

We are using multilayer perceptron (MLP) as a learner with the help of online back-propagation. Our network has biases and does not have the momentum term. The sigmoid activation function (also see figure 1.6) is used on the hidden and output layer of the network for activation. The network architecture used for the experiments can be seen in figure 4.1.

As we are dealing with online learning, it is possible that the system can forget the goals that it has already learned. So we modified the testing of the performance of the inverse estimate. One testing set we have as the fixed testing set spread over the whole space, as explained in section 4.2. We created another test set that will

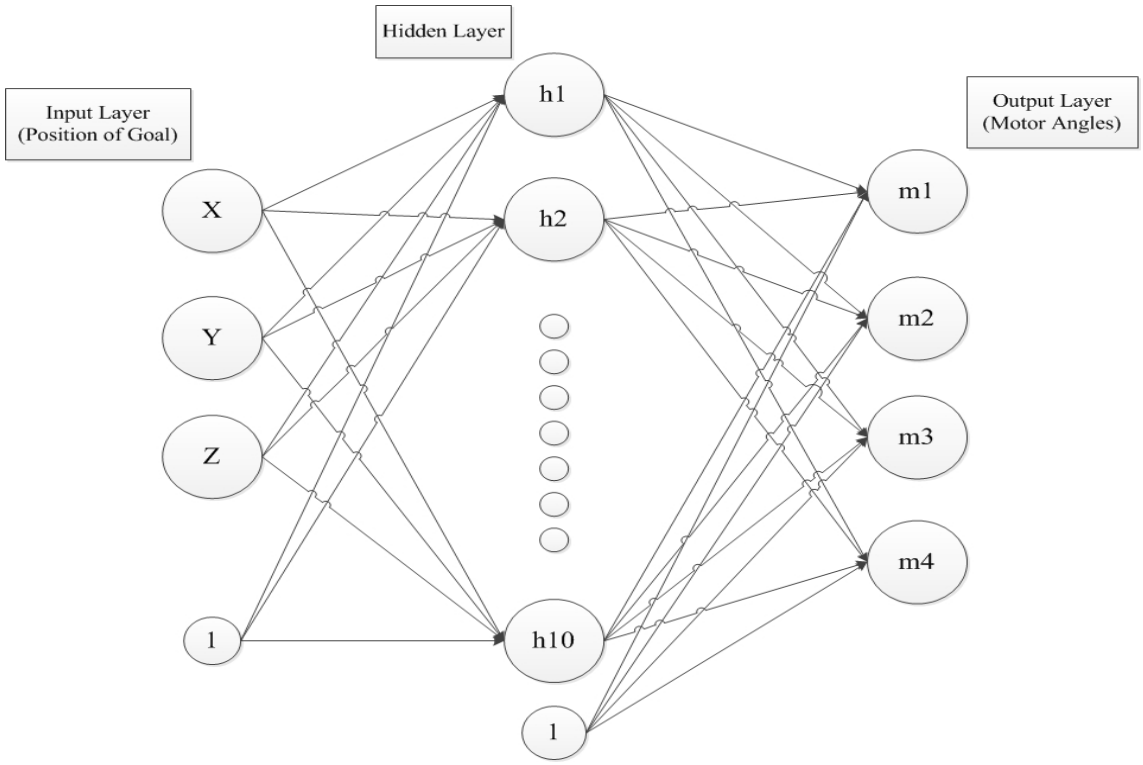


Figure 4.1: Artifical Neural Network Architecture for the learner

have the same number of test points as in the fixed testing set i.e 110. However, this set will be chosen randomly from the goal positions that are already seen. In every testing cycle, the points for the seen test set will be chosen randomly. This means that in every testing after exploration of 200 lines we will have different points for testing in the seen test set. This set will keep a check on the system performance regarding the already seen points and will indicate if in a new exploration, the system forgets the old explored points.

4.4 Results

In this section, we will discuss the results we obtained from our experiments. The accurate and effective use of Artificial Neural Network requires that the best parameters should be chosen for the problem under consideration. These parameters can largely vary from problem to problem and also differ for different approaches. The hyper parameters in our consideration were the Learning Rate α, Perturbation term $\boldsymbol{E}$, Number of nodes in hidden layer and the initialization of network weights. As explained in section 4.3, the initial weights for the experiments were made almost the same by largely over-fitting the network on the home position.

4.4.1 Hyperparameter Optimization

These parameters are normally identified by conducting experiments with different parametric values under same experimental circumstances. The problem in our approach is that the exploration is random and it can vary from one experiment to the other. This actually means that our training set cannot be same for all experiments. The iteration for following the lines can also vary in each case. If we use 100,000 goal lines exploration, the experiment takes about 10-12 hours. In the parameter exploration we have to test a lot of different parameters, so 10-12 hours time for one experiment is not a very realistic approach. We decided to fix the number of iterations for each experiment to 50,000 and after every 2000 goals (Iterations) the testing will be performed. As the goal exploration was totally random, there was no accurate way to compare it for different experiments. In fact,we also want to explore the phenomenon of how different networks will explore the space in similar circumstances. Therefore, we only varied the three parameters α, $\boldsymbol{E}$ and hidden nodes. The testings that occur after the exploration of 2,000 goals were considered to observe the final results. Once the network was a bit settled, the mean and the standard deviation were calculated for these tests. The heat-map for these three parameters can be seen in figure 4.2.

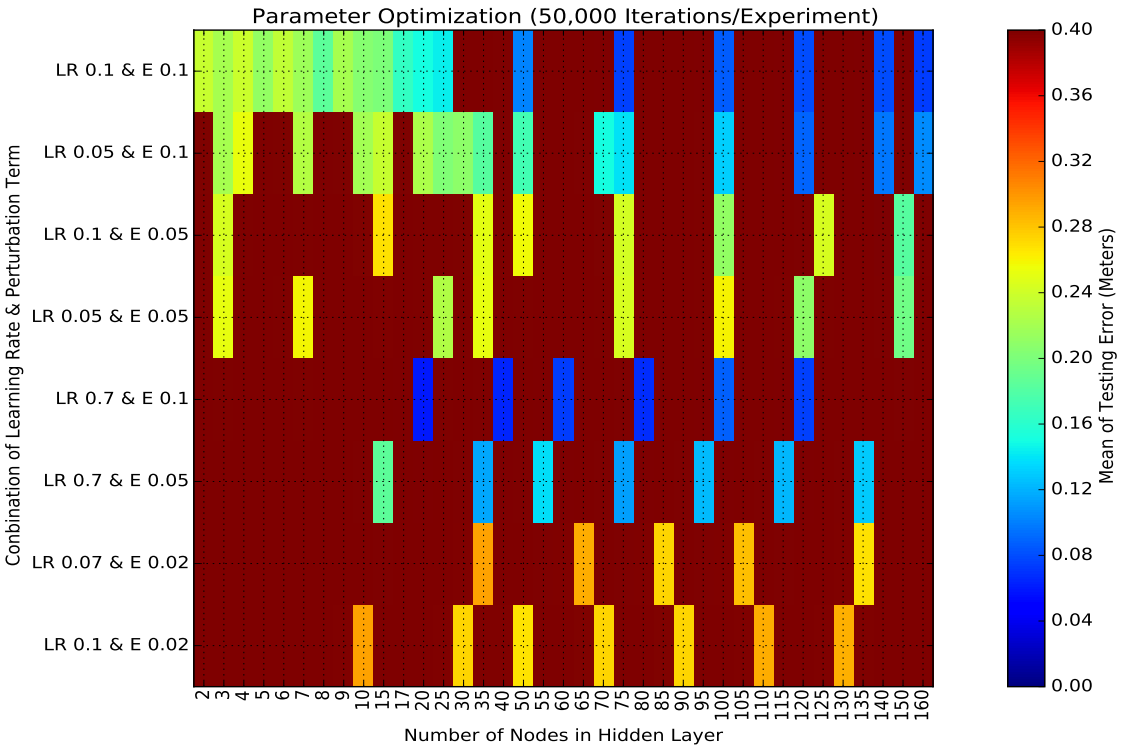


Figure 4.2: The plot shows the result of random parameter exploration. The maximum mean error is about 0.3 meters. The heat values in the map corresponds to the mean of average positioning error over the testing cycles. The dark red part shows the region for which the experiments were not performed

Hidden Nodes	LR0.1andE0.1	LR0.7andE0.1
75	0.075518	–
100	0.086544	–
160	0.074059	–
25	–	0.059752
50	–	0.065433

Table 4.1: Table shows the best results (lowest average positioning error) for experiments represented in figure 4.2. The values in the table are the mean of average positioning error over the testing cycles. The complete values of these experiments can be viewed in Appendix C in table C.1 to C.8

Rolf [2013] suggest the Learning rate to be 0.1 with a perturbation of 2 %. We used parameters similiar to the ones suggested in the mentioned paper. The results we obtained show that reducing the Learning rate for the same Perturbation term reduces the quality of results. Also reducing the perturbation term for the same learning rate depicts the same outcome.

In our exploration we also tested the Learning rate $\alpha = 0.7$. This gave surprisingly good results in terms of reduction in error. In fact, these were the best results obtained in these experiments. This opened another door to check high learning rates and high values for the perturbation term. These results sparked another thought of the idea of Wilson and Martinez [2003] explained in section 1.4.2 which states that online gradient decent algorithm can handle high learning rates and perform well. The best results obtained from these initial experiments can be seen in table 4.1.

Initial experiments were done using totally random parameters without any uniform variation. The networks with 2 to 3 nodes were also trying to learn which means that the function linking input with output can be very simple. Therefore, in the next experiments a further detailed and systematic variation in the parameters was done. It was also a matter of interest to check the results for linear perceptron as it can indicate if underlying functions are linear. The results for this systematic and detailed exploration can be viewed in the heat-map in figure 4.3

In these experiments the numbers of nodes were kept same for all combinations of learning rate and perturbation. This allows us to observe the effect of variations more precisely and easily. The best results for these experiments are mentioned in below table 4.2 and mean & standard deviation plot for the best case can be seen in figure 4.4.

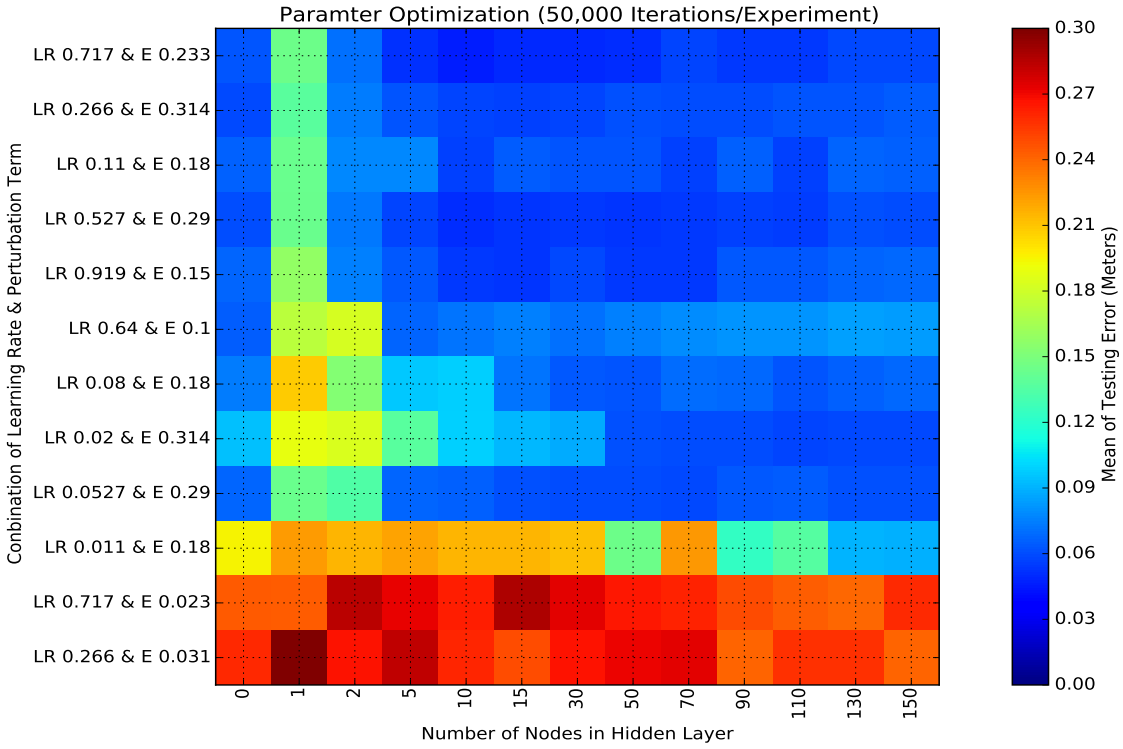


Figure 4.3: The plot shows the result of parameter exploration for higher learning rate and higher perturbation values than 4.2. The heat values in the map are the mean of average positioning error over testing cycles.

Hidden Nodes	LR0.717&E0.233	LR0.266&E0.314	LR0.527&E0.0.29	LR0.919&E0.15
10	0.046847	–	0.051050	–
15	–	–	–	0.053417
0	0.062618	0.059682	–	–

Table 4.2: Table shows the best results (with lowest positioning errors) for experiments represented in figure 4.3. The values in the table are the mean of average positioning error over testing cycles. The complete values of these experiments can be viewed in Appendix C in table C.9 to C.20

Furthermore, in order to further explore the range of best parameters we conducted more experiments. In these experiments we used the parameters close to the best case shown in Table 4.2. The obtained results can be seen in the form of a heat map in figure 4.5 and the accurate values can be seen in Appendix C in Table C.21.

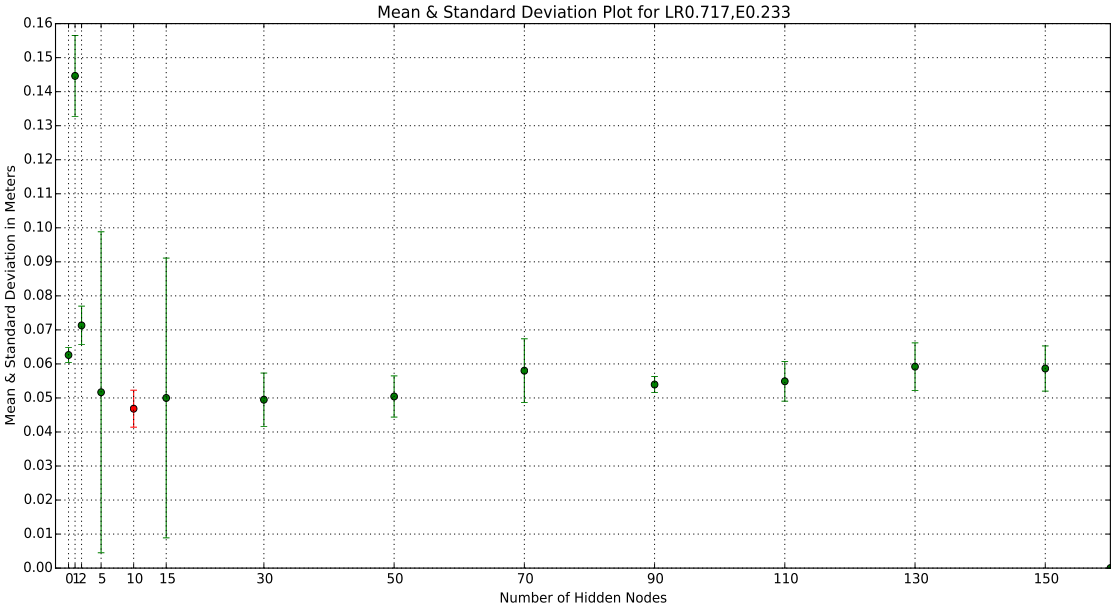


Figure 4.4: Mean & Standard Deviation of average positioning error (over testing cycles for best case of LR0.717 & E0.233). For optimal number of nodes the best result (against 10 Hidden Nodes) obtained can be viewed in red color. These experiments shows the exploration results for 50,000 iteration each.

The results in figure 4.5 show that 0.35 perturbation works faster and it is true. However, the problem is that we are only testing for 50,000 iterations and when we ran the experiments with full length of 100,000 lines exploration final error is more than the 0.25 perturbation values. The phenomenon here is that the values of perturbation above 0.25 quickly reduces the error over the whole space due to covering more space in exploration in less time. The behavior after 50,000 iterations is almost consistent as compared to reduction in error. We need a detailed exploration to further reduce the error on space which is difficult with the high perturbation term such as 0.3-0.35. Furthermore, if we are using such high perturbation values it may happen that the region boundaries are not fully explored. We therefore propose that the **10** hidden nodes along with Learning Rate **0.7-0.9** and perturbation **0.15-0.25** can be used as the optimal parameter region.

4.4.2 MLP With Best Results (LR0.717 & E0.233 &10 Hidden Nodes)

At the beginning, we tried to work with the parameters suggested by the experimental setup of Rolf [2013]. We worked with the learning rate of 0.1 and the perturbation term E of 0.1. With this combination of LR and E, we tried different number of nodes in the hidden layer. The initial experiment gave us a lead to also

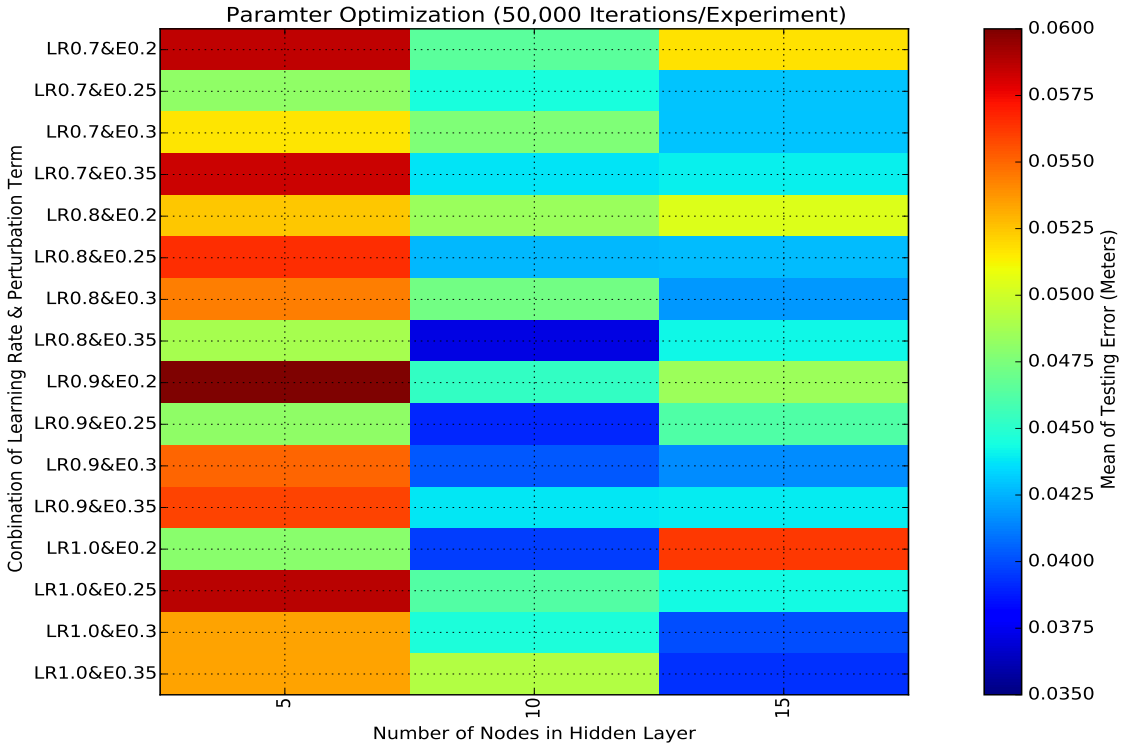


Figure 4.5: The plot shows the result of parameter exploration for parameters close to the best case (LR0.717 & E0.233 & 10 Hidden Nodes). The heat values in the map are the mean of positioning error over testing cycles.

check the higher learning rates. Another set of performed experiments gave better results than the initial ones. Here we will show the experiment that provided the best results under limited iteration numbers of 50,000. We reran this experiment to explore 100,000 lines in the reachable space. After every 200 line exploration, the testing was performed on seen testing points set as well as the fixed point testing set. The variation in the testing error for seen and fixed testing points for the experiment with the best results can be seen in figure 4.6.

It can be seen from the figure that the error over already seen points is always a bit less than the fixed testing points. The reason is that the fixed data set contains the points over the complete area that may not be explored by the algorithm and the algorithm had to approximate these points considering its knowledge. Moreover, we can see noise in the graph. The noise comes due to the random exploration. One testing is performed after training the Neural Network over 200 explored lines. If this exploration is not uniform over the whole reachable space and is focused in a small region of the reachable space then the network gets biased. This biased network cannot approximate the whole space with the same accuracy as it was doing in the previous test. We also observe that once the network gets biased, the next exploration can be relatively unbiased and it tries to counter the effect of network biasness. Therefore, we observe the cycles of biased exploration and counter biased exploration. It is also important to see the space explored by

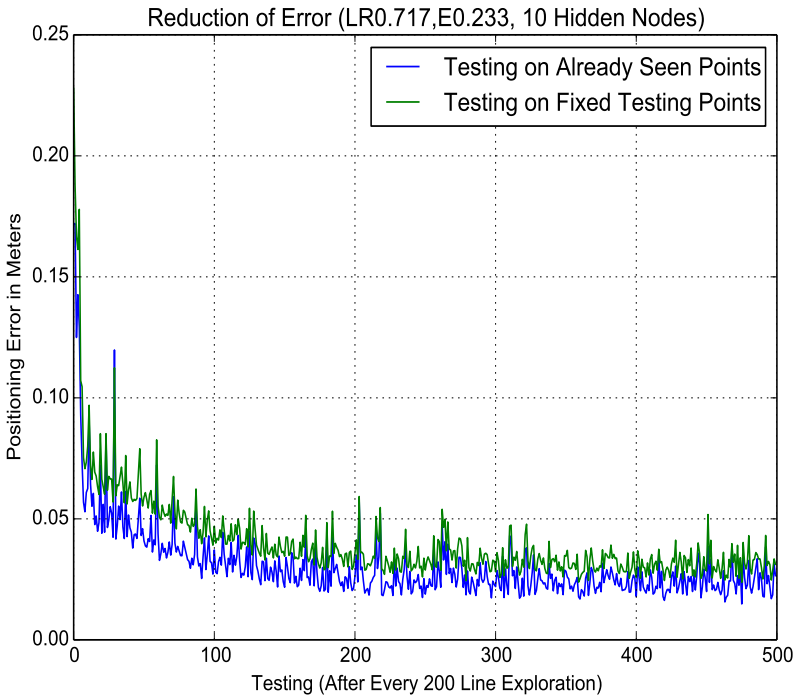


Figure 4.6: Average testing Error over Seen and Fixed Testing Sets

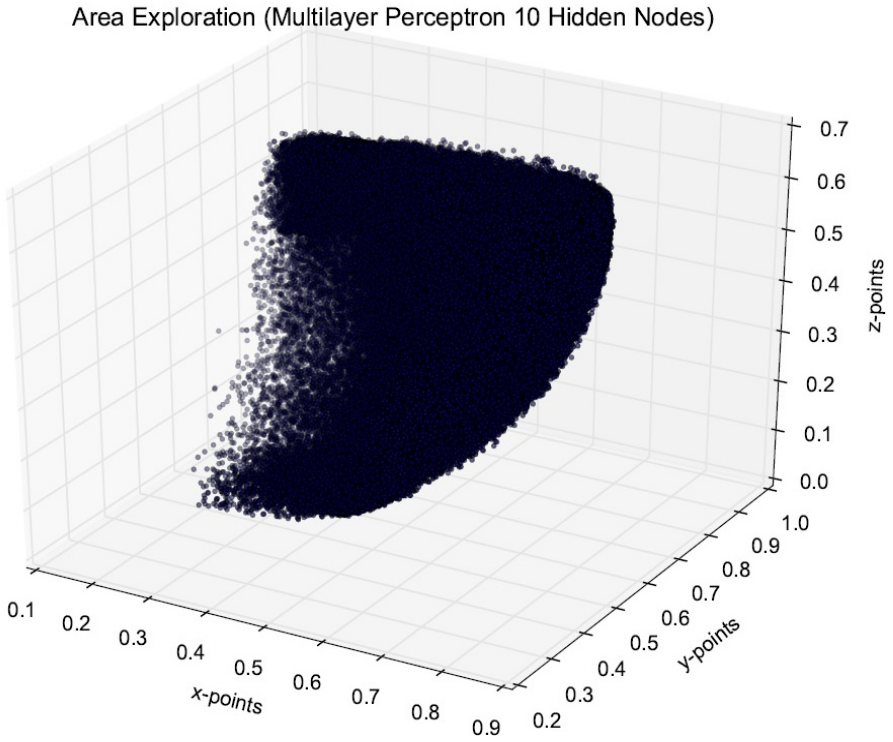


Figure 4.7: Area Explored by the MLP (LR0.717, E0.233 & 10 Hidden Nodes). The robot origion is at (x,y,z = 0.56,0.35,0.31)

this experiment. While running the experiment the data for exploration was also recorded and the visualization of the normalized explored area can be observed in figure 4.7.

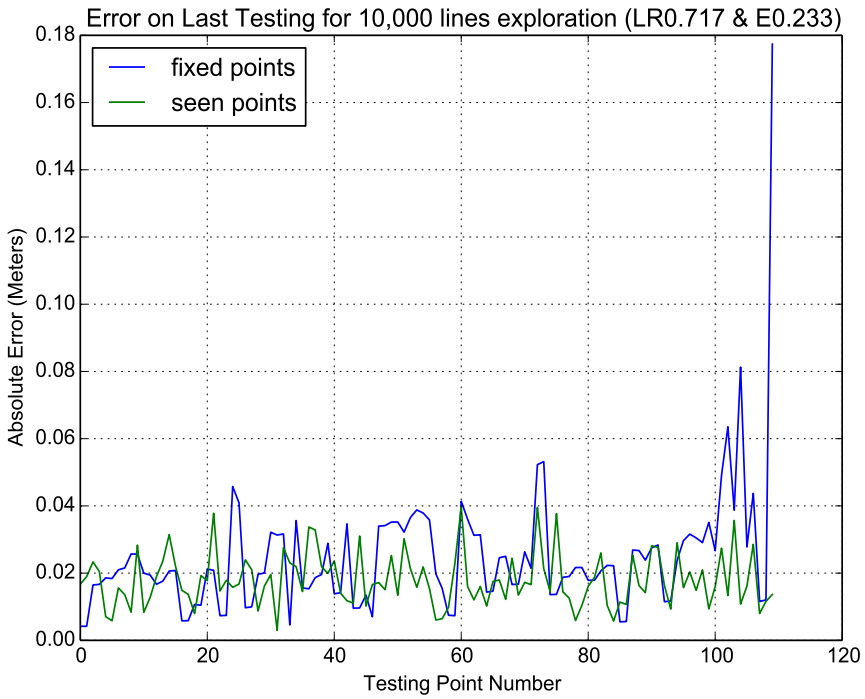


Figure 4.8: Error of each points in testing cycle for which average error over fixed testing set is 0.024745 & for seen testing set is 0.017962 meters. Absoulte error here is the positoning error of every single point in the corrosponding testing set

Another important thing here is to observe the error values for the last testing points for best case (10 nodes, LR0.717 & E0.233). Figure 4.8 shows some high peaks of error for some points in the fixed test set. The reason for these peaks is the fact that this test set was created from a highly dense exploration that was done using very small perturbation value. This means that we almost have every single point on the reachable space. It also included points that are very less likely to be reached by the arm for example the points may be explored once or twice. It also contained points inside the robot body because we did not use the collision detection in our experiments. There is one point in the fixed testing set that is not in the reachable space; therefore it always shows a constant error of around 15-18cm. Furthermore, the points on the extreme ends of the reachable area are difficult to reach. This is quite similar for us (humans) also. It is easy for us to reach the points right next to us and it gets difficult to take the arm behind the back or at its corner limits. In fact to avoid this situation we always try to place our body around the objects such that they will be right in front of us and will be easy to reach. So if we exclude these points that are difficult to reach and consider the concerned area right next to the robot chest (only green points in figure C.1)

we have an average error of **1.88 cm**. In order to confirm this justification, we have included a visual for understanding of this phenomenon. It can be viewed in the figure C.1. If we see the error for seen points the results are much nicer. The error fluctuates around 2cm and in few points it goes above 3cm. Again these points are located either at the boundary of reachable space or quite close to the robot body. So if we only see the average error over green point (see figure C.2), we observe an average error of **1.73 cm**. We have also included to show a visual for this phenomenon in figure C.2. The green points have error less than 3cm and the red points have more than 3cm. The blue cross shows the center of thr robot body.

4.4.3 Single Layer Perceptron

Another interesting result in table 4.2 is with the linear perceptron. Linear perceptron also tries to approximate the whole space but the final error observed is a bit higher than with the multilayer perceptron. Running an experiment to 50,000 iterations can tell about how fast the network is approximating. However, it is difficult to address the future behavior with this experiment. Therefore we also reran this experiment to explore 100,000 lines similar to the one we did in the MLP case. The results of Linear Perceptron experiment for the same learning rate and perturbation term can be seen in figure 4.9. It can be seen from the figure that

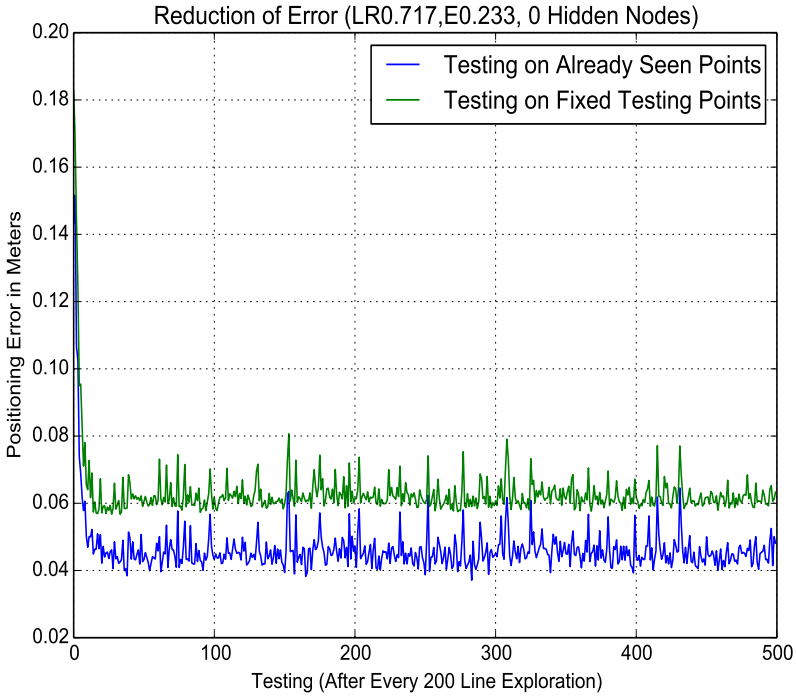


Figure 4.9: Average Testing Error over Seen and Fixed Testing sets For Single Layer Perceptron

Single Layer Perceptron(SLP) quickly reduces the error to 6 cm and then stays there. The SLP can be considered a good case but our long term goal is to get an accuracy of around 2 cm and it seems difficult. Due to the fact that SLP cannot map the complex non linearity in the data as good as the MLP. For example the region where the elbow has a higher bend angle, the SLP performance is not good. SLP works well on some areas but relatively bad in others. Overall it is not as good as the MLP for the approximation of the whole region. In the next section we will also show that MLP takes more iterations for the same line exploration as compared to SLP which enables it to cover more points in space. However, the common thing that was observed in the results is that the error reduction for seen points is more than fixed testing points, similar to the MLP. The area explored by the SLP can be seen in figure 4.10. If we compare this exploration with the exploration of MLP, we see some regions not covered as good as the MLP do.

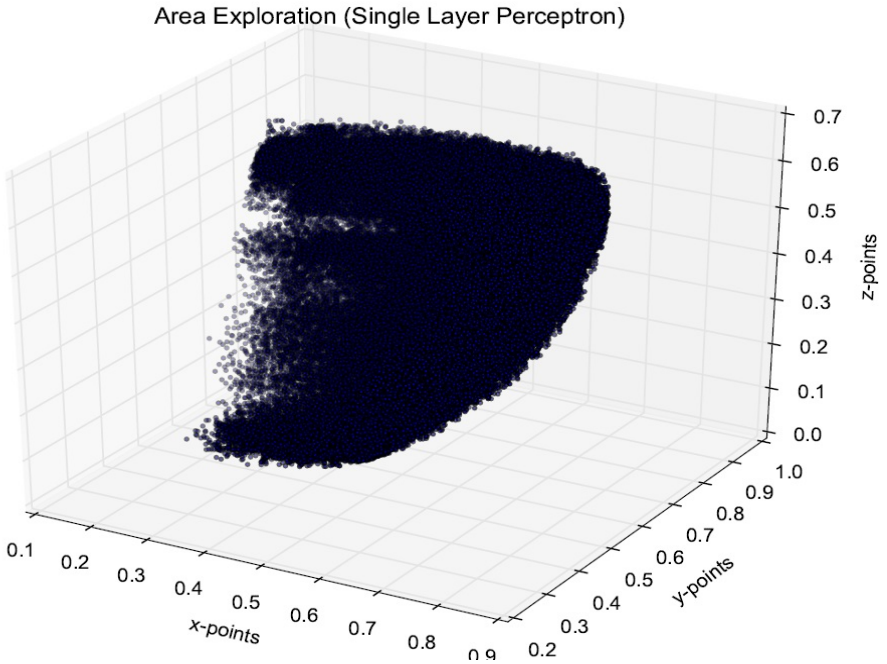


Figure 4.10: Area Explored by the SLP (LR0.717, E0.233 & 0 Hidden Nodes). The robot origion is at (x,y,z = 0.56,0.35,0.31)

4.4.4 Comparison of MLP and SLP

Considering the results discussed above, it was considered important to make a thorough comparison for better understanding of the behavior of MLP and SLP. The graph in figure 4.11 compared the error reduction of both perceptrons on fixed testing points. The SLP does error reduction faster and reaches its settling value quicker than the MLP. However, once it reaches the settling value, it increases the error a bit and just fluctuates on this value. Roughly it learns till 20th testing

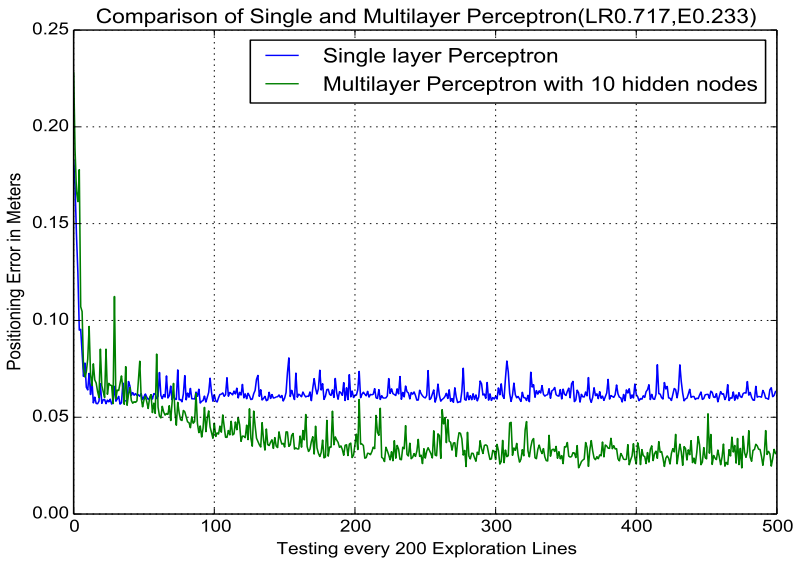


Figure 4.11: Comparison of fixed testing points error of MLP & SLP. Positioning error here is the average over the complete test set.

cycle which means that after 4,000 lines of exploration it stops learning. On the other hand MLP keeps on reducing the error till roughly 300th testing cycle. This means that the MLP is still getting better till exploring 60,000 lines. Further,

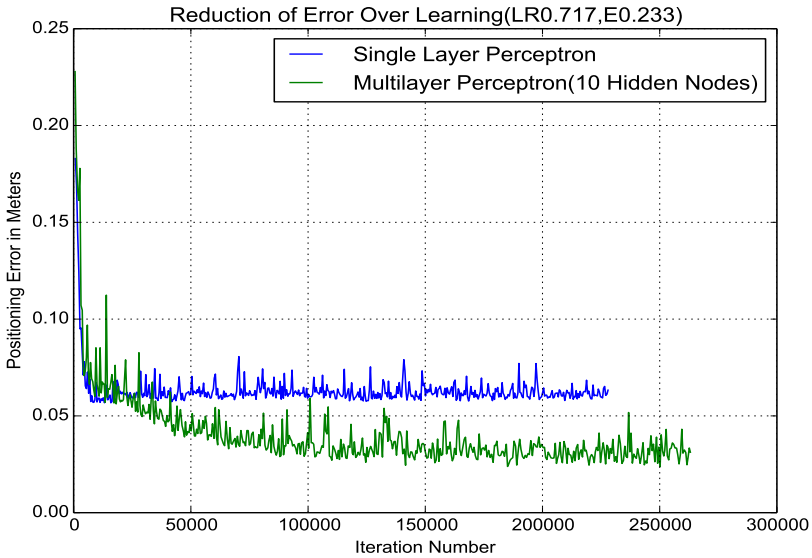


Figure 4.12: Reduction of error over iterations. Positioning error here is the average over the complete testing set

we compared the feasibility of our algorithm with Rolf [2013]. We plotted the reduction in error over iterations and the results can be viewed in figure 4.12.

MLP takes more iterations than SLP for the exploration of 100,000 lines. The exploration is purely random and answering why MLP takes more iterations is answered in terms of a more smoother line steps by the MLP. Also considering the fact that MLP is trying to trace the non-linearity better than the SLP, it jumps less inside the space and does more systematic exploration. Systematic exploration here means that the reachable space is explored is in lines and not random. As the MLP is taking more iterations than the SLP, this is the reason it takes more time in computation. Every 50,000 iterations take an average of 2 hours of computation time. The reduction of error over time can be seen in Appendix B in figure C.3

4.5 Conclusion

In this chapter we explained the model of the Neural Network learner for the designed arm of the robot NICU. We explained the approach of Rolf [2013] in detail at the start of the chapter. Later our approach for the learner was explained. We extended the original idea to explore three dimensional space with the use of Neural Networks. The safe use of high learning rate for online back-propagation was analyzed and explained. A detailed comparison of the performance of SLP and MLP was made in section 4.4.4. The parameter space was explored to find the best parameters in section 4.4.1. The best results were obtained using MLP and are explained in section 4.4.2 .This Neural learner required roughly 200,000 iterations to approximate the whole space as compared to the iteration 1,000,000 in 2D space in the approach mentioned earlier. Therefore, we can conclude the following points

1. Artificial Neural Networks (ANNs) can learn the inverse kinematics of a robotic arm in 3D space
2. Online back-propagation can safely handle high learning rates and perform better
3. Our learner learns the inverse kinematics faster than the original approach of Rolf [2013]. Furthermore, our learner traces points in space within accuracy of around 2 cm.
4. This learner can be easily extended to involve vision and head joints for real time grasping training.

There was a considerable amount of ideas that could not be implemented. So far the learner allows the robot to reach points in 3D space with an accuracy of 2 cm. In order to perform grasping, one important task is to detect the position of the objects in space. The robot has two cameras in the eyes that can be used to detect the objects. Once the object is detected this position will act as a goal and the arm will try to reach using the learner. An important modification here can be to put the eyes of the robot on object tracking. When the position of the object will be detected, the learner will take the position of the object as input as well as the position of two motors responsible for the head movement (See figure 3.8). The network can be trained in the simulator using the same algorithm with the extension that the inputs will take the position and orientation of the object as well as the position of the head and the neck motor to fully learn the 3D space around the robot. While performing grasping the robot head will again be on object tracking, the goal and head motor position will go as an input to the network and the learner can provide the accurate motor position to reach the object.

Chapter 5

Implementation on SURALP

5.1 Introduction

SURALP is a human sized full body humanoid robot developed by Sabanci University, Turkey. As stated in Erbatur et al. [2009], the robot has a total of 29 DOFs that include legs, arms, neck and the wrist of the robot. The actuators consist of a mechanism containing DC motors, belt and pulley system along with harmonics drive reduction gears. This robot contains inertial measurement systems, force/torque sensors, joint incremental encoders and cameras which are used for sensory feedback for the working of the robot.

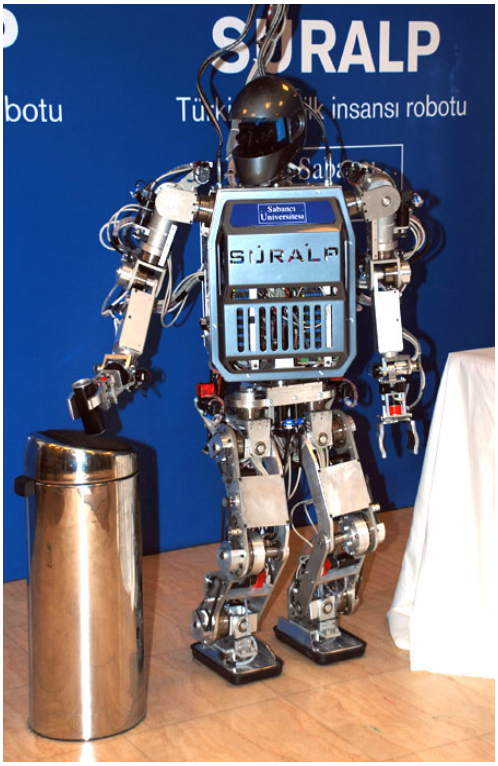


Figure 5.1: SURALP, A human sized full body humanoid robot

The control algorithm for the legs and walking trajectory generation is introduced in Erbatur et al. [2008a] .The legs contain 12 DOFs and are explained in Erbatur et al. [2008b]. This work explains the design of the legs and provides the detailed explanation about the components used in the design. The results reveal that the control algorithm used for the legs of the robot is successful in achieving a stable walk for the robot. SURALP can be visualized in figure 5.1.

SURALP also possess sophisticated and well performing arms with total of six DOFs on each arm. The shoulder contains three DOFs and additionally one on the elbow just like humans. The axes of the shoulder motion are orthogonal to each other and are followed by a revolute joint at the elbow. The wrist of the robot contains two DOFs for the orientation of the gripper that is present at the end of the arm. These joints at the wrist have rotations around one pitch and one roll axis.

In this chapter, we will further extend the approach explained in Chapter 4. The presented approach will be tested on the SURALP to enable it to position the gripper correctly in space. For this purpose, the simulator of the SURALP will be used for training the network and later the performance of the learner will be tested.

5.2 Simulator

The simulator model of SURALP is designed in MATLAB. An m-file computes the forward kinematics on the back of the simulator and the simulator provides the visual of the moment intended by the user. SURALP simulation model can be viewed in the figure 5.2. The simulator takes the joint angles and returns the

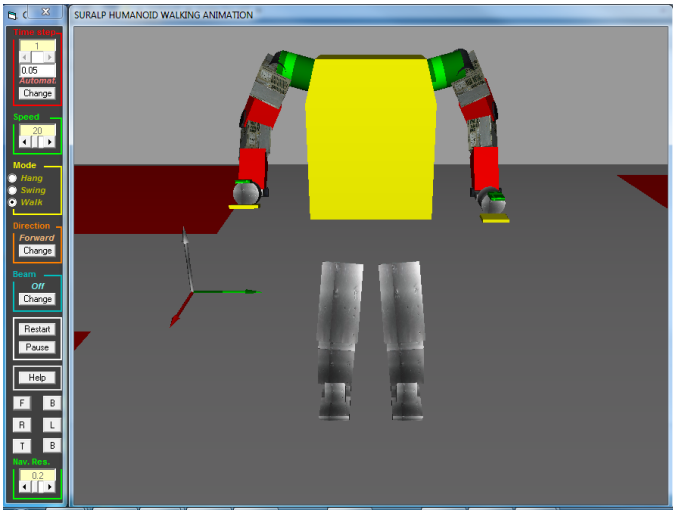


Figure 5.2: SURALP's Simulator Model

position of the point right in the middle of the grip of the robot. This simulation was very fast as compared to the simulation of NICU, due to the reason that it

is based on the computation of just one m-file. It was also possible to run the simulator without running the visual simulation and this was even faster in terms of running the experiments.

5.3 Experimental Setup

The arm design of SURALP has already been explained in the introduction of this chapter. It has a total six DOFs per arm. However, we will use first four i.e. three at the shoulder and one at the elbow. We want to solve the positioning problem of the end effector and this is reason we only want the first four DOFs. Humans also use these four DOFs for positioning; the later DOFs at the wrist are used to achieve the correct orientation.

The same idea of the neural network as explained in Chapter 4 is used for SURALP. The simulator model of SURALP runs with Matlab; therefore, it was more favorable to implement the algorithm in Matlab. The implemented algorithm was very fast regarding the computation. Our algorithm was already implemented in python in case of the experiments of robot NICU. Therefore, the idea of connecting the Python algorithm with Matlab was also considered. However, due to the delays in Python-Matlab connections, it was consider better to only use the whole experimental setup in Matlab.

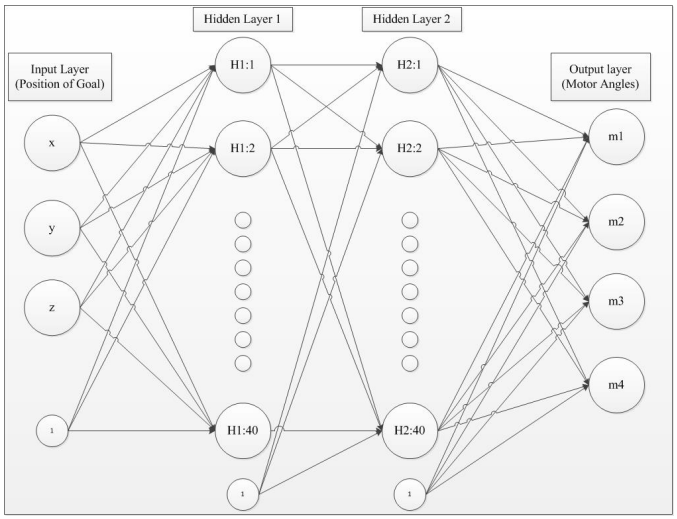


Figure 5.3: Artificial Neural Network Architecture for SURALP

We have the goal coordinates in 3D space (x,y,z) as the input and four motor values as the output of the learner. All the joints were used in a restricted space to avoid the collisions with the robot body. For the motors 1,2,3,4 respectively rotations of 90°,90°,45°,90° were allowed. We are using multilayer perceptron (MLP) as a learner with the help of online back-propagation. Our network has biases and does not have the momentum term. The sigmoid activation function (also see figure 1.6) is used on the hidden and output layer of the network for activation. The architecture of our Artificial Neural Network can be observed in 5.3.

Similar to the testing performed on robot NICU, we will create two testing set to check the learner's performance after a certain period of exploration. This means that in every testing after exploring 200 lines we will have different points for testing in the seen test set. This set will keep a check on the system's performance regarding the already seen points and will indicate if in a new exploration, the system forgets the old explored points. So in our fixed (static) testing set, we had a total of 60 points in the reachable space of the arm of SURALP that were at least 16 cm away from each other.

5.4 Results

In this section we will discuss the results obtained against the experimental setup explained in the last section. We explored several possibilities for the hyper parameters for SURALP experiments. However, a detailed exploration of parameters for SURALP was not possible. The best results were obtained against the network architecture of two hidden layers with 40 neurons each with learning rate of 0.7 and perturbation term of 0.233. The learning in the case of SURALP was more complex then NICU due to the non-linearity imposed by the shoulder design of SURALP. The reduction of error for the fixed testing set (No. of training samples) is demonstrated in figure 5.4.

In can be seen that after 2839 testing we reached an average error of 2.8 cm and it is further decreasing as the learner continues to learn the space. One important aspect here is to check the number of samples needed for reaching this error level and it can be seen in the figure 5.5.

The error over the testing set for the seen points was also recorded and it can be observed in 5.6. We reached an average error as low as 1.9 cm. The difference in the error value for fixed and seen testing can be justified same way as explained in the case of NICU. The fixed testing set gives the measure of error reduction over the whole reachable space. The complete reachable space also comprises of the points that are very difficult to reach or are maybe covered only few times. Therefore, the network is not able to learn these points very accurately. The reduction of error over the seen point against the iterations (Number of samples) can be seen in figure 5.7.

In the case of SURALP, the iteration required for the learner to reach an acceptable error level is the same as was needed for the experiment of Rolf [2013]. However, there is one thing to be considered, the problem size. Our problem is bigger in comparison to the problem of the above mentioned work, as it is spread in 3D and the author's problem is for 2D planer arm. Therefore, it can be said that our approach is faster than the approach of the author mentioned above. Another important aspect is visualization of the area covered by the arm during exploration of space. For experimentation we used only the right arm of SURALP and the area covered by the arm can be seen in figure 5.8

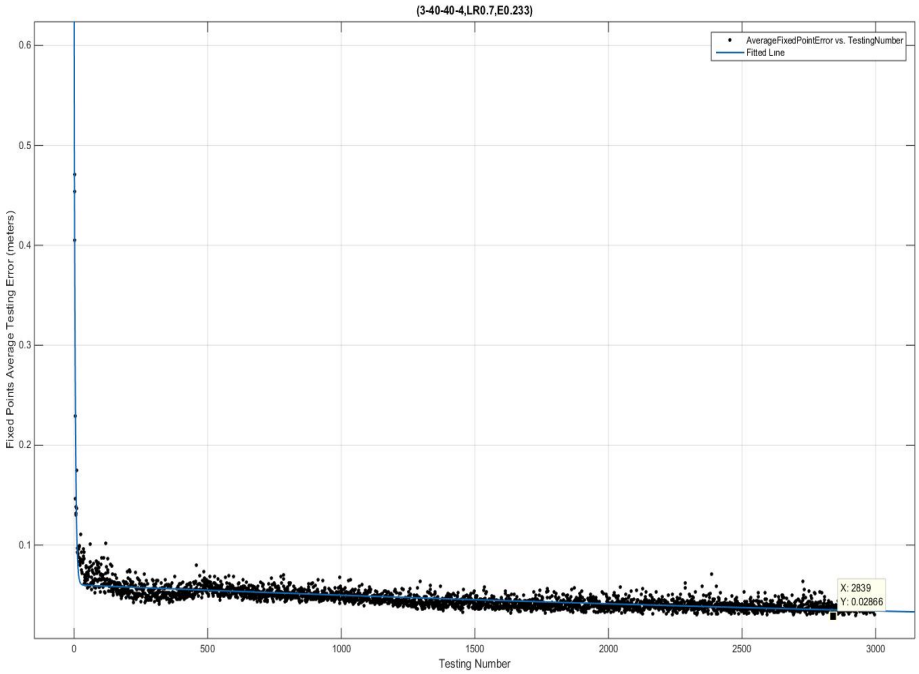


Figure 5.4: Reduction of average error over fixed testing set

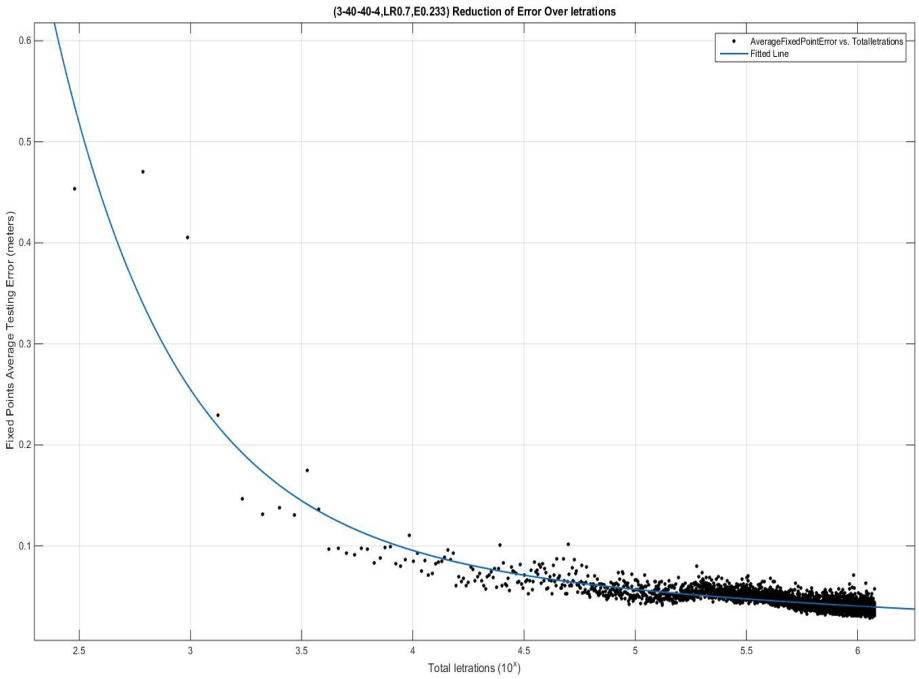


Figure 5.5: Reduction of average fixed testing error over iterations

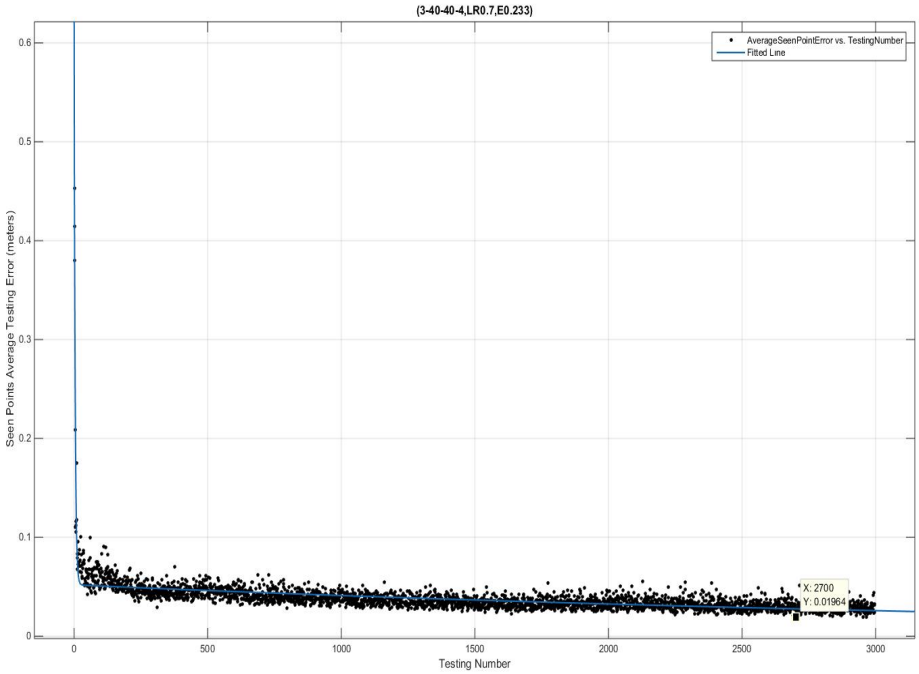


Figure 5.6: Reduction of average error over seen testing set

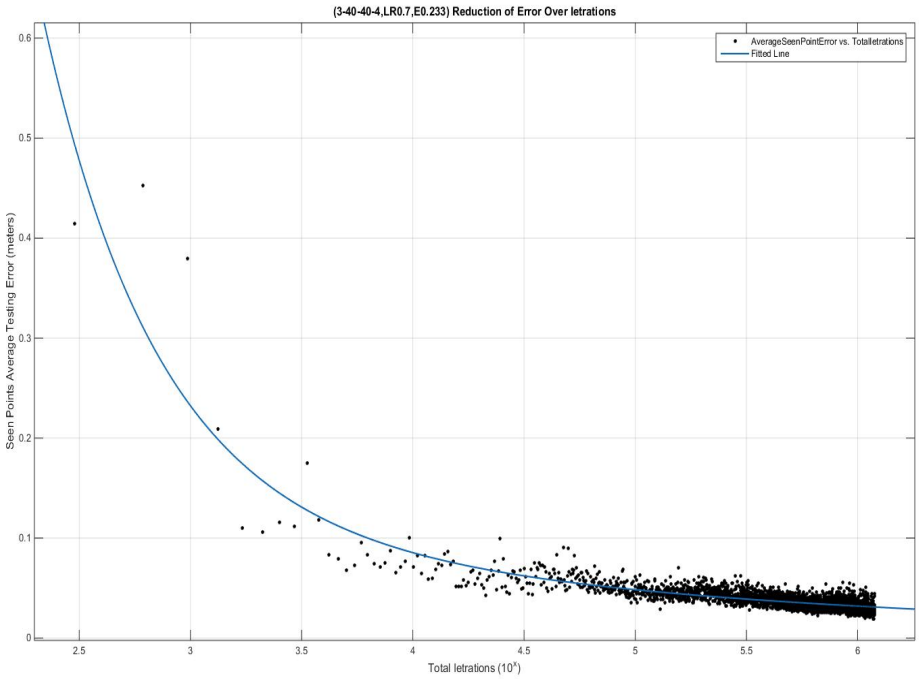


Figure 5.7: Reduction of average seen testing error over iterations

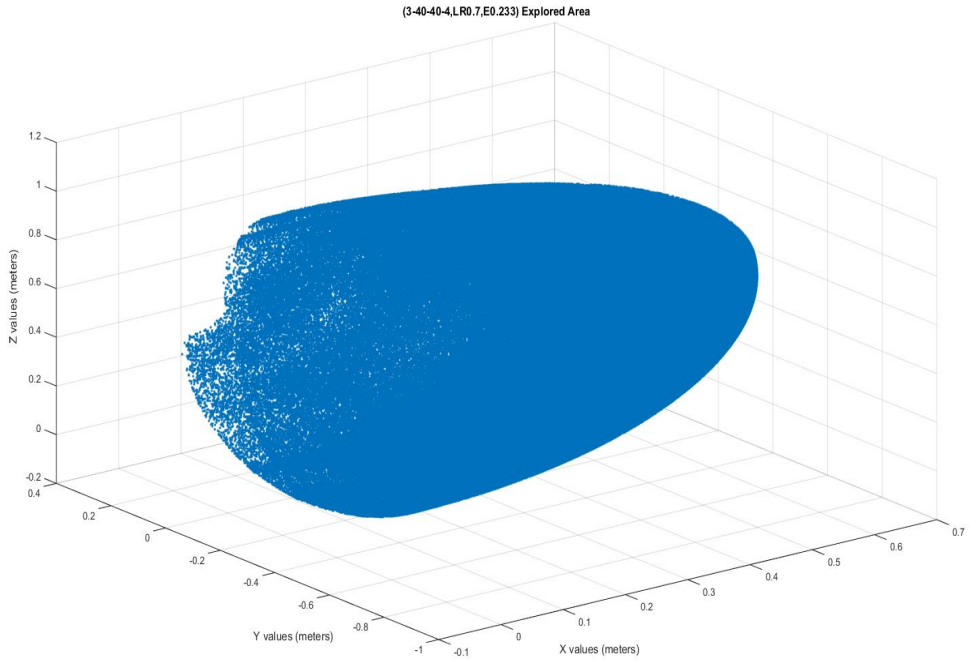


Figure 5.8: Area explored by the right arm of SURALP. Robot's origin is at (0,0,0)

5.5 Conclusion

In this chapter we explained the model of the Neural Network learner designed for the arms of the robot SURALP. The best results were obtained for a network with two hidden layers that suggest that spreading the problem over higher space (increased rotations of joints) and the shoulder design of SURALP makes the problem harder to learn. The results of the algorithm implementation on SURALP also strengthen the generalizing approach of our algorithm. Therefore, we can conclude the following points

1. Artificial Neural Networks (ANNs) can learn the inverse kinematics of a humanoid robotic arm in 3D space.

2. Changing the shoulder design or the rotation joints or spans of rotation make the learning problem more difficult.

3. This learner can easily be extended to involve vision and head joints for the training of real time grasping.

So far the learner allows the robot to reach points in 3D space and we have not included the orientation of the gripping. The orientation of the gripper can be included and the network can be retrained until an acceptable error is achieved.

Chapter 6

Conclusion

In this thesis we worked on the design of the arms of a humanoid robot and the control algorithm of it. Chapter 2 showed the related work done in the field of humanoid robotic arm design and later showed some approaches for solving the inverse kinematics problem. Chapter 3 explained the approach and the design of the robotic arm. There we explained in detail the design procedure of the arm as well as the design procedure for the URDF file of the robot. The printing of the parts in plastic depicts an excellent use of 3D plastic printer for rapid prototype design. The advantages achieved by this design have been stated in section 3.6.

In chapter 4, we worked on solving the inverse kinematics of the arm. We introduced the inspiration for using online goal babbling with direction sampling and then explained our approach. The approach works well with the use of Artificial Neural Networks (ANNs). The inverse kinematics problem is solved by using goal babbling with direction sampling and ANNs as the learner in the process. The network requires no prior knowledge about the arm. It starts with one single position and explores the whole space with time. Another task on the way was to find the best parameters for the network. This has been addressed in section 4.4.1. After the exploration the network can approximate the whole 3D space with an accuracy of 2 cm. The results have been shown in section 4.4. The contribution with this learner design has already been mentioned in section 4.5. However, they will be highlighted again in this chapter. In chapter 5, we implemented the same algorithm on SURALP, a full human sized humanoid robot with bigger arms. The algorithm worked well for the bigger sized robot. Also the learner worked for a higher span of joint movements as compared to NICU.

Now we will discuss the thesis goals set in section 1.3. We modified the design of the robot arms of NICU and equipped it with a multi degrees of freedom arms. This arm design removed the problem of movement restriction due to an extended chest of the robot. The arm joints are bio-inspired but we could not make it a completely bio-inspired 7 degrees of freedom arm. The reason why we used 5 instead of 7 degrees of freedom was to keep the weight of the new arm close to the old one. This removes the stability issue for the safe walk of the robot with new arms. Also with this design the robot can reach considerable amount of points in 3D space as shown in section 4.4. In the light of this discussion, we

can conclude that we met the requirements for an arm design task. The second task was to implement a bio-inspired learner for the arm. We implemented an Artificial Neural Network that is inspired by bio logical brain to learn the inverse kinematics of the arm. The learner allows the robot to reach the points in 3D space with an accuracy of around 2 cm. Our learner learns the inverse kinematics faster compared to Rolf [2013]. The third task was to implement the algorithm on SURALP. It also worked well for SURALP and the results can be seen in section 5.4.

The learner design can easily accumulate vision input and can be retrained to check its utility with vision input. Considering this discussion, we can conclude that we were successful in designing a neural arm learner design that can also adaptive.

6.1 Contributions

In the light of the discussion in the last section and the thesis goal defined in section 1.3, we can conclude that the thesis fulfilled the targeted tasks successfully. The completion of three targeted tasks can be justified as below answers;

- ✓ The designed arms of robot NICU have five degrees of freedom that can allow it to maneuver excellently in the a 3D space. All the designed parts of the arm were printed using 3D printer. A grip at the end of the arm allows the robot to have a physical interaction with the environment and it can perform grasping in the future.
- ✓ We implemented a learner model using bio-inspired Artificial Neural Networks. The learner allows the robot NICU's arm to successfully trace multiple test points in the 3D space with an accuracy of around 2cm.
- ✓ We implemented the learner on SURALP and it successfully traced multiple points in 3D space with an accuracy of roughly around 2.5-3 cm.

Furthermore, this thesis also contributes to following topics

1. A multi DOF arm design using Dynamixel Servos for humanoid robot.
2. A structure for the shoulder design of the robot that will minimize the danger to the stability of the robot.
3. The procedure of creating the accurate robot model (URDF) for the simulator.
4. Online learning can handle high learning rates and can provide better results.
5. Multilayer Perceptron can approximate the problem of inverse kinematics in 3D space.

6. The developed learner works faster than the original learner of Rolf [2013]. The mentioned work is in 2D and a comparison can be viewed in the Appendix A.

6.2 Future Works

There were considerable amount of ideas that could not be implemented. The future work has already been discussed in section 3.6 for the arm design, the learner design of NICU in section 4.5 and the learner design of SURALP 5.5; they will be shortly summarized again.

1. The wrist of the arm can be extended to have one or two more degrees of freedom to provide more orientation possibilities for the gripper.

2. The learner can be extended to take vision and head motor positions as input and can be trained to perform real time grasping.

Appendix A

Additional Information

A.1 The case of the planner arm

Rolf [2013] implements the idea on a planner arm of one meter length with 5 degrees of freedom. In order to have a clear comparison we reconstructed the environment for this experiment. This section will provide the details of the experiments and discuss the results obtained.

A.1.1 Simulator and Experimental Setup

We created a 5 degrees of freedom planner arm with a length of one meter in V-rep which was controlled in the similar way the simulator of NICU. We trained this arm with our algorithm. The arm can be seen in figure A.1.

For quick experimental result, we used a Neural Network learner with two hidden layers with 100 neurons in each. This high sized network was chosen to exploit the results and the hyper parameter optimization was not performed for the optimal number of neurons selection. There can be a solution that can approximate the problem with much smaller network size, but we did not explore the network size due to a lack of time. The network architecture can be seen in figure A.2. We have x and y coordinate of the point in 2D space as input and there are five joints values at the output of the network. The network has learned the sequence of joints positions to reach a goal successfully.

A.1.2 Results

Similar to the experiments explained in the chapter 4 and 5, we explored 200 lines in this experiment and performed the testing. The reduction in the testing error against the testing cycles can be seen in figure A.3.

It can be seen from the figure that the network starts to settle between 250 to 300 testing cycles that roughly correspond to 250,000 to 300,000 samples. Though our final settling error is high, it can be reduced once the optimal size and hyper parameters for the network are discovered. Figure A.4 can be seen to have more clarity about the reduction in error over iterations (samples).

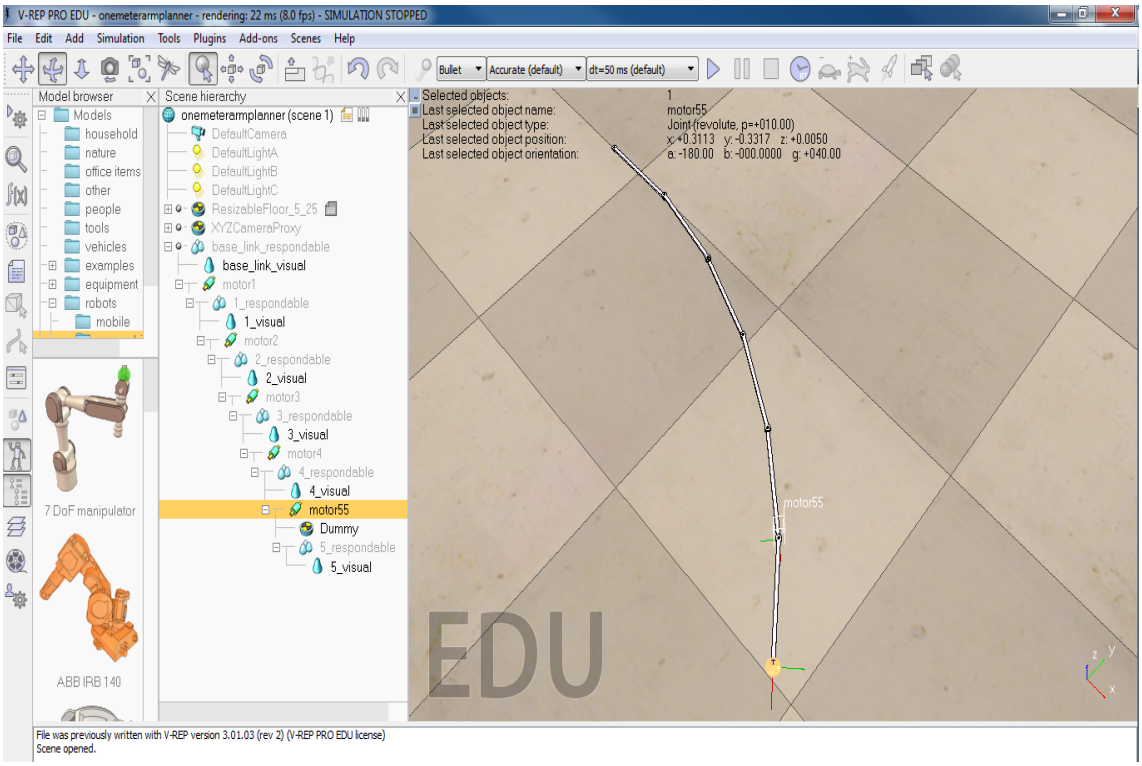


Figure A.1: SURALP, A human sized full body humanoid robot

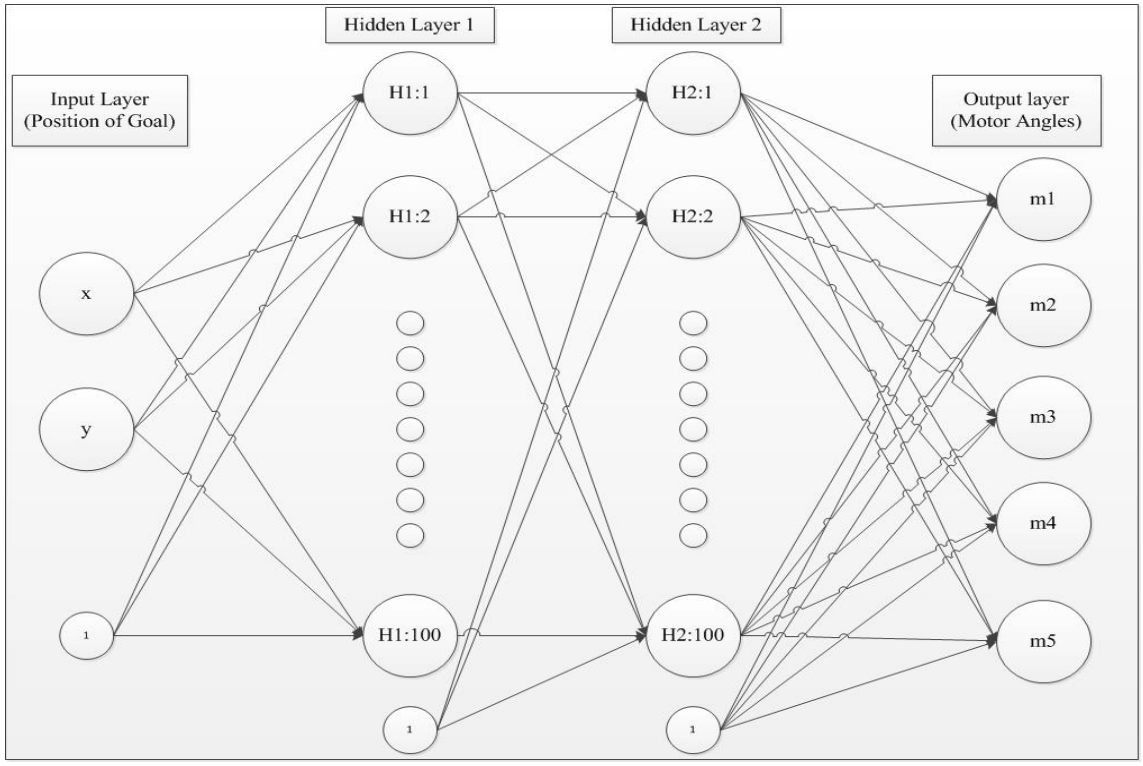


Figure A.2: Artificial Neural Network Architecture

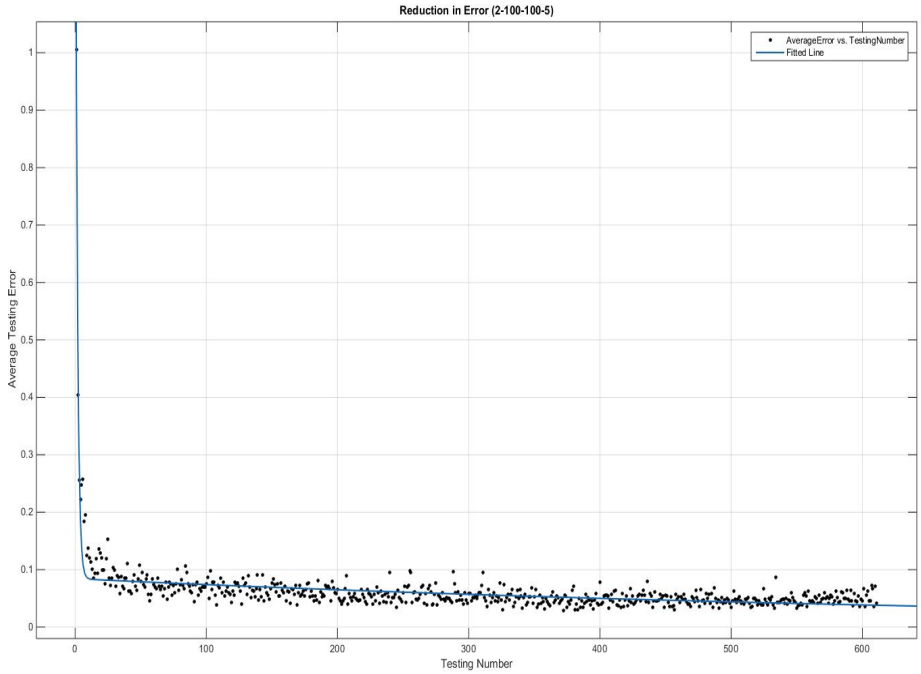


Figure A.3: Reduction in average testing error

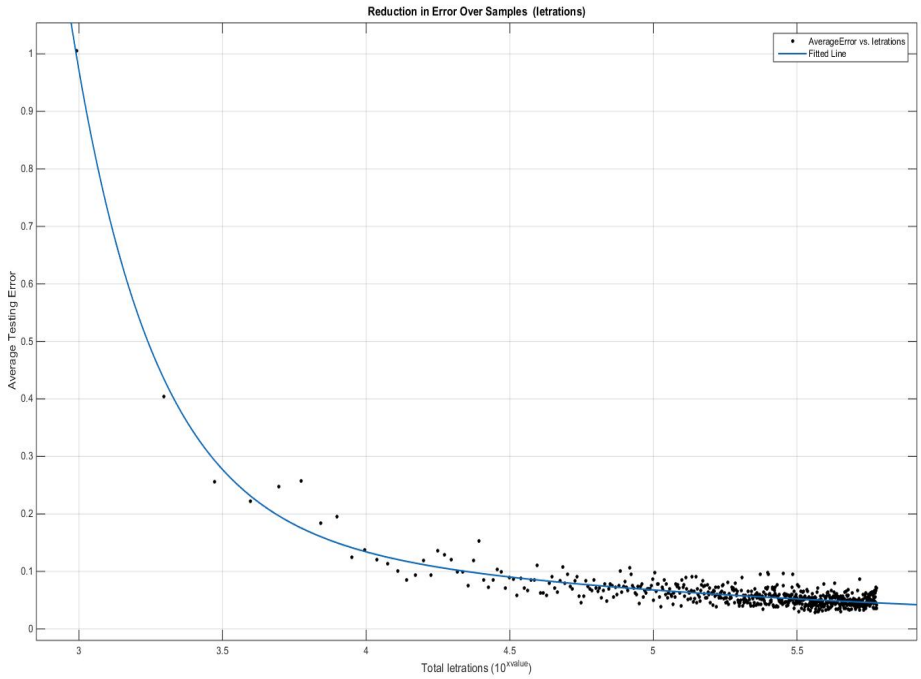


Figure A.4: Reduction in average testing error over iterations

A.1.3 Workspace discovery

In the experiment of Rolf [2013], a claim was made about the workspace discovery. The author shows the exploration of the reachable space in 10^3,10^4,10^5 and 10^6 samples of exploration. Similarly, we draw the work space discovery for our experiment and figure A.5, A.6, A.7 and A.8 shows the workspace discovery for 10^3,10^4,10^5 and $10^{5.7784}$ samples of exploration. It can be seen from the exploration till 10^4 that the algorithm has already covered major part of the reachable space. Nevertheless, author's approach need more samples to see the full space.

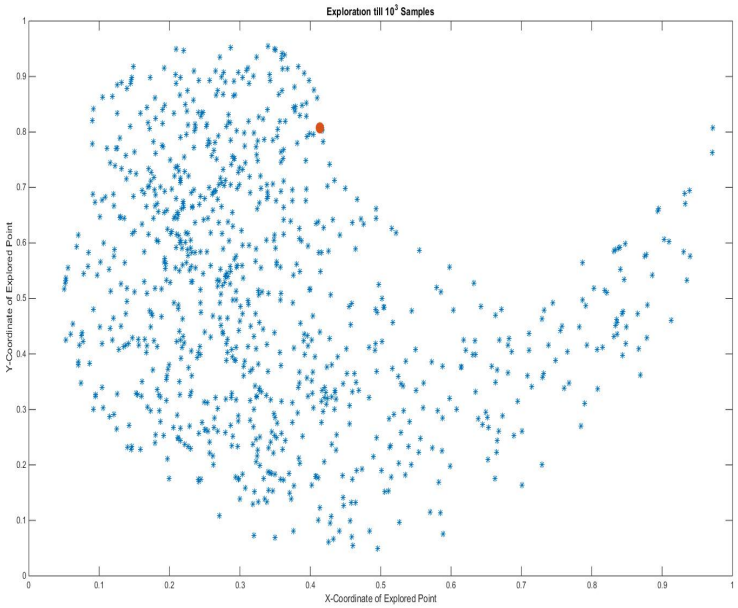


Figure A.5: Workspace exploration till 10^3 samples. Red dot in the figure shows the origin of the arm.

A.1.4 Conclusion

Considering the results generated above and the case of 3D problem of NICU and SURALP, we can claim that our algorithm works faster than the original of Rolf [2013]. Also the space is covered more quickly by our algorithm. A good choice of the parameters and size of the network can enable fast exploration as well as fast learning of the inverse estimate as was seen in the case of robot NICU.

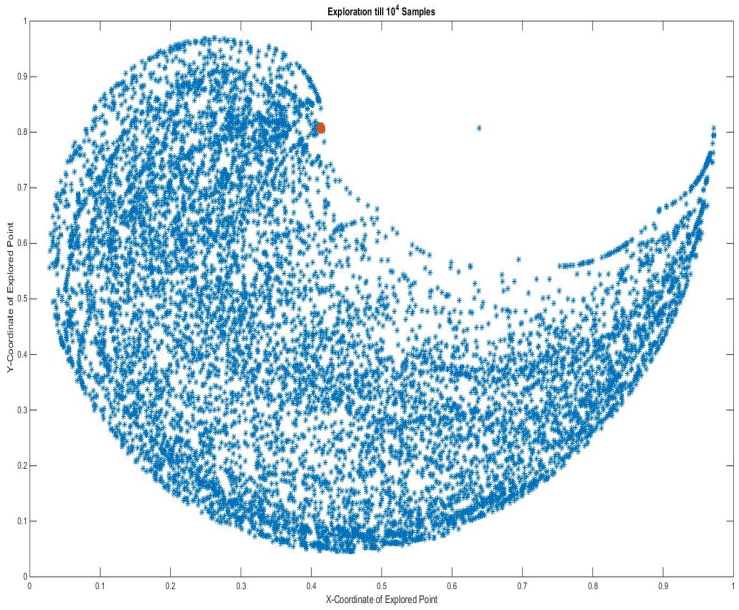


Figure A.6: Workspace exploration till 10^4 samples. Red dot in the figure shows the origin of the arm.

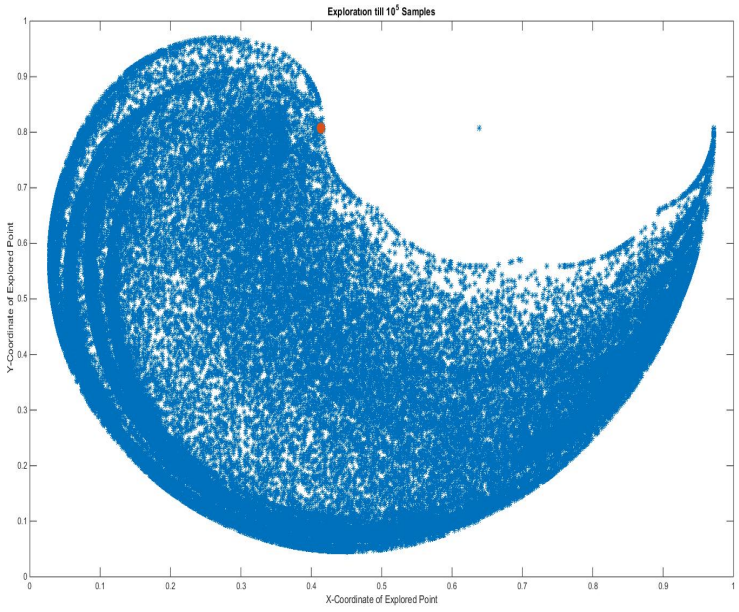


Figure A.7: Workspace exploration till 10^5 samples. Red dot in the figure shows the origin of the arm.

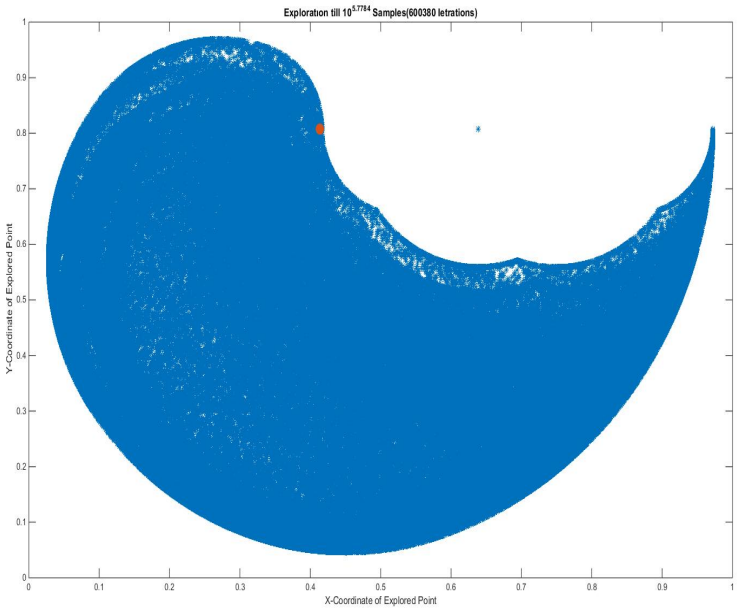


Figure A.8: Workspace exploration till $10^{5.7784}$ samples. Red dot in the figure shows the origin of the arm.

Appendix B

Technical Details

B.1 NICU

This chapter will explain the tools used for programming and simulations. We worked with Python as the programming language and V-rep as the simulator for robot actions. The algorithm is fully developed by using "Numpy" module of Python. The "Time" & "Math" is used for additional help in implementation. Python code is connected to v-rep with the help of remoteAPI function of V-rep. RemoteAPI connects the algorithm with V-rep and helps to translate coded action to the simulator action. The algorithm uses "vrep.py" to acquire the handles for all the joints that the program will control as well as it enables the remoteAPI. The developed algorithm is explained on the next page

B.2 SURALP

In the case of SURALP same algorithm is used but the implementation was done in Matlab. The simulator also worked with Matlab and the connection with the simulator was done by calling the control file of the simulator in the algorithm. The algorithm can be seen on the next page

B.3 Learning Algorithm

Algorithm 1 Learning algorithm for the Neural Learner as explained in section 4.2

Require: Handles for All joints in V-rep
Ensure: Remote API connected & All joint handles acquired
Get End effector position $\mathrm{P}^{Home}_{(x,y,z)}$ against motor position $\mathrm{M}^{Home}_{(m1,m2,m3,m4)}$
Over-fit the network on ($\mathrm{P}^{Home}_{(x,y,z)}$,$\mathrm{M}^{Home}_{(m1,m2,m3,m4)}$)
while $AverageError \geq 0.25$ **do**
  while $Lines \leq 200$ **do**
    Generate a random perturbation term E
    while Joints limits are not saturated **do**
      Add perturbation term in position ($P_{new} = P_{x,y,z} + E$)
      Run the learner on P_{new} and get the output motor position $\mathrm{M}_{(m1,m2,m3,m4)}$
      Add E in motors ($M_{new(m1,m2,m3,m4)} = M_{(m1,m2,m3,m4)} + E$)
      Send $\mathrm{M}_{New(m1,m2,m3,m4)}$ to simulator and get accurate P^*
      Train Network on ($\mathrm{M}_{New(m1,m2,m3,m4)}$,$P^*$)
    end while
  end while
  while $SeenTestingPoints \leq 110(60 for SURALP)$ **do**
    Generate testing point from seen data P_{Seen}
    Run the network on P_{Seen} and get $\mathrm{M}_{Seen(m1,m2,m3,m4)}$
    Send $\mathrm{M}_{Seen(m1,m2,m3,m4)}$ to simulator and get real Position P_{Real}
    Calculate the distance between P_{Seen} & P_{Real} and save
    Calculate the average of error on all points and save
  end while
  while $FixedTestingPoints \leq 110(60 for SURALP)$ **do**
    Take the points one by one from Fixed data set P_{Fixed}
    Run the network on P_{Fixed} and get $\mathrm{M}_{Fixed(m1,m2,m3,m4)}$
    Send $\mathrm{M}_{Fixed(m1,m2,m3,m4)}$ to simulator and get real Position P_{Real}
    Calculate the distance between P_{Fixed} & P_{Real}
    Calculate the average of error on all 110(60 in case of SURALP) points and save as "AverageError"
  end while
end while
Save the Network weights

Appendix C

Additional Data

All the tables included in this section contain the mean and standard deviation values in meters for the respective experiments. This mean and standard deviation is calculated over the average of positioning error for the fixed testing set cycles. Where one testing occurs after the exploration of 200 lines.

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
2	0.1	0.1	0.236096	0.026751
3	0.1	0.1	0.220356	0.019332
4	0.1	0.1	0.238868	0.023525
5	0.1	0.1	0.212238	0.026288
6	0.1	0.1	0.233760	0.029222
7	0.1	0.1	0.216312	0.032273
8	0.1	0.1	0.184633	0.026380
9	0.1	0.1	0.221171	0.023068
10	0.1	0.1	0.205095	0.025801
15	0.1	0.1	0.200731	0.039947
17	0.1	0.1	0.164684	0.038653
20	0.1	0.1	0.151482	0.021772
25	0.1	0.1	0.144096	0.011967
50	0.1	0.1	0.101859	0.024777
75	0.1	0.1	0.075518	0.010685
100	0.1	0.1	0.086544	0.031497
120	0.1	0.1	0.080635	0.010513
140	0.1	0.1	0.081105	0.019714
160	0.1	0.1	0.074059	0.009267

Table C.1: This table shows the node variation against Learning rate of 0.1 and Perturbation of 0.1

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
3	0.05	0.1	0.221377	0.022174
4	0.05	0.1	0.254315	0.027652
7	0.05	0.1	0.227548	0.022398
10	0.05	0.1	0.219146	0.024870
15	0.05	0.1	0.237034	0.030213
20	0.05	0.1	0.223948	0.021444
25	0.05	0.1	0.202520	0.016187
30	0.05	0.1	0.209175	0.029464
35	0.05	0.1	0.182068	0.029732
50	0.05	0.1	0.172291	0.027626
70	0.05	0.1	0.150433	0.042675
75	0.05	0.1	0.139117	0.033082
100	0.05	0.1	0.132246	0.024265
120	0.05	0.1	0.090488	0.022088
140	0.05	0.1	0.097007	0.017543
160	0.05	0.1	0.105333	0.039306

Table C.2: Node variation for Learning rate of 0.05 and Perturbation of 0.1

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
3	0.1	0.05	0.244255	0.022113
15	0.1	0.05	0.267995	0.036022
35	0.1	0.05	0.250224	0.039265
50	0.1	0.05	0.256366	0.036852
75	0.1	0.05	0.242214	0.047565
100	0.1	0.05	0.211772	0.049542
125	0.1	0.05	0.245231	0.040243
150	0.1	0.05	0.182118	0.046863

Table C.3: Node variation for Learning rate of 0.1 and Perturbation of 0.05

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
3	0.05	0.05	0.253033	0.023304
7	0.05	0.05	0.258917	0.020386
25	0.05	0.05	0.225370	0.029799
50	0.05	0.05	0.252009	0.040319
75	0.05	0.05	0.244883	0.030021
100	0.05	0.05	0.260085	0.042495
125	0.05	0.05	0.209285	0.032891
150	0.05	0.05	0.194287	0.030509

Table C.4: Node variation for Learning rate of 0.05 and Perturbation of 0.05

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
10	0.1	0.02	0.294123	0.033903
30	0.1	0.02	0.272654	0.041830
50	0.1	0.02	0.266493	0.030158
70	0.1	0.02	0.272377	0.045938
90	0.1	0.02	0.272372	0.053519
110	0.1	0.02	0.289562	0.036083
130	0.1	0.02	0.288927	0.037664

Table C.5: Node variation for Learning rate of 0.1 and Perturbation of 0.02

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
20	0.7	0.1	0.059872	0.006127
40	0.7	0.1	0.065433	0.009900
60	0.7	0.1	0.073884	0.011385
80	0.7	0.1	0.068599	0.005651
100	0.7	0.1	0.088985	0.027700
120	0.7	0.1	0.075554	0.017645

Table C.6: Node variation for Learning rate of 0.7 and Perturbation of 0.1

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
15	0.7	0.05	0.185031	0.039550
35	0.7	0.05	0.116503	0.036042
55	0.7	0.05	0.138861	0.040169
75	0.7	0.05	0.113368	0.019562
95	0.7	0.05	0.124562	0.030492
115	0.7	0.05	0.122645	0.022745
135	0.7	0.05	0.130080	0.040625

Table C.7: Node variation for Learning rate of 0.7 and Perturbation of 0.05

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
35	0.07	0.02	0.295082	0.025513
65	0.07	0.02	0.290077	0.039307
85	0.07	0.02	0.273196	0.033505
105	0.07	0.02	0.281550	0.040388
135	0.07	0.02	0.268184	0.034184

Table C.8: Node variation for Learning rate of 0.7 and Perturbation of 0.02

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.717	0.233	0.062618	0.002161
1	0.717	0.233	0.144636	0.011873
2	0.717	0.233	0.071314	0.005665
5	0.717	0.233	0.051673	0.047163
10	0.717	0.233	0.046847	0.005433
15	0.717	0.233	0.049994	0.041104
30	0.717	0.233	0.049489	0.007825
50	0.717	0.233	0.0504315	0.006054
70	0.717	0.233	0.058008	0.009380
90	0.717	0.233	0.053945	0.002377
110	0.717	0.233	0.054893	0.005846
130	0.717	0.233	0.059206	0.006983
150	0.717	0.233	0.058645	0.006644

Table C.9: Node variation for Learning rate of 0.717 and Perturbation of 0.233

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.266	0.314	0.059682	0.001036
1	0.266	0.314	0.137183	0.003235
2	0.266	0.314	0.074255	0.001831
5	0.266	0.314	0.063103	0.002142
10	0.266	0.314	0.057487	0.004176
15	0.266	0.314	0.057252	0.002531
30	0.266	0.314	0.057828	0.003700
50	0.266	0.314	0.060947	0.005107
70	0.266	0.314	0.060213	0.004148
90	0.266	0.314	0.060909	0.003233
110	0.266	0.314	0.062925	0.0047789
130	0.266	0.314	0.062144	0.006594
150	0.266	0.314	0.065111	0.012237

Table C.10: Node variation for Learning rate of 0.266 and Perturbation of 0.314

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.11	0.18	0.066122	0.006199
1	0.11	0.18	0.143915	0.007858
2	0.11	0.18	0.078253	0.010661
5	0.11	0.18	0.078289	0.019845
10	0.11	0.18	0.0566002	0.011136
15	0.11	0.18	0.064612	0.00425
30	0.11	0.18	0.0627069	0.002297
50	0.11	0.18	0.063207	0.003538
70	0.11	0.18	0.056823	0.011099
90	0.11	0.18	0.066069	0.006549
110	0.11	0.18	0.056525	0.011130
130	0.11	0.18	0.067029	0.005009
150	0.11	0.18	0.066746	0.007615

Table C.11: Node variation for Learning rate of 0.11 and Perturbation of 0.18

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.527	0.29	0.060136	0.002843
1	0.527	0.29	0.143997	0.006232
2	0.527	0.29	0.073270	0.002021
5	0.527	0.29	0.057751	0.006119
10	0.527	0.29	0.0510503	0.006042
15	0.527	0.29	0.052934	0.006125
30	0.527	0.29	0.054537	0.006770
50	0.527	0.29	0.053361	0.003653
70	0.527	0.29	0.054875	0.004912
90	0.527	0.29	0.056487	0.005185
110	0.527	0.29	0.055359	0.002912
130	0.527	0.29	0.061873	0.008824
150	0.527	0.29	0.060922	0.006988

Table C.12: Node variation for Learning rate of 0.527 and Perturbation of 0.29

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.919	0.15	0.067429	0.005298
1	0.919	0.15	0.158841	0.019250
2	0.919	0.15	0.075729	0.005921
5	0.919	0.15	0.063710	0.012985
10	0.919	0.15	0.054506	0.010272
15	0.919	0.15	0.053417	0.014045
30	0.919	0.15	0.059211	0.010019
50	0.919	0.15	0.054364	0.005234
70	0.919	0.15	0.054606	0.006909
90	0.919	0.15	0.064199	0.011599
110	0.919	0.15	0.063820	0.010673
130	0.919	0.15	0.066861	0.006672
150	0.919	0.15	0.068034	0.019641

Table C.13: Node variation for Learning rate of 0.919 and Perturbation of 0.15

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.64	0.1	0.065276	0.005862
1	0.64	0.1	0.172595	0.022316
2	0.64	0.1	0.883950	0.018928
5	0.64	0.1	0.067343	0.009050
10	0.64	0.1	0.071726	0.015799
15	0.64	0.1	0.075028	0.027545
30	0.64	0.1	0.071155	0.018334
50	0.64	0.1	0.076032	0.014377
70	0.64	0.1	0.079569	0.030135
90	0.64	0.1	0.081321	0.01457
110	0.64	0.1	0.081329	0.015519
130	0.64	0.1	0.0848758	0.014430
150	0.64	0.1	0.083283	0.011265

Table C.14: Node variation for Learning rate of 0.64 and Perturbation of 0.1

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.08	0.18	0.074160	0.005019
1	0.08	0.18	0.207759	0.022851
2	0.08	0.18	0.152389	0.022036
5	0.08	0.18	0.096359	0.020687
10	0.08	0.18	0.098850	0.022148
15	0.08	0.18	0.072339	0.011962
30	0.08	0.18	0.064055	0.004669
50	0.08	0.18	0.063208	0.002735
70	0.08	0.18	0.069150	0.022853
90	0.08	0.18	0.068836	0.009135
110	0.08	0.18	0.062927	0.006195
130	0.08	0.18	0.065719	0.004893
150	0.08	0.18	0.068208	0.006928

Table C.15: Node variation for Learning rate of 0.08 and Perturbation of 0.18

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.02	0.314	0.093980	0.006467
1	0.02	0.314	0.189213	0.015279
2	0.02	0.314	0.184484	0.016922
5	0.02	0.314	0.138199	0.010098
10	0.02	0.314	0.099485	0.012001
15	0.02	0.314	0.091549	0.012063
30	0.02	0.314	0.088384	0.013563
50	0.02	0.314	0.061791	0.003565
70	0.02	0.314	0.060111	0.002488
90	0.02	0.314	0.059793	0.002714
110	0.02	0.314	0.058157	0.001344
130	0.02	0.314	0.059015	0.001912
150	0.02	0.314	0.059156	0.001697

Table C.16: Node variation for Learning rate of 0.02 and Perturbation of 0.314

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.0527	0.29	0.067587	0.004657
1	0.0527	0.29	0.143738	0.013429
2	0.0527	0.29	0.134324	0.004421
5	0.0527	0.29	0.067809	0.005193
10	0.0527	0.29	0.066371	0.005776
15	0.0527	0.29	0.061485	0.002714
30	0.0527	0.29	0.060614	0.002342
50	0.0527	0.29	0.060793	0.001725
70	0.0527	0.29	0.058747	0.001356
90	0.0527	0.29	0.063705	0.007110
110	0.0527	0.29	0.064936	0.003702
130	0.0527	0.29	0.061898	0.004213
150	0.0527	0.29	0.061499	0.005231

Table C.17: Node variation for Learning rate of 0.0527 and Perturbation of 0.29

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.011	0.18	0.195042	0.015694
1	0.011	0.18	0.223507	0.021315
2	0.011	0.18	0.214471	0.013082
5	0.011	0.18	0.221027	0.017516
10	0.011	0.18	0.215418	0.015766
15	0.011	0.18	0.214533	0.011596
30	0.011	0.18	0.211348	0.045017
50	0.011	0.18	0.144849	0.018683
70	0.011	0.18	0.224528	0.016833
90	0.011	0.18	0.123248	0.023721
110	0.011	0.18	0.136134	0.024184
130	0.011	0.18	0.090369	0.010361
150	0.011	0.18	0.089089	0.009314

Table C.18: Node variation for Learning rate of 0.011 and Perturbation of 0.18

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.717	0.023	0.243960	0.031031
1	0.717	0.023	0.243578	0.033957
2	0.717	0.023	0.284386	0.034913
5	0.717	0.023	0.27286	0.031973
10	0.717	0.023	0.263586	0.043219
15	0.717	0.023	0.28756	0.040251
30	0.717	0.023	0.273833	0.033092
50	0.717	0.023	0.265837	0.037996
70	0.717	0.023	0.262458	0.045209
90	0.717	0.023	0.249371	0.051501
110	0.717	0.023	0.243495	0.046726
130	0.717	0.023	0.239509	0.039217
150	0.717	0.023	0.259036	0.043361

Table C.19: Node variation for Learning rate of 0.717 and Perturbation of 0.023

Hidden Nodes	Learning Rate	Perturbation	Mean	Standard Deviation
0	0.266	0.031	0.259651	0.034883
1	0.266	0.031	0.285035	0.030802
2	0.266	0.031	0.266964	0.044683
5	0.266	0.031	0.282754	0.029072
10	0.266	0.031	0.260505	0.048857
15	0.266	0.031	0.248344	0.043904
30	0.266	0.031	0.266421	0.042296
50	0.266	0.031	0.271387	0.030441
70	0.266	0.031	0.273595	0.048482
90	0.266	0.031	0.241216	0.031597
110	0.266	0.031	0.256748	0.043586
130	0.266	0.031	0.257564	0.038239
150	0.266	0.031	0.241182	0.049682

Table C.20: Node variation for Learning rate of 0.266 and Perturbation of 0.031

LR&E	5 Nodes	5 Nodes	10 Nodes	10 Nodes	15 Nodes	15 Nodes
values	Mean	Std	Mean	Std	Mean	Std
0.7&0.2	0.058569	0.007935	0.046426	0.005735	0.051792	0.009372
0.7&0.25	0.048094	0.004691	0.044476	0.008253	0.042911	0.005487
0.7&0.3	0.051614	0.002864	0.047673	0.003609	0.042974	0.005500
0.7&0.35	0.058255	0.003849	0.043782	0.004286	0.044063	0.006042
0.8&0.2	0.052475	0.009998	0.048428	0.009771	0.050493	0.004559
0.8&0.25	0.056541	0.00751	0.042662	0.009311	0.042766	0.005767
0.8&0.3	0.054382	0.004600	0.047286	0.004401	0.041841	0.003568
0.8&0.35	0.048777	0.004184	0.037193	0.005857	0.044227	0.005315
0.9&0.2	0.060118	0.006952	0.045304	0.004967	0.048556	0.009201
0.9&0.25	0.048174	0.009187	0.039070	0.003072	0.046178	0.005843
0.9&0.3	0.055117	0.006072	0.040348	0.004179	0.041602	0.005468
0.9&0.35	0.055986	0.003723	0.043844	0.005933	0.043934	0.006579
1.0&0.2	0.047922	0.012682	0.039615	0.005448	0.056215	0.005706
1.0&0.25	0.058727	0.006360	0.046322	0.008512	0.044346	0.008428
1.0&0.3	0.053426	0.004635	0.044604	0.004828	0.039999	0.004517
1.0&0.35	0.053434	0.003485	0.049197	0.002724	0.039377	0.005088

Table C.21: Exploration of paramters near the best case of LR0.717 & E0.233

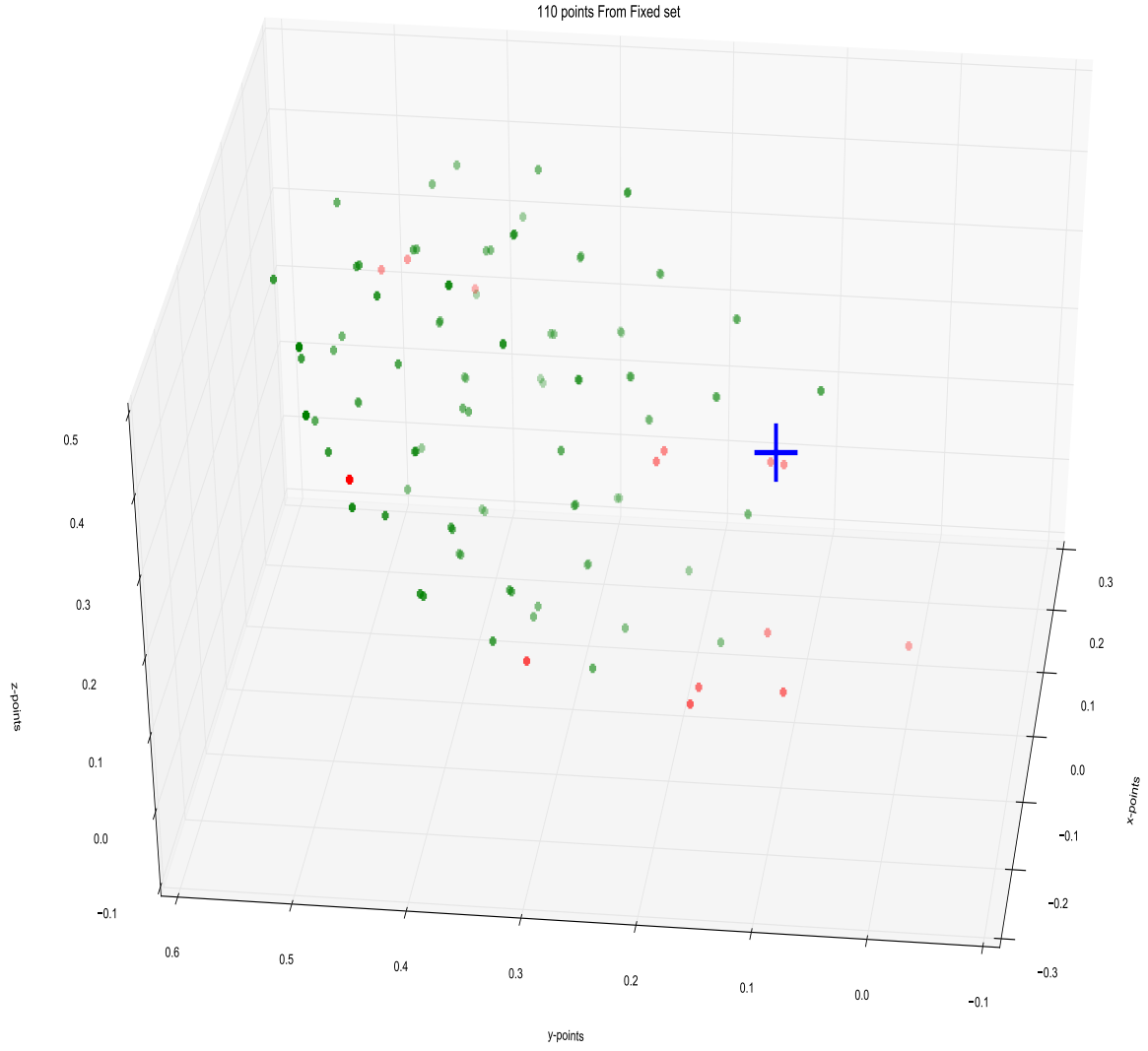


Figure C.1: Fixed points testing points-red shows the points with absolute error more than 4cm and green show less than 4cm. Blue cross show the center of the robot model (LR0.717, E0.233 & 10 Hidden Nodes)

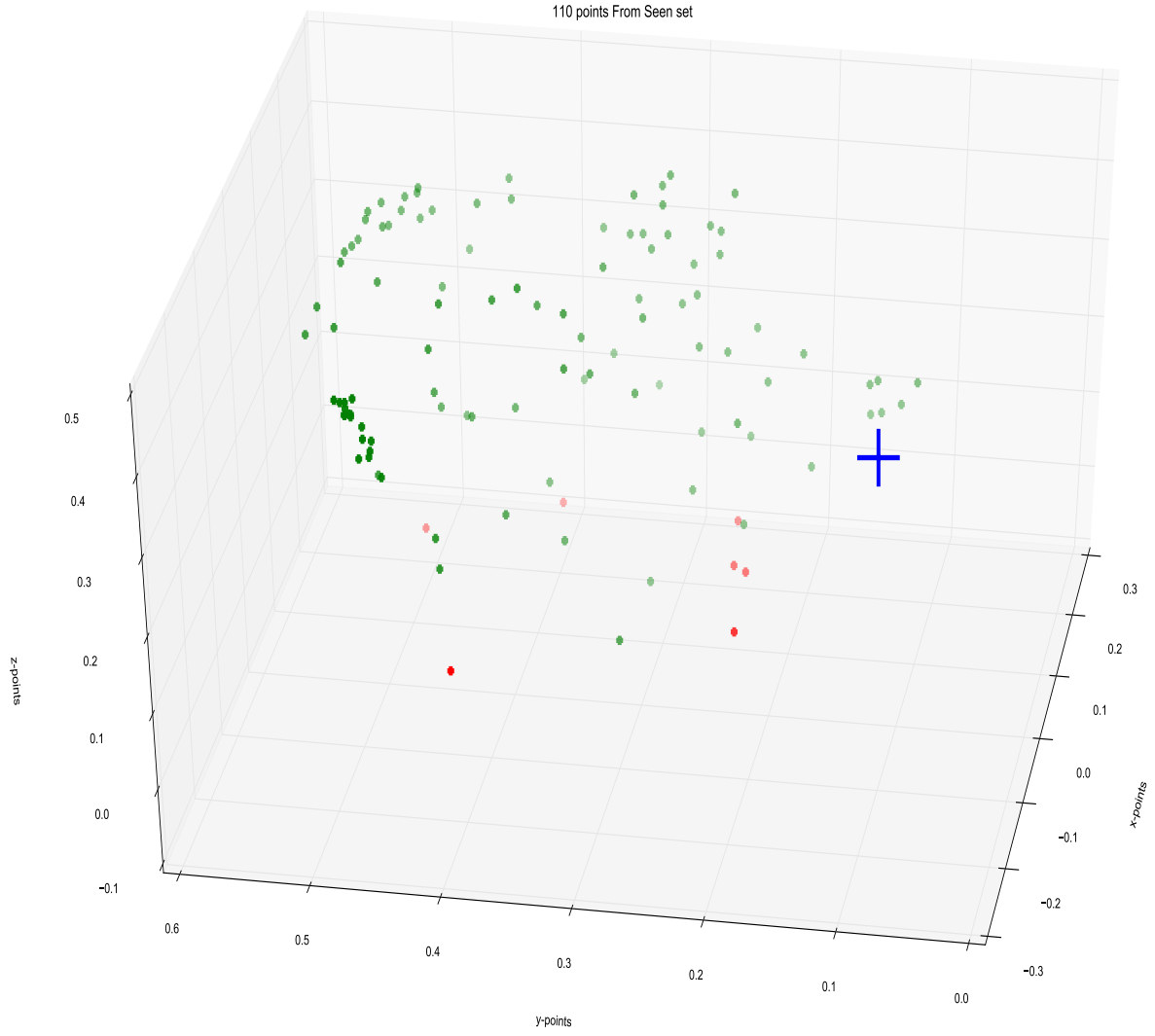


Figure C.2: Seen points testing points-red shows the points with absolute error more than 3cm and green show less than 3cm. Blue cross show the center of the robot model (LR0.717, E0.233 & 10 Hidden Nodes)

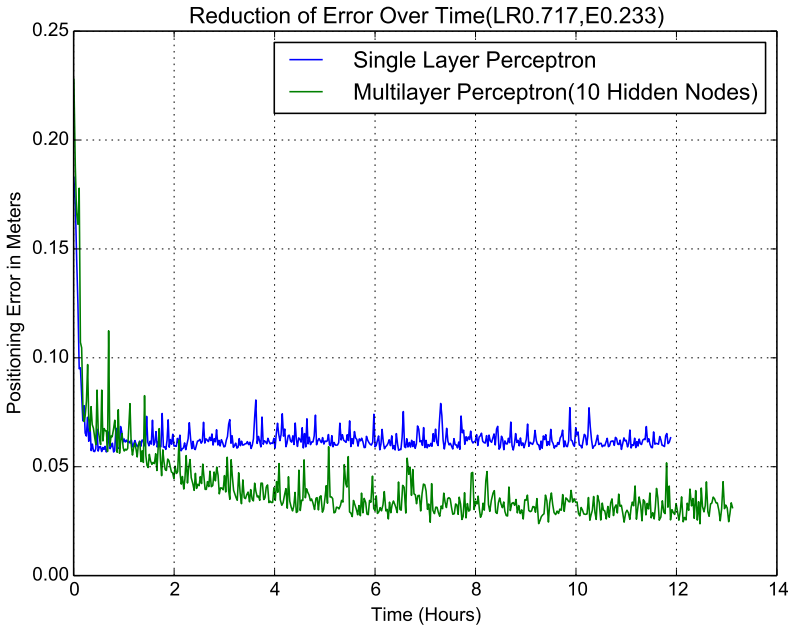


Figure C.3: Reduction of Error Over Time (LR0.717, E0.233 & 10 Hidden Nodes)

Bibliography

S. Agatonovic-Kustrin and R. Beresford. Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727, 2000.

P. Allgeuer, M. Schwarz, J. Pastrana, S. Schueller, M. Missura, and S. Behnke. A ros-based software framework for the nimbro-op humanoid open platform. In *Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE Int. Conf. on Humanoid Robots, Atlanta, USA*, 2013. URL `http://www.ais.uni-bonn.de/papers/HSR13_Allgeuer_ROS_NimbRo-OP.pdf`.

M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida. Cognitive developmental robotics: a survey. *Autonomous Mental Development, IEEE Transactions on*, 1(1):12–34, 2009.

R. Berka. Inverse kinematics-basic methods. *Dept. of Computer Science & Engineering, Czech Technical University, Prague/Czech Republic*. URL `http://www.cescg.org/CESCG-2002/LBarinka/paper.pdf`.

L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL `http://leon.bottou.org/papers/bottou-98x`. revised, oct 2012.

K. Erbatur, U. Seven, E. Taşkıran, and Ö. Koca. Walking trajectory generation and force feedback control for the humanoid robot leg module suralp-l. IASTED, 2008a.

K. Erbatur, U. Seven, E. Taskiran, O. Koca, G. Kiziltas, M. Unel, A. Sabanovic, and A. Onat. Suralp-l-the leg module of a new humanoid robot platform. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 168–173. IEEE, 2008b.

K. Erbatur, U. Seven, E. Taskran, O. Koca, M. Ylmaz, M. Unel, G. Kzltas, A. Sabanovic, and A. Onat. Suralp: a new full-body humanoid robot platform. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4949–4954. IEEE, 2009.

K. Grave, J. Stuckler, and S. Behnke. Improving imitated grasping motions through interactive expected deviation learning. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 397–404. IEEE, 2010. URL `http://www.ais.uni-bonn.de/papers/IEEE_RAS_2010_Graeve_Stueckler_Behnke.pdf`.

A. T. Hasan and H. Al-Assadi. Performance prediction network for serial manipulators inverse kinematics solution passing through singular configurations. *International Journal of Advanced Robotic Systems*, 7(4):10, 2010. URL `http://dx.doi.org/10.5772/10492`.

D. O. Hebb. The first stage of perception: growth of the assembly. *The organization of behavior: A neuropsychological theory*, pages 60–78, 1949.

M. Islam, B. Boswell, and A. Pramanik. An investigation of dimensional accuracy of parts produced by three-dimensional printing. In *Proceedings of the World Congress on Engineering*, volume 1, pages 3–5, 2013. URL `http://www.iaeng.org/publication/WCE2013/WCE2013_pp522-525.pdf`.

J. Konczak, M. Borutta, and J. Dichgans. The development of goal-directed reaching in infants ii. learning to produce task-adequate patterns of joint torque. *Experimental Brain Research*, 113(3):465–474, 1997.

M. Lapeyre, P. Rouanet, and P.-Y. Oudeyer. The poppy humanoid robot: Leg design for biped locomotion. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 349–356. IEEE, 2013. URL `https://hal.inria.fr/hal-00852858/document`.

M. Lapeyre, P. Rouanet, J. Grizou, S. Nguyen, F. Depraetre, A. Le Falher, and P.-Y. Oudeyer. Poppy Project: Open-Source Fabrication of 3D Printed Humanoid Robot for Science, Education and Art. In *Digital Intelligence 2014*, page 6, Nantes, France, Sept. 2014. URL `https://hal.inria.fr/hal-01096338`.

M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental robotics: a survey. *Connection Science*, 15(4):151–190, 2003.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. URL `http://www.aemea.org/math/McCulloch_Pitts_1943.pdf`.

G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM, 2008. URL `https://flowers.inria.fr/mlopes/myrefs/10-neuralnetworks.pdf`.

C. Moulin-Frier and P.-Y. Oudeyer. Exploration strategies in developmental robotics: a unified probabilistic framework. In *Development and Learning*

and Epigenetic Robotics (ICDL), 2013 IEEE Third Joint International Conference on, pages 1–6. IEEE, 2013. URL https://flowers.inria.fr/CMF_PYO_ICDL2013.pdf.

M. Quigley, A. Asbeck, and A. Ng. A low-cost compliant 7-dof robotic manipulator. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 6051–6058, 2011. URL http://people.seas.harvard.edu/~aasbeck/papers/Quigley2011-ICRA-ALowCostCompliant7-DOFRoboticManipulator.pdf.

R. F. Reinhart and M. Rolf. Learning versatile sensorimotor coordination with goal babbling and neural associative dynamics. In *Development and Learning and Epigenetic Robotics (ICDL), 2013 IEEE Third Joint International Conference on*, pages 1–7. IEEE, 2013. URL http://www.cor-lab.de/system/files/ReinhartRolf2013-ICDL.pdf.

M. Rolf. Goal babbling with unknown ranges: A direction-sampling approach. In *Development and Learning and Epigenetic Robotics (ICDL), 2013 IEEE Third Joint International Conference on*, pages 1–7. IEEE, 2013. URL http://www.cor-lab.de/system/files/Rolf2013-ICDL.pdf.

M. Rolf, J. J. Steil, and M. Gienger. Goal babbling permits direct learning of inverse kinematics. *Autonomous Mental Development, IEEE Transactions on*, 2(3):216–229, 2010. URL http://www.cor-lab.de/system/files/RolfSteilGienger-TAMD2010-GoalBabbling.pdf.

M. Rolf, J. J. Steil, and M. Gienger. Online goal babbling for rapid bootstrapping of inverse models in high dimensions. In *Development and Learning (ICDL), 2011 IEEE International Conference on*, volume 2, pages 1–8. IEEE, 2011. URL http://www.cor-lab.de/system/files/RolfSteilGienger2011-ICDL-EpiRob.pdf.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. URL http://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5, 1986. URL http://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf.

M. Schwarz, M. Schreiber, S. Schueller, M. Missura, and S. Behnke. Nimbro-op humanoid teensize open platform. In *Proceedings of 7th Workshop on Humanoid Soccer Robots. IEEE-RAS International Conference on Humanoid Robots*, 2012. URL http://www.ais.uni-bonn.de/papers/HSR12_NimbRo-OP.pdf.

M. Schwarz, J. Stückler, D. Droeschel, K. Gräve, D. Holz, M. Schreiber, and S. Behnke. Nimbro@ home 2014 team description. 2014. URL http://www.ais.uni-bonn.de/nimbro/@Home/papers/TDP_NimbRo_Home_2014.pdf.

J. Stückler and S. Behnke. Dynamaid, an anthropomorphic robot for research on domestic service applications. *Automatika–Journal for Control, Measurement, Electronics, Computing and Communications*, 52(3), 2011. URL `http://www.ais.uni-bonn.de/papers/automatika2010_dynamaid.pdf`.

J. Stückler, R. Steffens, D. Holz, and S. Behnke. Real-time 3d perception and efficient grasp planning for everyday manipulation tasks. In *ECMR*, pages 177–182, 2011. URL `http://www.ais.uni-bonn.de/papers/ECMR_2011_Stueckler.pdf`.

E. Thelen, D. Corbetta, and J. P. Spencer. Development of reaching during the first year: role of movement speed. *Journal of Experimental Psychology: Human Perception and Performance*, 22(5):1059, 1996.

D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003. URL `http://axon.cs.byu.edu/papers/Wilson.nn03.batch.pdf`.