Modeling Upper Body Movement in a Virtual Environment using a Neural Network

A thesis submitted to the University of Manchester for the degree of MSc in Advanced Computer Science in the Faculty of Science

2002

Stephan James Dale

Department of Computer Science

Contents

List of Figures

List of Tables

Abstract

The realistic animation of humans in a virtual environment is a complex problem. Modern applications require a great deal of interaction, and the animation of a user is expected to be realistic and accurate. This is particularly true of multi-user collaborative applications, where a user must react to the movements of other users in the environment.

Short of tracking the entire body, an approach that is often not viable, the movements of certain parts of the body must be approximated. Traditional approaches are either too slow due to high computational overheads or unrealistic due to the constraints that have to be imposed for real-time performance. This project explores the use of Neural Networks. After an initial training process, they are small, fast, and can model the dynamics of physical systems with a great deal of success.

Sensors track the movement of the user's head, hands and torso. A neural network is used instead of a sensor to predict the orientation of the user's upper arm. Finally, an inverse kinematics based algorithm corrects any inaccurate predictions to ensure that the user's hands are always in the correct place. The results show that after suitable training, a neural network can predict the movement of the arm to a high degree of accuracy.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or institute of learning.

Copyright

Acknowledgements

The work described in this project was supervised by

Dr Roger Hubbold,
Advanced Interfaces Group,
Department of Computer Science,
University of Manchester.

I would like to thank all members of the Advanced Interfaces Group for their support.

Chapter 1

Introduction

This chapter introduces the project. It provides an overview of virtual humans and discusses relevant previous work.

1.1 Overview

Virtual humans are representations of humans in a virtual environment. There is a huge range in the "virtual fidelity" of such representations [2]. Human appearance, movement and behaviour are modeled to different degrees depending on the requirements of the application. A virtual human can be a character with computer controlled movements and behaviour [31], or simply an avatar animated by tracking the user's movement [25]. In either case, the high degree of interaction required by most virtual reality applications mean that an accurate representation of movement is essential. The user's hands must appear in the correct place in order to manipulate objects, their limbs should have a smooth realistic motion and avoid collision [24], and their body must travel in the correct direction at the correct time. This is particularly true of multi-user collaborative environments, where a user must react to the movements of other users in the environment. If the movement of a user's avatar does not match their actual movement then effective collaboration will be difficult. A good example is the simulation of a stretcher evacuation [16], where two people are carrying a third person on a stretcher. In this case, the movement of one

user directly influences that of the second because they are "connected" by the stretcher.

The creation of a virtual human requires a model of the body and some method of controlling its movement. The model is usually skeletal, so consists of many parts connected by joints, see figure 1.1. Each joint has several degrees of freedom (DOF). The range of motion possible with even a simple model is vast, making realistic animation a challenging task. With the added requirement of real-time animation, few systems are capable of producing movement that can go undetected by the human eye as computer generated.

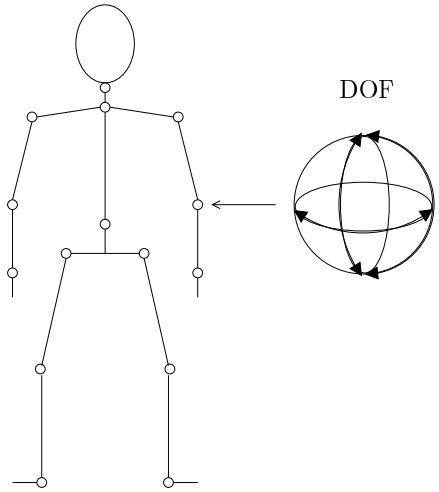


Figure 1.1: Simple human body model.

The most straight forward way of animating human movement in real time is to track the entire body. There are several reasons why this is not always practical:

- The hardware needed to track the entire body is expensive and above the budget of most educational establishments.
- Cheaper tracking systems use cables that restrict movement. The more sensors attached to the body, the more cables there are to avoid.
- An increase in the amount of sensors leads to an increase in the required bandwidth and hence an increase in hardware needed to cope with the requirement.

- Outside-in systems that use cameras and complicated image recognition software have a large computational requirement that could be used rendering the virtual environment or performing other important calculations.

This project researches a new method of reducing the amount of sensors required to track the body by using an Artificial Neural Network (ANN) to approximate the readings of certain sensors, thereby replacing them. This will reduce the cost of the setup by decreasing the amount of equipment needed, increase the speed at which a user can start using the virtual environment and increase their freedom of movement by reducing the number of sensors attached to their body.

1.2 Project Description

The project aims to produce a system capable of real-time animation of the upper body, with the hands and torso tracked using sensors and the upper arm positioned by a neural network. Setup and use of the system should be quick and easy. Overall, I aim to produce an "upper body tracking system in a box", where a user can quickly attach some sensors to their body, start the program, and see their accurately animated avatar.

A neural network must be found that is capable of the real-time prediction of upper arm orientation. The project will explore different neural networks and provide estimates of their performance when applied in a range of mock real-world situations. Ideally, a network will be able to predict upper arm orientation for a range of different tasks and users. Its accuracy should be within the range of human tolerance to error, initial estimates of which have been determined by A. Beeharee [3].

1.3 Previous Work

There are many approaches to virtual human animation [30]:

- **Motion Capture** [6, 25] gathers data from an actor performing set motions. The data can be used to create very life-like movements but can not be easily generalised to create new ones. Motion capture data is often used with other techniques, for example to create the poses required by key-framing.

- **Key-framing**[1] [23] and other interpolation based techniques [32] rely on an animator to set the location of various control points on the virtual human to create a number of poses, or key-frames. The animation is created by interpolating movement between these key-frames. Key-framing can create very complex motion but requires an experienced animator to define a set of poses and the transition between them. Motion capture data can be used to help create the poses.

- **Dynamic Simulation** [34] uses complicated models to simulate the dynamics of a physical system, in this case the dynamics of human movement. This technique creates physically accurate movement useful for applications that require the virtual humans to respond to a changing environment. The complexity of the model prevents real-time performance.

- **Forward Kinematics** takes the joint parameter space of a linked structure and computes the position of its end-effector in Cartesian space. **Inverse Kinematics (IK)** [33, 8] does the opposite of this and computes the joint parameters from the position of the end-effector. A set of constraints govern the range of possible motion. As with dynamic simulation, the movements are physically sound. Forward Kinematics is good for defining complex motions but error propagation makes it difficult to reliably position an end-effector on a target. IK has found more application in systems that require this goal-oriented motion. It is a good approach for modelling periodic motion, for example walking [19, 9]. However, other more complicated movements, such as that of the arm, can appear "mechanical".

[1] In full, this approach is known as *Parametric* Key-frame Animation and is not to be confused with the 2D key-framing used by the cartoon industry.

Applications will often use more than one approach to create the desired effect. For example, the AGENTlib environment integrates Motion Capture, Key-framing, Dynamic Simulation and Inverse Kinematics to animate an avatar in real-time [7].

1.3.1 Neural Networks

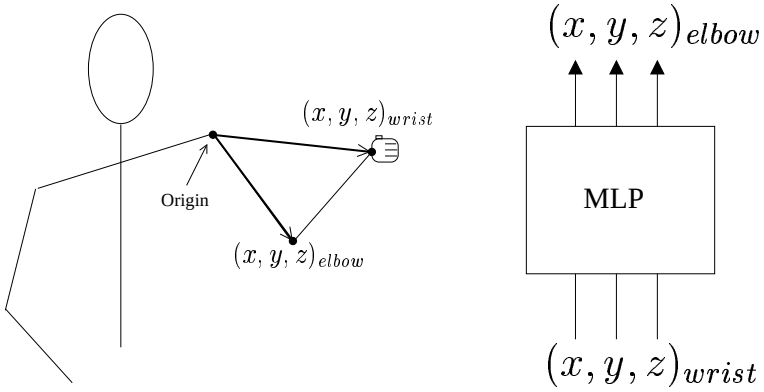
Neural networks have been used successfully in a wide range of applications such as gesture recognition [20] and robot control [26]. The application of neural networks to virtual human animation is recent, but it is a promising technique that has produced good results from the few people that have done so. For those unfamiliar with neural networks, suffice to say that they consist of an interconnected network of simple processing units, resulting in a system capable of learning highly complex tasks. At this stage they can be thought of as a "black box" that accepts a number of inputs and produces a number of outputs. A full description of the neural network used in this project is given in chapter 3.

One example of the use of neural networks is the NeuroAnimator system[12, 11] which uses a network to emulate physics-based models. Essentially they encode the model in the network to produce a system that no longer has to perform the complicated time-consuming calculations required by the model. Their results show that the network performs well. However, the movements that the network generates will have the same properties as the model it emulates, and hence they will not be as natural as one would hope for.

The use of a neural networks for real-time avatar control is somewhat different. In this case the neural network serves as more of a sensor replacement. A neural network learns the position of a particular body part given the position of other body parts. Literature on this application of neural networks is sparse. I will discuss two papers, "Enhanced Avatar Control Using Neural Networks" by H. Amin and R. Earnshaw [1], and "Real-time Avatar Control with a Neural Network" by A. Beeharee and R. Hubbold [4]. Both approaches use a common form of neural network, the Multi-layer Perceptron (MLP) (described in section 3.1.2).

Amin & Earnshaw

Amin & Earnshaw's method uses a MLP whose inputs are the position of the wrist and outputs are the position of the elbow, both in Cartesian space and relative to the shoulder, see figure 1.2. The lengths of the avatar's arms are initially unconstrained. The neural network predicts the position of the elbow, and a real-time algorithm called SHAKE, based on a subset of IK, corrects any network predictions that would lead to inaccurate arm lengths.



(a) Neural Network

(b) Inputs and outputs

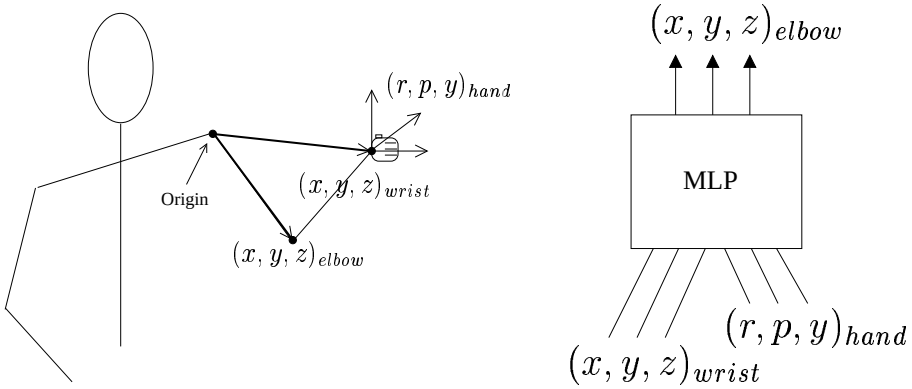
Figure 1.2: Amin & Earnshaw's approach.

Their results are good, showing that the network can produce arm movements that are extremely close to the actual movements of the user. However, whilst their network returns the position of the elbow in Cartesian space they use this information to model the orientation of the elbow with respect to the position of the end-effectors of the arm. This is a simpler problem than modelling the position of the elbow.

Beeharee & Hubbold

Beeharee & Hubbold's paper is taken from work by Ashwin Beeharee in his research for his Masters degree [3], which in turn built upon previous work carried out by Alan Beverage in 1998 [5]. They used a larger MLP. They also

use a network to output the position of the elbow. However, unlike Amin and Earnshaw's approach the inputs consist of the rotation of the hand as well as the position of the wrist, see figure 1.3. The larger network size is perhaps due to the increased complexity of the task due to the inclusion of hand rotation.



(a) Neural Network

(b) Inputs and outputs

Figure 1.3: Beeharee & Hubbold's approach.

Their preliminary studies found that humans can tolerate errors of up to 2 inches in the position of the elbow. Their results show that the neural network can make predictions within these limits.

1.4 My Approach

My approach is similar to the both of the neural network techniques previously mentioned. I have adopted what I feel to be the best properties from each.

Beeharee & Hubbold's approach uses hand rotation as inputs into the network, as well as wrist position. The use of hand rotation has obvious advantages in dealing with movements that require rotational movement of the hand. This rotation plays a large part in the position of the elbow. You only need to look at the pouring action to see that rotation of the hand results in change in elbow position, as illustrated in figure 1.4. Because of the influence on arm position, I felt it was necessary to include hand rotation as well as wrist position in

the network inputs. It means that the network will learn a greater range of actions, which in turn means that the network can be used for a greater variety of tasks.

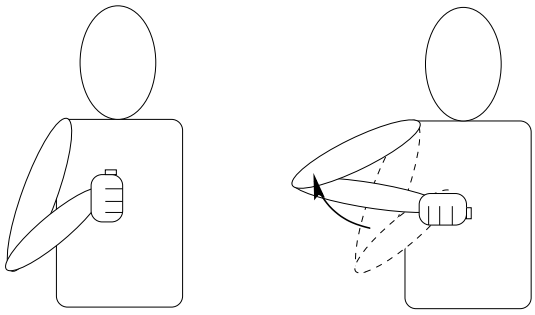


Figure 1.4: Pouring action.

Both approaches use a network to give to position of the elbow in space in *Cartesian* co-ordinates. However, since the users dimensions are known (they are needed to normalise the data for the network, see section 2.5.1) then the position of their elbow can be specified using rotation by pitch and yaw, requiring only a simple calculation to map from orientation to position (described in section **??**). This means that the network will have to learn a mapping to only 2 rather than 3 outputs. Hopefully the simplification of the problem will lead to a smaller network and hence a decrease in the computation required.

In neither approach does the rotation of the torso directly effect the data that the network is expected to learn. The position of the hand is calculated relative to the position of the shoulder in the world coordinate system. This means that the user is expected to keep their torso rigid. For instance, consider a person who first holds their hand in front of their chest, then rotates at the waist. The position of their hand relative to the shoulder in world co-ordinates changes, but their upper body pose does not. I calculate the position and orientation of the hand, and the orientation of the upper arm, relative to the torso's co-ordinate system. This means that the user is free to move their upper body as they wish.

In summary, the neural network I will use will take hand position and orientation, and produce outputs that give the orientation of the upper arm (specifically pitch and yaw). The torso is also animated. Measurements of

hand position and orientation, and the upper arm orientation, is relative to the torso's co-ordinate system. The method is described in detail in chapter 2.

Chapter 2

My Approach

My approach feeds the hand position and orientation to a neural network which predicts the orientation of the user's upper arm. All measurements are relative to the torso's co-ordinate system. A basic Inverse Kinematic algorithm ensures the hand is in the correct place regardless of the networks prediction of arm orientation. This chapter will describe the details of this implementation[1].

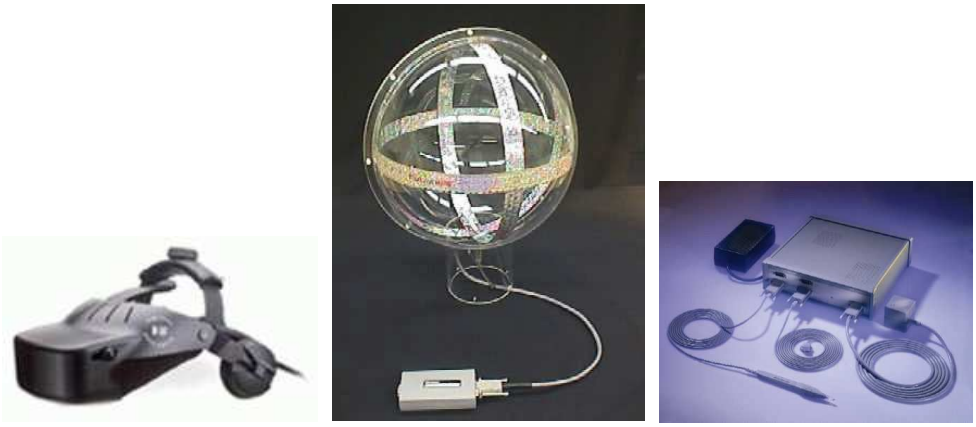
The code was written in C and used the MAVERIK virtual reality libraries [14, 15] and Jeff Shufelt's backpropagation library [28].

2.1 Set-up

2.1.1 Hardware

Figure 2.1 shows the hardware used in the project. It consists of the Virtual Research Systems V8 Head Mount Display (HMD), a Polhemus Long Ranger emitter and several spatial sensors (receivers), connected to a Polhemus Fast-Track tracking system. In total, 5 sensors are available but only 4 can be tracked at any one time. One is mounted on the HMD to track head movement, two are coupled with Division 3D mice to track hand movement, one is fixed on a velcro strap for attaching to the upper arm, and one is fixed to a belt so that it can be attached to the chest

[1]It is assumed that the reader has knowledge of basic matrix and vector arithmetic.



(a) V8 HMD (b) Polhemus tracker (c) FastTrack tracking system

Figure 2.1: Equipment.

2.1.2 User Set-up

User Measurement

Before the sensors are attached to the body, the user's shoulder width l_s, upper arm length l_{ua}, and lower arm length L_{la} are measured, as shown in figure 2.2. These are needed to scale the avatar to meet the user's proportions (section 2.12) and to normalise the data that will be presented to the neural network (section 2.5.1). The measurements are stored in a file so that they can be easily loaded when required. The users dimensions are tabulated in appendix C, section C.1.

Attaching Sensors

Figure 2.3 shows the set-up of the equipment when gathering data, and figure 2.4 shows the set-up for normal use when the neural network determines the elbow position. The user sits facing the Polhemus emitter. For data gathering (section 4.2.3), the user wears the HMD, a torso sensor mounted on their chest, a 3D mouse in their right hand, and a sensor attached to their upper arm. For normal use of the system, when the neural network is predicting upper arm

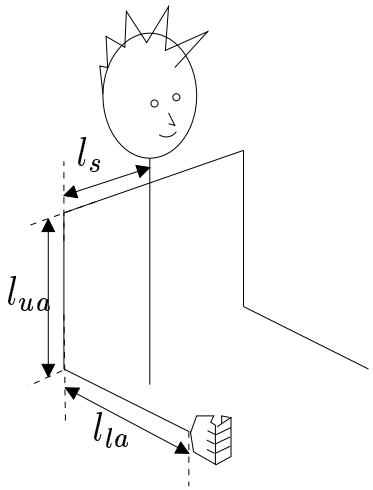


Figure 2.2: User measurements.

orientation, the user replaces the upper arm sensor with another 3D mouse held in their left hand to allow both arms to be animated.

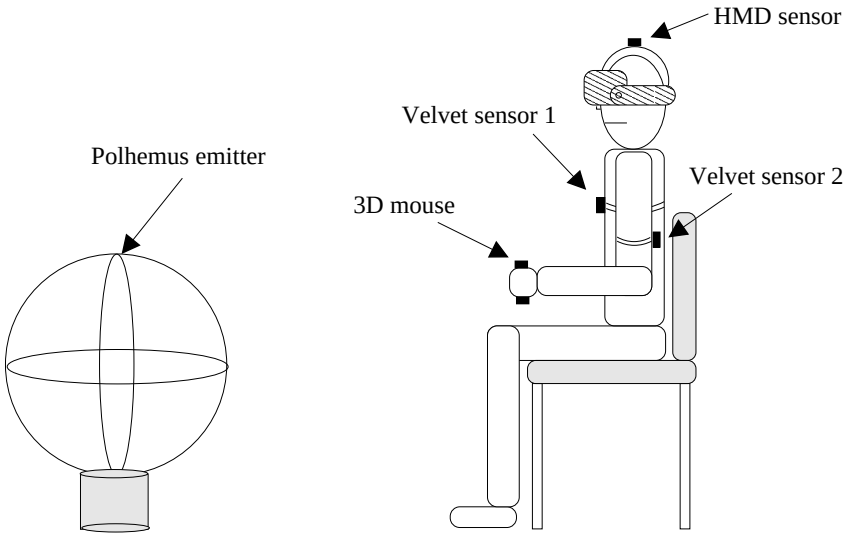


Figure 2.3: Equipment setup for data gathering.

Calibration Range

In practice the effective range of the Polhemus tracker covers and an area of about 2^2 meters. It is possible to move out of range, in which case the movement of the avatar freezes. To minimise the likelihood of this it is necessary

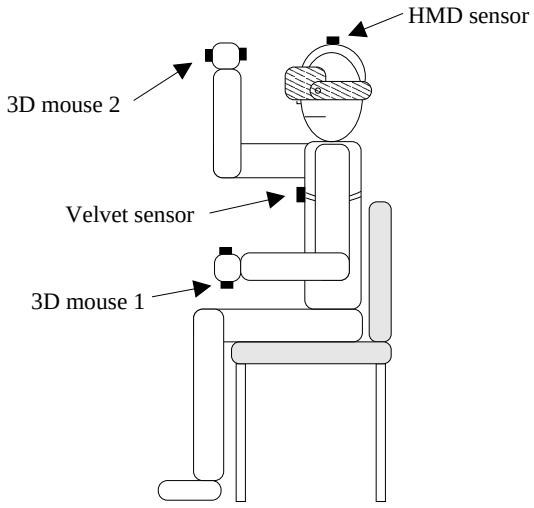


Figure 2.4: Equipment setup for normal use.

to define a set location where the user must be seated (see figure 2.5). During data gathering (section 4.2.3) the participants were told that if they witness the avatar freezing they should move back towards their starting position until the avatar becomes animated again, and that they should avoid that area of space in the future. This rarely occurred, and was only a problem for the taller subjects (the tallest of which was 6'5"). People of average height experience no problems unless they stretch to reach an object, or move their arm below the seat of the chair. To help prevent this the participants were given specific instructions for use. Additionally, the data preparation stage removes any readings that are in any way anomalous (see section 4.3.3).

The accuracy of the tracker within the calibrated area is roughly ± 2 centimeters.

2.2 User Integration in the Virtual Environment

2.2.1 View

The user has a choice of viewpoints. In use the user is free to switch between viewpoints as desired, but when gathering data as realistic a view as possible

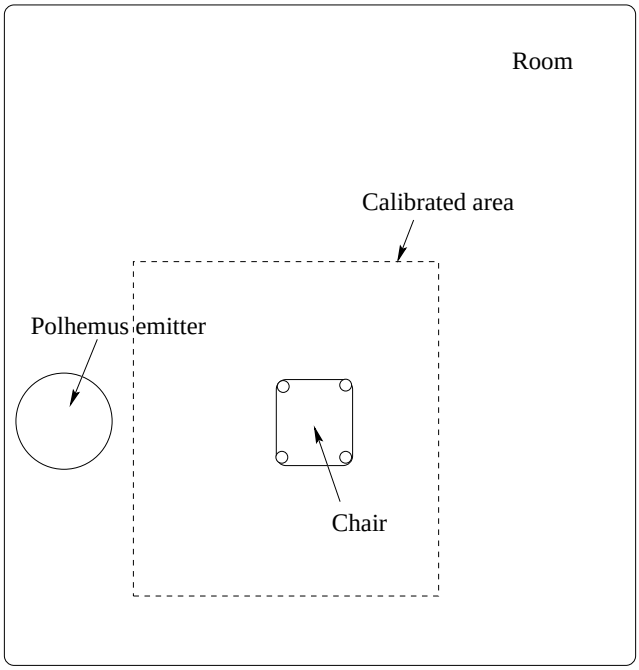


Figure 2.5: Chair position

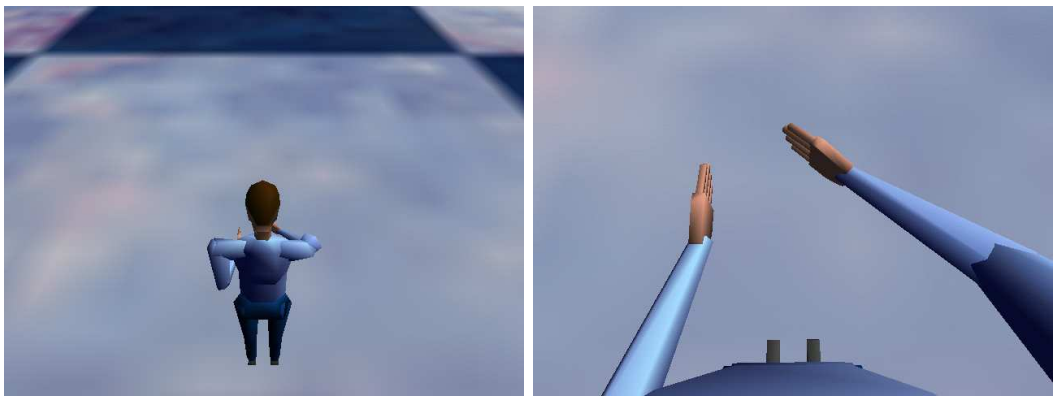
is desired hence they see from their virtual eyes[2]. Figure 2.6 shows two of the possible views that were provided, the latter used when gathering data from the participants.

2.2.2 Control

The user can move about the environment by pressing buttons on the 3D mouse. The right hand controls forward/backward motion, and the left hand controls left/right motion. User control of avatar motion was disabled during data gathering so that all users remained in the same position in the virtual environment.

The user can grasp objects with either hand by pressing and holding a button on the 3D mouse. Only a single object can be held in each hand at any one time.

[2] Although not exactly - the eyes are seen to be fixed points, as with most VR applications.



(a) View fixed above avatar (b) View from eyes

Figure 2.6: Possible views.

2.2.3 Other Considerations

The environment provided by VR is not accurate to the real-world. For this reason an amount of inaccuracy can be introduced to the environment without the user noticing [27].

It is relatively strenuous to be immersed in the environment for long lengths of time, for various reasons;

- Prolonged use of the HMD and the 3D mice can be tiring because of their weight.
- The user is looking at a screen that is positioned very close to the eyes and displays with a low frequency and low resolution.
- The user is essentially being "tricked" into seeing 3 dimensions, a process that does not present the user with the exact images and focus that would be experienced in the real-world.

2.3 Program Cycle

The program has the structure shown in figure 2.7. The sensors are continuously polled to get their position and orientations. From this data, the

position and orientation of the hands can be calculated relative to the torso's co-ordinate system (section 2.4). A neural network predicts the rotation of the upper arm (section 2.5.1) which is corrected by a basic IK algorithm to ensure that the hands are always in the correct place (section 2.5.2).

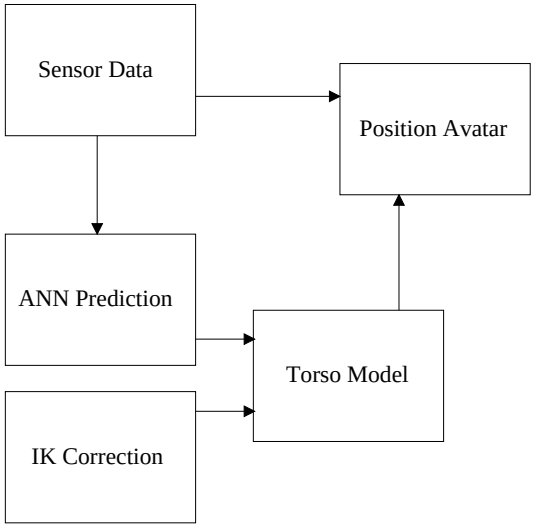


Figure 2.7: Program structure

2.4 Mapping from Sensors to Avatar

2.4.1 Avatar

An avatar is a virtual representation of the user's body. MAVERIK provides a simple human avatar that, with some extensions, is adequate for my research. I did not want to modify the avatar or any of the MAVERIK code behind its implementation, but rather produce a program that uses the library in its release form. If necessary my code can be later integrated into the MAVERIK library.

Structure

The avatar is built up of 19 parts that approximate a male human body. The parts are of simple form as they are not required to be accurate representations

of the body, but to provide a simple representation of a human for populating virtual environments. This structure of the avatar is shown in figure 2.8.

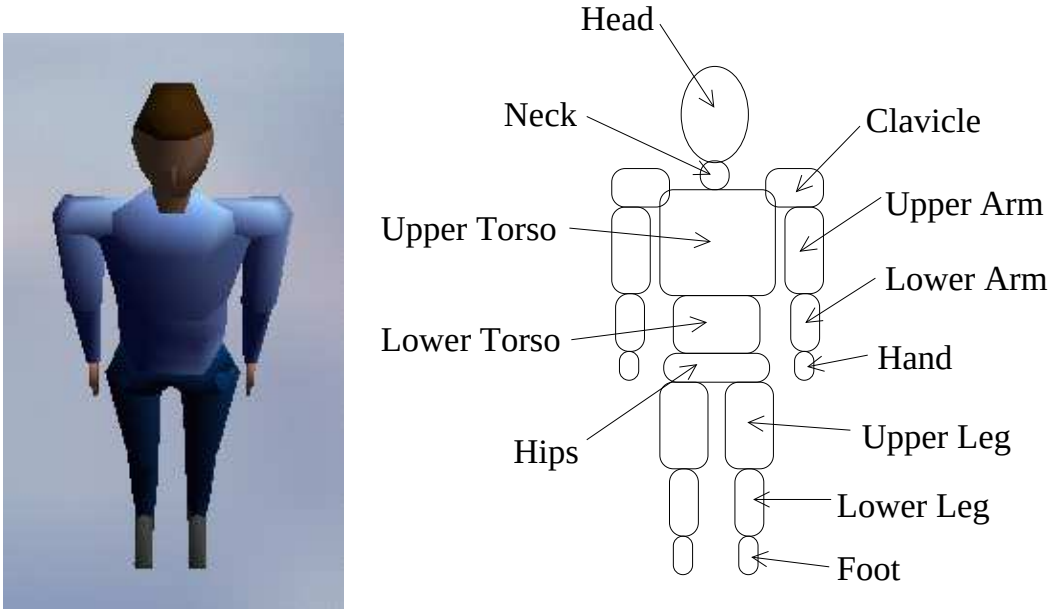


Figure 2.8: MAVERIK avatar

The parts are relative to one another, with the root at the hips. So the position and orientation of the lower torso is relative to the hips, the upper torso is relative to the lower torso, the neck is relative to the upper torso, and so on.

Each part has associated with it 2 matrices. One specifies the part's position and the other specifies its orientation. The co-ordinate systems of some avatar parts are shown in figure 2.9, which illustrates the co-ordinate systems of the right upper arm and the hands. The direction of the axes are the same as those of the upper arm for every part but the hands. Of course, the origins of the parts differ.

2.4.2 Offsets

Because the sensors are positioned on the surface of the users body, and often displaced from the exact point that they represent, their positions require

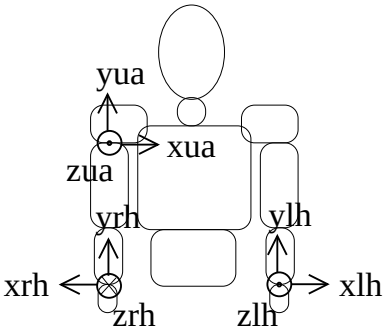
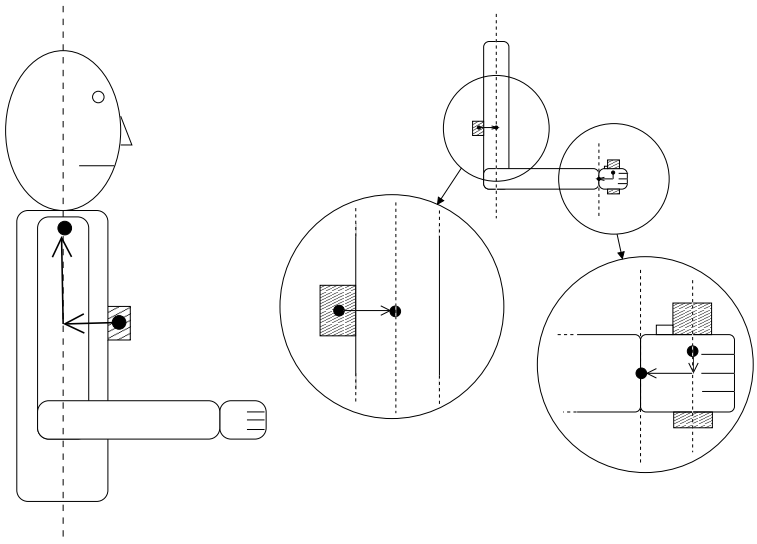


Figure 2.9: Avatar part co-ordinate systems.

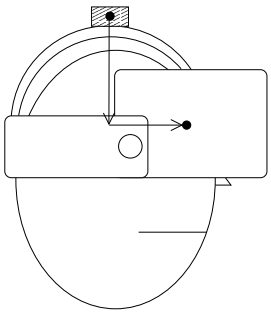
offsetting. So, for each sensor there is an associated offset, as shown in figure 2.10. These map from the polled sensor positions to their corresponding positions on the torso model (section 2.4.3). They have been kept as simple as possible to reduce the scope for error. The full mathematical details of the sensor offsetting are given in appendix A, section A.3.

The torso, arm, and hand offsets are most important in terms of the experiments, as they directly affect the data that is used to train the neural network. The torso and arm offsets are measured for each user. It is sufficient to measure the hand offset only once, and apply it to all users, as it simply moves the position of the sensor from the center of the 3D mouse to the user's wrist. The difference in this measurement between users is so small that an accurate measurement of its change is not practical or indeed possible.

The head sensor offset is of least importance. The position of this sensor gives the position of the users eye-point, and so an accurate measurement of this offset leads to an accurate view in VR. The offset does not directly effect the values of the data gathered but it can effect the way the user performs a task and so it can still influence the data. The user must feel comfortable with the view. It was found that the offset could be the same for most users, with a small amount of tweaking necessary if the user felt the view was unnatural. To ensure that the view was relatively accurate the user was asked to look at their shoulders, arms, and torso.



(a) Torso sensor (b) Upper arm sensor



(c) HMD offset

Figure 2.10: Sensor offsets

Sensor Movement

One problem encountered with the attachment of sensors to the body is that of sensor movement. This can occur due to movement of the muscle and skin surrounding the bone, and due to movement of clothes. This introduces some error. The positions of attachment on the body were chosen to limit the amount of such movement.

2.4.3 My Upper Body Model

The calculation of an upper body model requires the measurement of the user's shoulder width l_s, upper arm length l_{ua}, and lower arm length l_{la}. The process described in section 2.4.2 gives us the positions and orientations of the users body parts. The vector representing the shoulder can be determined from the position and orientation of the torso and the width of the shoulders, and the vectors representing the users arms can then be calculated using the position of the hand and upper arm and the length of the users upper and lower arm. Figure 2.11 shows the steps in calculating the model.

All measurements are relative to the torso, the origin O is the (offset) position of the torso (x_t^o, y_t^o, z_t^o), and the world axes X, Y, and Z are the axes of the torso X_t^o, Y_t^o, and Z_t^o. The position and orientation of any given part is translated relative to the torso by multiplying the inverse of the torso's matrix $[M_t^o]$ by the part's matrix $[M_{part}^o]$ as follows,

$$[M_{part}^{relTorso}] = [M_t^o]^{-1}[M_{part}^o]$$

Step 1: The position of the users shoulder S is simply a point the distance l_s along the x-axis in the negative direction, where l_s is the length from the point in-between the users shoulders to their right shoulder,

$$S = (-l_s, 0, 0)$$

Step 2: Subtracting the position of the shoulder S from the position of the

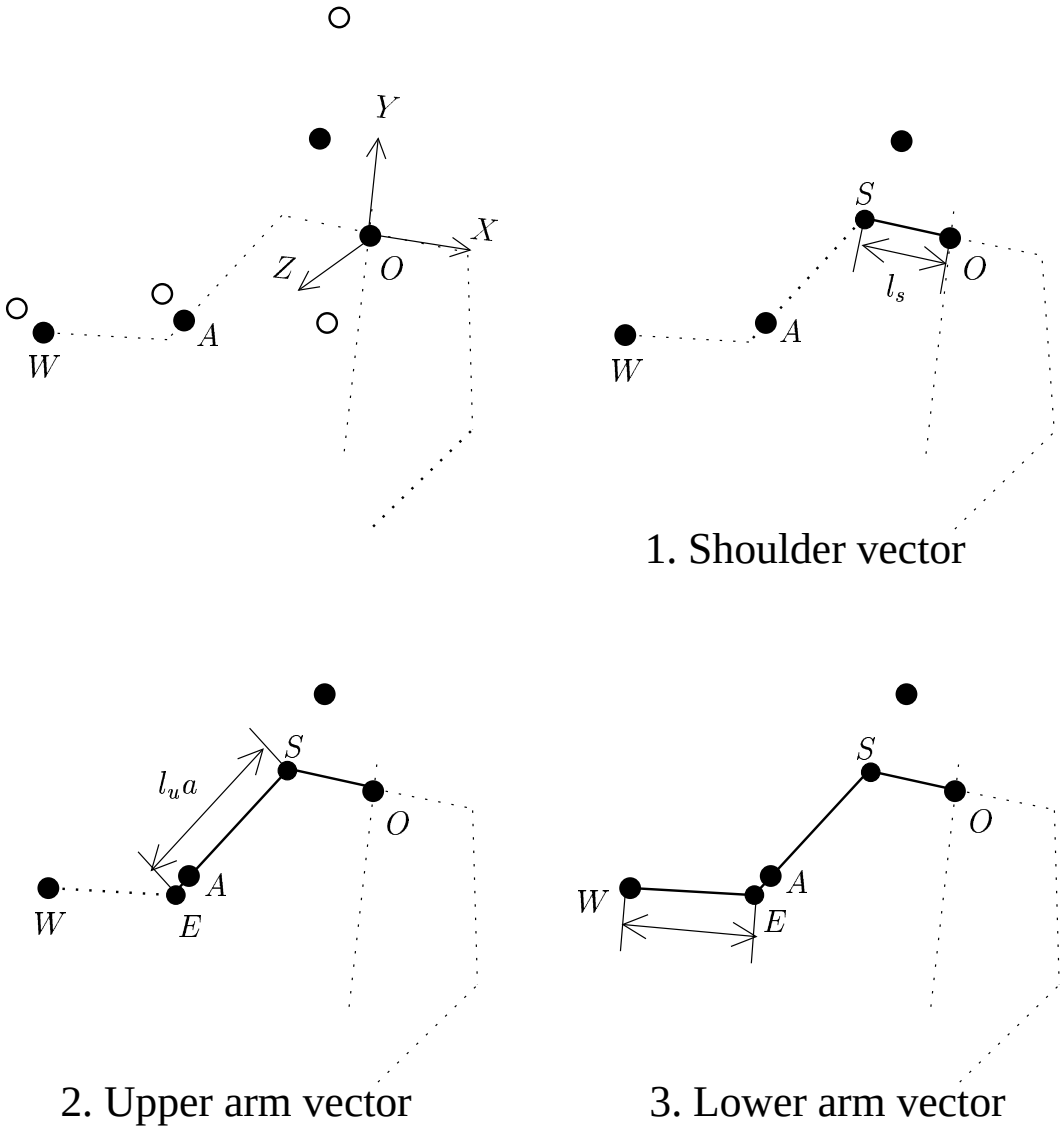


Figure 2.11: Torso model.

upper arm A gives us the vector representing the upper arm SA,

$$SA = A - S$$

This is extended so that its magnitude is the length of the users upper arm l_{ua}, and shifted along the shoulder, to give us the position of the elbow E,

$$SE = \frac{SA}{|SA|} l_{ua}$$

$$E = SE + S$$

Alternatively, if the upper arm sensor is not attached, and the neural network has "replaced" it, E is instead determined by the neural network prediction and the IK correction algorithm. This is described in section 2.5.

Step 3: The lower arm vector is simply calculated by subtracting the position of the elbow from the position of the hand,

$$EH = H - E$$

2.4.4 Mapping from model to avatar

The model must be continuously mapped onto the avatar to produce real-time movement. A number of problems arise with this task. Firstly, different users have different dimensions that the avatar must be scaled to meet. Secondly, the MAVERIK avatar has parts representing the clavicles while the upper body model I use does not. The desired result is shown in figure 2.12.

Scaling

The default proportions of the avatar only approximate human dimensions. Scaling of the avatar to meet the users proportions is necessary so that the user sees a representation of their body that is as accurate as possible. Some users will have proportions that are closer to those of the avatar than others.

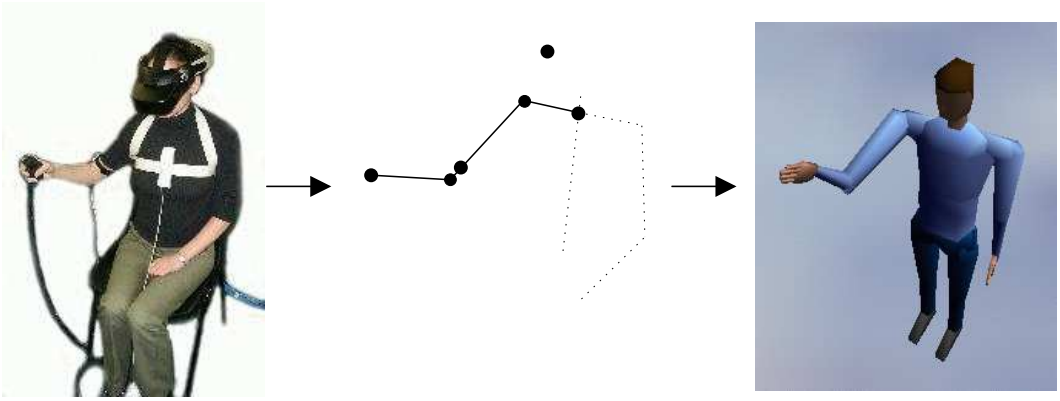


Figure 2.12: Mapping from model to avatar.

For instance, the avatar represents a male human body, so female users will be less accurately represented than male users.

For this research the only parts that need to be scaled are those that correlate with my torso model, namely the shoulders, upper arm, and lower arm. The legs are not in use, but they are scaled in proportion to the torso so that the user is a reasonable distance off the ground and the proportions of the whole avatar do not look too unusual. The users' dimensions and a full description of the scaling process are given in appendix A, sections A.1 and A.2.

Clavicle Removal

The representation of the upper body I am using is very simple. It consists of a single vector representing the shoulders, directly joined to vectors representing the upper arms. The MAVERIK avatar has clavicle parts, and since the parts are relative to one another, the orientation of the avatar's upper arm must be specified relative to the corresponding clavicle. This means that the upper arm orientation of my model must be transformed to give an equivalent orientation for the MAVERIK avatar. This is simply a matter of transforming the position of the elbow E by the inverse of the matrix $[C]$ that defines the position and orientation of the clavicle relative to the torso,

$$E' = E[C]^{-1}$$

E' is then used to determine the orientation of the upper arm part, as described in section 2.5.

2.5 Arm Positioning

When in use, the position of the elbow is predicted by a neural network rather than being determined using the position of the upper arm sensor. The network is given data derived from the model defined in section 2.4.3 as inputs, and outputs the orientation of the upper arm. A simple IK algorithm then corrects the predicted orientation to ensure that the user's hand appears in the correct place.

2.5.1 Neural Network Prediction

Inputs

As inputs, the neural network accepts the position and orientation of the user's hand relative to the torso's co-ordinate system, shown in figure 2.13. The position of the hand H has already been determined in the calculation of the model, section 2.4.3. The orientation is determined by first getting the hand's matrix relative to the torso,

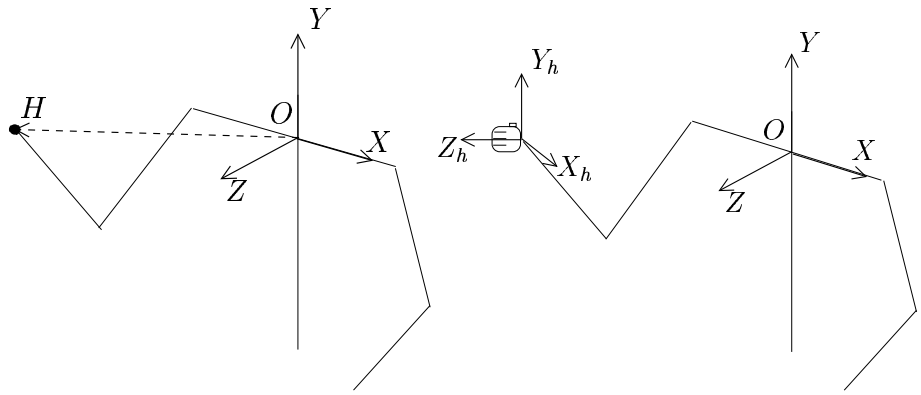
$$[M_h^{relTorso}] = [M_t^o]^{-1}[M_h^o]$$

and then querying its rotational components θ_X, θ_Y, and θ_Z.

Outputs

The network outputs the pitch and yaw of the upper arm relative to the Y and X axes of the torso's co-ordinate system, figure 2.14. The pitch α is simply defined as,

$$\alpha = \arccos(-Y \bullet \widehat{E})$$



(a) Vector from back to hand (b) Hand orientation

Figure 2.13: Network inputs.

The calculation of yaw is more complicated. First, the vector E representing the upper arm must be reflected onto the XZ plane. This is done by setting its y component to zero,

$$E^{ref} = (E.x, 0, E.y)$$

Yaw β is then determined by,

$$\beta = \arccos(X \bullet \widehat{E^{ref}})$$

Next, the direction in which to rotate needs to be determined (i.e. β's sign). First a normal to X and E^{ref} is calculated,

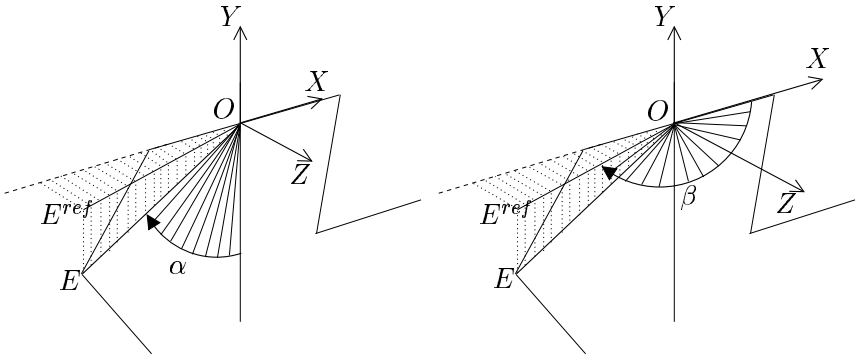
$$norm = X \times \widehat{E^{ref}}$$

If this normal is at 180° to Y then the yaw must be negative. This can be determined by calculating the dot product between $norm$ and Y, giving us the cosine of the angle between them,

$$\cos(\theta) = \widehat{norm} \bullet Y$$

If $\cos(\theta) = -1$ then *norm* is 180° to Y and the yaw should be negative, otherwise yaw should be positive,

$$\beta = \begin{cases} -\beta & if \quad \cos(\theta) = -1 \\ \beta & otherwise \end{cases}$$



(a) Pitch (b) Yaw

Figure 2.14: Network outputs.

Note that a simpler specification of upper arm orientation could have been used. The use of this rather unusual specification was very necessary to allow the network to learn the task adequately, as described in section 5.3.1 when the need to redefine the initial specification became apparent.

The network as a "black box" is shown in figure 2.15.

Data Pre-Processing

To allow the network to predict the upper arm orientation of different users, regardless of their proportions, the data must first be normalised. This is because the rotation of the upper arm (pitch and yaw) has the same range for all users (0° to 360°), while the range of the position of the hand will differ for different users with different arm lengths. For example, when the users arm is held straightened, out to their side as shown in figure 2.16 (pitch -90°, yaw 90°), taller users could have a hand vector H of magnitude 80 cm, while shorter users could have a hand vector of magnitude 70 cm.

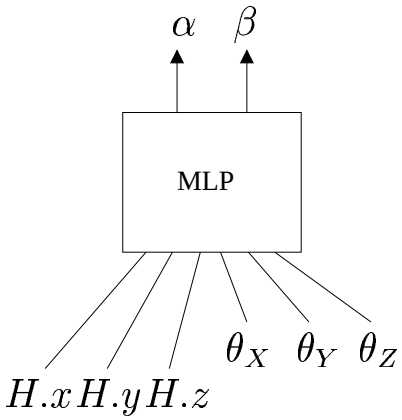


Figure 2.15: The neural network as a "black box".

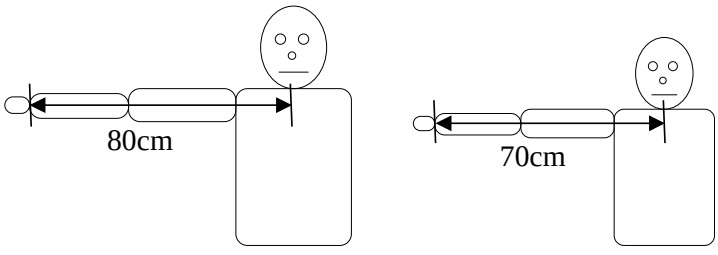


Figure 2.16: Example orientation.

First, the maximum possible magnitude of H is determined from the measurements of the users shoulder width l_s, upper arm length l_{ua}, and lower arm length l_{la},

$$max = l_s + l_{ua} + l_{la}$$

This is used to scale H so that it lies in the range -1 to 1,

$$H^s = \frac{\left(\frac{|H|}{max}\right)}{|H|}$$

where $-1 \leq H^s \leq 1$.

The neural network requires inputs in the range 0 to 1, so H^s must be mapped accordingly,

$$H^n.x = \left(\frac{H^s.x}{2}\right) + 0.5$$

$$H^n.y = \left(\frac{H^s.y}{2}\right) + 0.5$$

$$H^n.z = \left(\frac{H^s.z}{2}\right) + 0.5$$

the result of which is $0 \leq H^n \leq 1$, where H^n is the final normalised hand vector.

The orientation of the hand, θ_X, θ_Y, and θ_Z ranges from 0° to 360° . This must also be mapped to the range 0 to 1,

$$\theta_X^n = \frac{\theta_X}{360}$$

$$\theta_Y^n = \frac{\theta_Y}{360}$$

$$\theta_Z^n = \frac{\theta_Z}{360}$$

resulting in $0 \leq \theta_X^n \leq 1$, $0 \leq \theta_Y^n \leq 1$, and $0 \leq \theta_Z^n \leq 1$.

The range of the network's outputs is also 0 to 1. So pitch and yaw, which are in the range -360°to 360°, are first mapped to the range 0 to 360,

$$\alpha^m = \begin{cases} \alpha + 360 & if \quad \alpha < 0 \\ \alpha & otherwise \end{cases}$$

$$\beta^m = \begin{cases} \beta + 360 & if \quad \beta < 0 \\ \beta & otherwise \end{cases}$$

then scaled to be in the range 0 to 1,

$$\alpha^n = \frac{\alpha^m}{360}$$

$$\beta^n = \frac{\beta^m}{360}$$

where $0 \leq \alpha^n \leq 1$ and $0 \leq \beta^n \leq 1$.

Data Post-Processing

The pitch α and yaw β of the upper arm output by the neural network is in the range 0 to 1. This must be mapped back to degrees in the range 0° to 360°. This is simply achieved by multiplying the output by 360,

$$\alpha = o_1 * 360$$

$$\beta = o_2 * 360$$

where o_1 and o_2 are the first and second outputs of the network.

Determining Elbow Position

The position of the elbow E can be determined by taking a $-Y$ vector, first rotating it by α about the $-Z$ axis, and then rotating it by β about the Y axis as shown in figure 2.17. This gives us a vector OA indicating the direction of the elbow. Since we know the length of the upper arm, a triangle can be formed, as shown in figure 2.18. Calculate the angle between OA and OS,

$$\theta = \arccos(OA \bullet \widehat{OS}) = \arccos(OA \bullet -X)$$

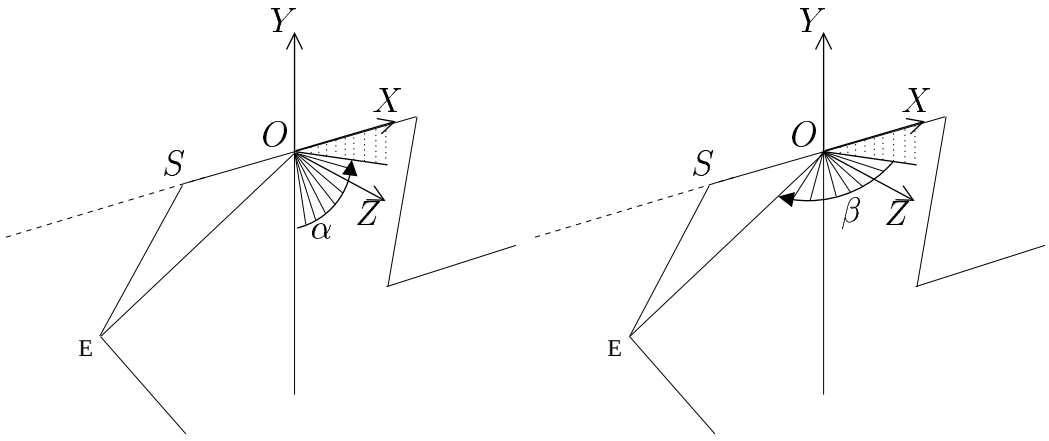
The length $|OE|$ can be calculated as follows,

$$|OE| = \sqrt{l_u^2 - (l_s \sin(\theta))^2} + l_s \cos(\theta)$$

So OE can be determined by simply extending OA to this length,

$$OE = OA * |OE|$$

The elbow position can be set at this stage. However, during normal use of the system the neural network is responsible for the values of upper arm pitch and yaw, so only a prediction of the elbow position E^{pred} can be determined. Because the avatar's arm is a linked structure and the length of the parts are fixed, using E^{pred} to position the upper arm could prevent the hand appearing in the correct place. To ensure it appears in the same position as the hand sensor, an IK correction algorithm is used rotate the upper arm until it has the correct orientation.



(a) Pitch (b) Yaw

Figure 2.17: Determining plane.

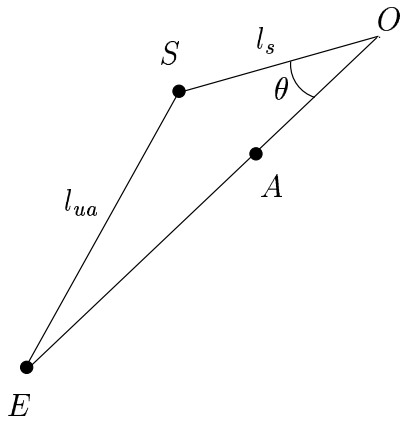


Figure 2.18: Elbow plane triangle.

2.5.2 Inverse Kinematic Correction

The most important aspect in animating the users arm is that their hand position and orientation is accurate. In a virtual environment, where manipulation of virtual objects is a necessity, the position of the users hand must be accurate to allow contact to occur with the object at the correct point. In my system this is achieved by applying a simple IK algorithm to correct any inaccuracy in the networks prediction of arm orientation.

The method works by looking at the position of the hand and calculating the angle that the upper arm should make with the vector from the shoulder to the hand. The upper arm is positioned using the neural network, and rotated until it makes the correct angle with the shoulder-to-hand vector. Finally, the lower arm is positioned.

First, the desired angle between the upper arm and the shoulder-to-hand vector is calculated. Figure 2.19 shows the triangle made by the shoulder S, elbow E and hand H. At this point only the correct position of S and H are known in space, the elbow position determined so far is only a prediction E^{pred}. The angle θ between SH and SE is calculated,

$$SH = H - S$$

$$\theta = \arccos\left(\frac{|SH|^2 + l_{ua}^2 - l_{la}^2}{2|SH|l_{ua}}\right)$$

Next, the predicted upper arm vector is calculated,

$$SE^{pred} = E^{pred} - S$$

and the predicted angle θ^{pred} and normal $norm$ between SE^{pred} and SH is calculated,

$$\theta^{pred} = \arccos\left(\widehat{SE^{pred}} \bullet \widehat{SH}\right)$$

$$norm = \widehat{SE^{pred}} \times \widehat{SH}$$

The difference between θ and θ^{pred} is calculated,

$$\theta^{diff} = \theta^{pred} - \theta$$

And finally, SE^{pred} is rotated about $norm$ by θ^{diff} to produce an orientation of the upper arm that allows the hand to appear in the correct place, as figure 2.20 shows.

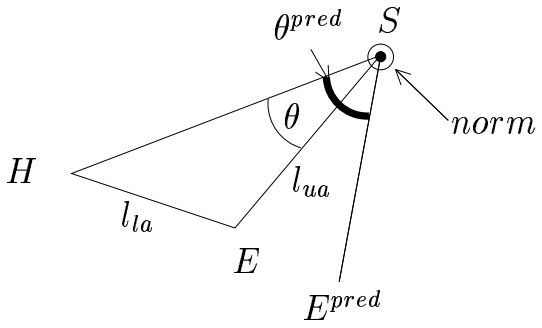


Figure 2.19: Calculation of correct upper/lower arm angle.

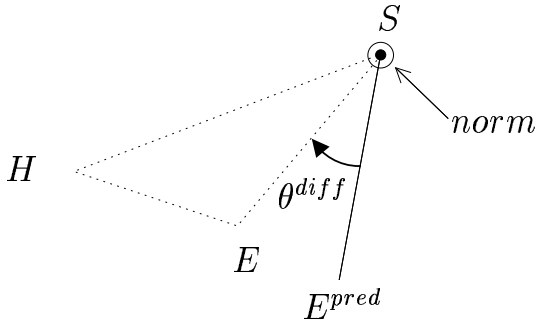


Figure 2.20: Rotation of predicted upper arm.

The rotation of the predicted upper arm SE^{pred} about $norm$ has the effect of performing the IK correction in a plane predicted by the neural network. Hence an element of the network's prediction remains even if it is inaccurate.

2.5.3 Other Considerations

Upper Arm Roll

The network prediction of upper arm orientation does not include its roll. The added computational cost of its calculation is an unnecessary overhead when considering the following points;

- The movement of the human upper arm does not roll much in real life.
- People tend not to look at their upper arms directly, but see them only in their peripheral vision. This means that fixed upper arm roll is not noticed.
- The user can not easily look at their upper arm in a virtual environment due to restrictions of their field of view [18].

Lower Arm Roll

The lower arm is much more visible so its roll must be animated. It is easily determined by observing the rotation of the hand. This is described in detail in appendix A, section A.4.

Left Arm

The neural networks were created using data from the right arm only. To allow the orientation of the left arm to be predicted by the same network the left hand's position and rotation (input) data is mirrored in the YZ plane to represent the equivalent right hand data. This is given to the network whose output is used to determine an elbow position as normal. This is then mirrored back to give the equivalent left arm elbow position.

Chapter 3

Neural Networks Introduction

This chapter aims to describe the fundamental concepts of neural network design, training and testing. It will present key concepts that are needed for a full understanding of the remaining chapters.

For my research I have used a standard type of neural network, the Multi-Layered Perceptron (MLP). The network is trained via Backpropagation, the most common and successful training algorithm. The algorithms described here are those described in the book "Machine Learning" by Tom Mitchell [22]. My neural network code is based on an implementation of these algorithms by Jeff Shufelt [28].

3.1 Structure

The structure of a MLP is built up of many *nodes* - single processing units based on the *Perceptron*. For this reason it is necessary to first describe Perceptrons, before proceeding to a description of MLPs.

3.1.1 Perceptron

The Perceptron is a single processing unit that is based on the biological neurons that make up the human brain. The structure is shown in figure 3.1. It

accepts a number of inputs and outputs an activation if their weighted sum exceeds some threshold. The inputs have associated weights that indicate the strength of the connection. These weights are altered during the training process so that the network can build an internal representation of the input/output mappings present in the data, i.e. learn it.

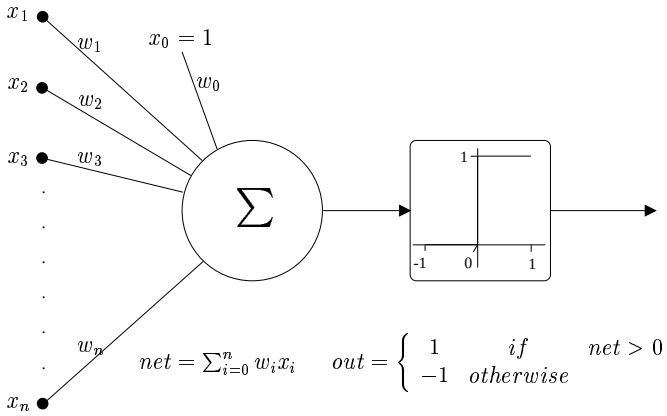


Figure 3.1: The Perceptron

To determine whether the Perceptron fires, each input is multiplied by its associated weight, and summed to produce the net input into the Perceptron,

$$net = \sum_{i=0}^{n} w_i x_i$$

They are presented to the Perceptron's *stepping* function (shown in figure 3.2). It simply tests the net input to see if it exceed some pre-defined threshold, usually 1,

$$out = \begin{cases} 1 & if \quad net > 0 \\ -1 & otherwise \end{cases}$$

The problem with Perceptrons is their simplicity. They only produce binary output, and hence are able to classify only *linearly separable* data, i.e. data for which the input space can be separated by a single straight line. For example, the AND logic task is linearly separable, as shown by figure 3.3. An example of a simple task that is not linearly separable is the XOR logic task.

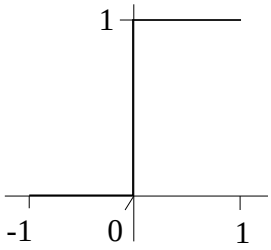


Figure 3.2: Stepping function

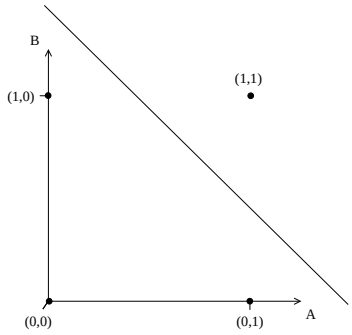


Figure 3.3: AND Logic Task

XOR Problem

The XOR logic task cannot be learnt by a Perceptron as it is not linearly separable. Figure 3.4 shows the input space, and an example division that will classify the inputs properly. Note that no possible divisions are linear. The XOR problem is fundamental as nearly all complex tasks, which would be useful to solve with a neural network, involve the XOR task in some form or another.

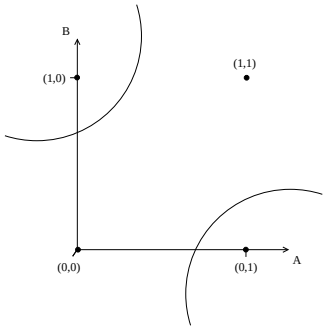


Figure 3.4: XOR Logic Task

This problem was solved with the invention of Multi-Layered Perceptron's, described in section 3.1.2.

3.1.2 Multi-Layered Perceptron (MLP)

With only slight alteration Perceptrons become the nodes that make up a Multi-Layered Perceptron (MLP). An MLP must produce a continuous output, so the stepping function must be changed to one that can do so. Figure 3.5 shows the widely used Sigmoid function[1]. The output of a node is now calculated by presenting its weighted input to the Sigmoid function,

$$out = f(net)$$

$$f(y) = \frac{1}{1 + \exp^{-y}}$$

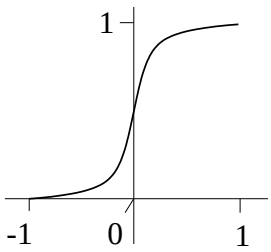


Figure 3.5: Sigmoid function.

Figure 3.6 shows the structure of a MLP. They are made up of many nodes organised into layers - an input layer that carries the input pattern, 1 or more hidden layers, and an output layer that carries the networks prediction of the desired output pattern. The layers are connected so that the outputs of one layer feed into the inputs of the next. MLPs are usually fully connected, meaning that each node in a layer is connected once to every node in the surrounding layers, as shown by figure 3.6.

[1]Note that the diagram displays only an approximation of the function.

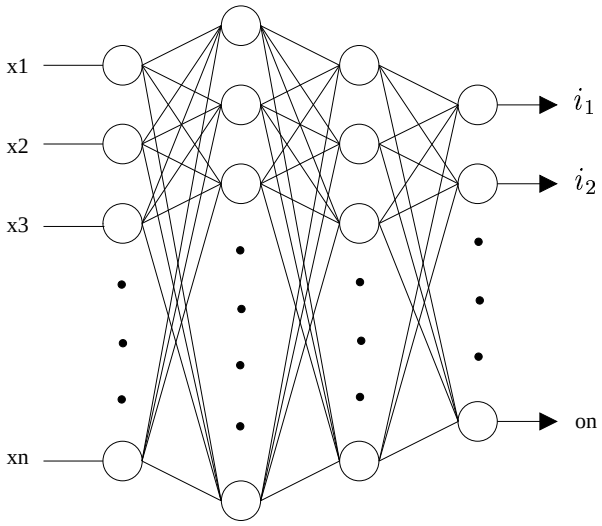


Figure 3.6: MLP

3.2 Design

There is no set method to design the architecture of a neural network. For simple problems it can be possible to theorise which architectures could work well, but for complex problems the designer must rely on their past experience of similar problems, and trial and error. Rules of thumb are often proposed but have not been widely adopted due to limitations in their flexibility or indeed validity.

3.3 Training

Training of the MLP will be performed by a popular and successful gradient descent training algorithm called Backpropagation.

The network is trained by presenting it with an input pattern $\overrightarrow{x}$, calculating the error of the network by comparing the output of the network $\overrightarrow{o}$ and the desired output $\overrightarrow{t}$, and adjusting the weights of the network so that on a future presentation of the pattern the error will be reduced. The presentation of a set of input-output patterns and the corresponding weight adjustment takes

place over an *epoch* of training. The complete training is carried out over many such epochs. A learning rate η determines how quickly the network learns by determining the degree to which the weights are changed during training. A momentum term α acts to accelerate learning. The training process can be visualised by imagining the weight space of a network that has only 2 weights to adjust. This space is shown in figure 3.7. Backpropagation adjusts the weight in the direction of the steepest gradient[2] along a path towards the point with lowest error. The learning rate η determines the size of the steps along this path, and the momentum term α increases the size of the steps if the gradient remains steep and decreases it if the gradient starts to level out. The analogy is that of a ball rolling down a hill. For example, if the weights are initialised at position A where the gradient is steep and the error is high, the weights are adjusted by a relatively large amount. As the position approaches B, they are adjusted in finer amounts until they eventually settle on their final values at position B, the point of lowest error.

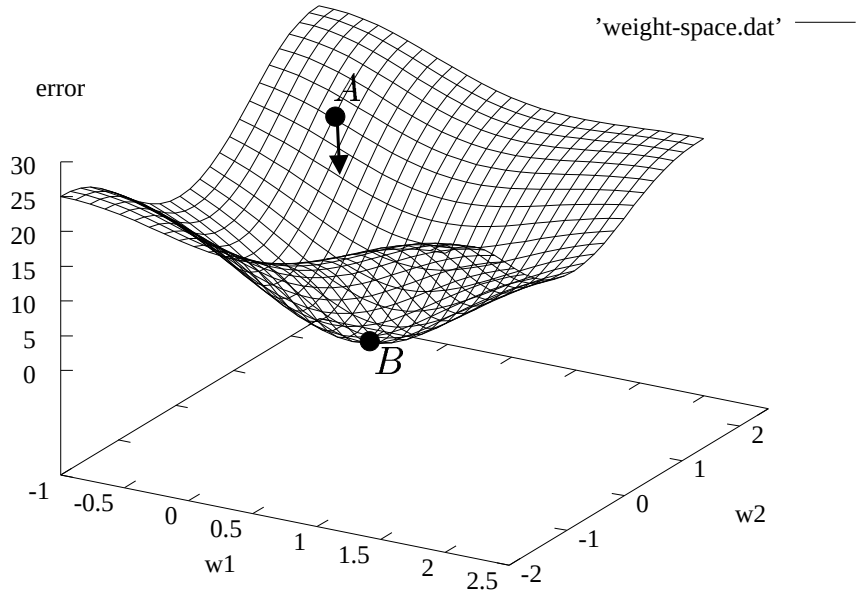


Figure 3.7: Weight space.

A full mathematical description of the Backpropagation training algorithm is given in appendix B.

[2]Backpropagation is one of a class of learning algorithms called Gradient Descent algorithms.

3.3.1 Validation

The errors calculated in training (used to adjust the weights) are indications of how well the network performs on the data it has "seen". It is not a good estimate of the error when tested on a new set of data. To predict how well the network *generalises*, i.e. how well it performs on unseen data, the network must be tested on a set of data that did not contribute in training. This stage is called *validation*, the set of data used to determine the error is referred to as the *validation set,* and so the error of the network is the *validation error.*

Early Stopping

Because the validation error is an estimate of how well the network generalises, it is used to determine when training should stop - i.e. when the validation error starts to rise, as shown in figure 3.8. Notice that in this case training should stop when the error on the training data is still decreasing. If the error on the training data is used as a stopping criteria, the network will over-generalise, that is, it will have learnt the training data so well as to be detrimental to its performance on unseen data.

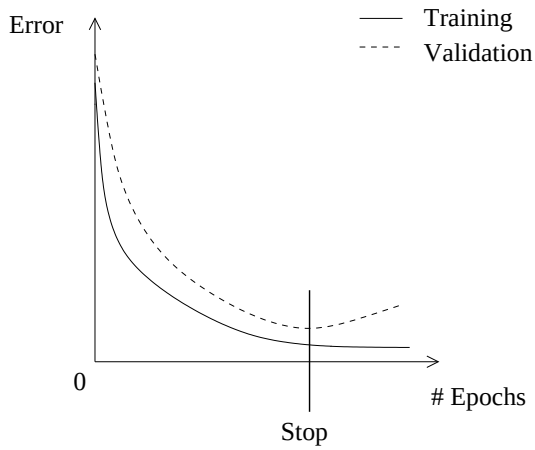


Figure 3.8: Early Stopping

3.4 Testing

Testing, like validation, should be performed on another different set of data, referred to as the *testing set*. Like validation, testing is used to determine how well the network performs when trained and in use. The validation error is a first indication of this, but as it was already presented to the network during training to determine when it should stop, a different set is needed to determine the error after training.

It is often the case that validation determines when the network has sufficiently learnt the data available for training, and testing indicates the networks performance on new data. It is at this stage when the network can be tested on unusual patterns and those that are not expected to frequently occur. These patterns should not be used in the training or validation as they represent cases that the network is not likely to be presented with in use, but for which it is still good to have an indication of how the network will react.

3.5 Error Measurements

The real error E of a node is simply the difference between the desired (target) output t and the actual output o of the node,

$$E = t - o$$

This gives the error on a single element of the target pattern. The error of the whole network on an entire target pattern is calculated using the Sum-Squared Error (SSE),

$$E_n = t_n - o_n$$

$$SSE = \frac{1}{2} \sum_{n \in N} E_n^2$$

where t_n is the target output and o_n is the actual output for a node n in the set of output nodes N. The error of the network for the set of patterns presented in 1 epoch of training is either the Mean-Squared Error (MSE),

$$MSE = \frac{\sum_{p \in P} SSE}{size(P)}$$

where the set of patterns P consists of $size(P)$ patterns p, or the Root Mean Squared (RMS) error,

$$RMS = \sqrt{MSE}$$

Chapter 4

Experimental Procedure

This chapter begins by introducing 3 experiments that explore the performance of neural networks at predicting elbow position. It explains the tasks required to gather data from a number of participants, and the steps needed to prepare the data for presentation to the neural networks.

4.1 Experiments

4.1.1 The 3 Experiments

There are 3 experiments:

Experiment 1 - Initial Exploration: This is an initial exploration of both the networks and the data.

- A number of networks of varying complexity will be trained and tested on the data from each participant in turn. This will determine the extent to which the networks can approximate the data.
- Testing the networks will serve as an indication of their potential performance, and help identify any anomalies in the data that may effect their ability to learn the data.

- The most promising network structures will be explored further in experiment 2.

Experiment 2 - Exploration of the Best Network Structures: This is a full exploration of the most promising network structures identified in experiment 1.

- It will compare the performance of *user-specific* networks trained on data from a single participant with that of *user-independent* networks trained on data from all participants.
- Similarly, the performance of *task-specific* networks trained on data from a structured task will be compared to that of *task-independent* networks trained on data from random arm movements.
- The data used was gathered from participants during a single session so that error caused by re-attaching the sensors is not present. This allows comparison of the networks without having to consider the extent to which error has been introduced by external influences.
- The best network structure identified will be tested in experiment 3.

Experiment 3 - Testing Generalisation: This tests the best network structure identified in experiment 2 on its generalisation to new users and new tasks.

- Data for this experiment was gathered in a different session to that which gathered the training data.
- The ability of the network to generalise to new users will be determined by testing it on data from a new set of participants.
- The performance of the network on a variation of the structured task will determine whether it has sufficiently learnt the nature of the movements required to generalise to slight variations.

- It will determine the error introduced by re-attaching the sensors by comparing the network's performance on data gathered in the same session as the training data, with its performance on data gathered in different session.
- A set of scenarios are introduced to give the expected performance of the network for a range of possible application requirements.

4.1.2 Experimental Aim

The aim of the experiments is to determine the extent to which neural networks can learn to predict the orientation of a user's upper arm given the position and orientation of their hand.

4.1.3 Positional Error

It is useful to define the positional (real) error of the network as the mean distance between the target elbow positions and the predicted elbow positions. The distance between the predicted elbow position E_p^{pred} to the target elbow position E_p^{target} of a single pattern p is given by,

$$Distance_p = |E_p^{target} - E_p^{pred}|$$

Over all patterns in a testing set the positional error is given by,

$$Error = \frac{\sum_{p \in P} Distance_p}{size(P)}$$

where the set of patterns P consists of $size(P)$ patterns p.

The positional errors highlighted in the results chapters are determined when the IK correction technique is operating, as it would be in normal use of the system. For comparison the full results in appendix C also gives the performance when the IK correction is not in operation.

Some of the graphs later refer to the positional error as the "real" error.

4.1.4 Network Notation

A neural network will be notated in the format {number of input nodes}x{number of nodes in hidden layer 1}x{number of nodes in hidden layer 2}x{number of output nodes}. So 6x8x8x2 represents a network with 6 input nodes, 2 hidden layer with 8 nodes each, and 2 output nodes. A 0 indicates no layer, so 6x8x0x2 represents a network with 6 inputs, a single hidden layer containing 8 nodes, and 2 outputs.

Networks with 2 hidden layers will be referred to as 2-layer networks, similarly those with only 1 hidden layer will be referred to as 1-layer networks.

4.2 Data Gathering

To facilitate exploration of whether a task-specific or task-independent network would perform best we require a structured task and a task that produces random arm movements. The latter, henceforth referred to as the "random" task, is simply a convenient replacement for saying to the user "move your arms around randomly". The structured task, referred to as the "rotation" task due to the frequent hand rotation, is designed to emulate a real-world application, and has a predefined structure that requires specific arm movements.

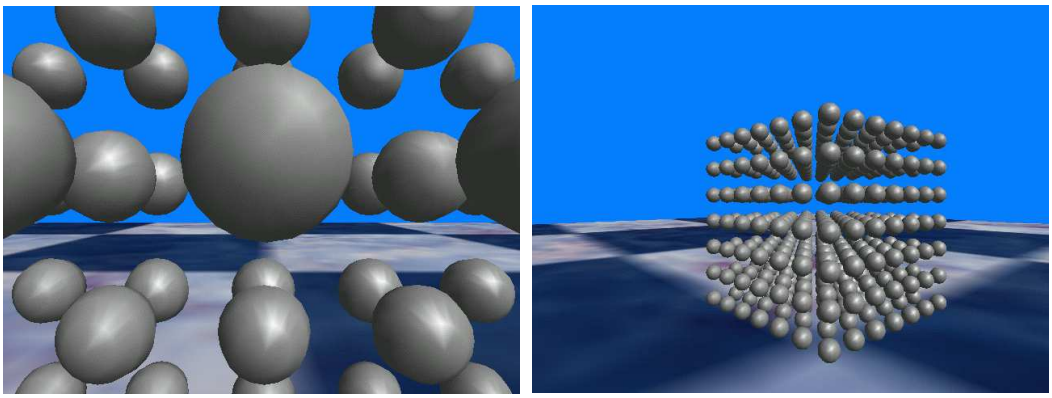
Additionally, to test whether the network can generalise to variations in a structured task, a modification of the rotation task, called the "high rotation" task, is defined. This task requires the rotation task to be performed in different (higher) region of space.

4.2.1 Random Task

The problem with asking a participant to move their arms around randomly is that different participants will tend to cover different areas of space and move their arms in a manner specific to them, often repeating certain motions. In other words, their movements will not be as random as one wishes. The random task is designed to allow the participants to visualise the areas of space

they have already covered, so that they are more likely to move their arm to uncovered areas, less likely to cover the same area more than once, and so that different participants are more likely to cover the same region of space.

The participant was placed in a grid of spheres, shown in figure 4.1, that disappear on contact with the their hand. The participant was asked to move their hand over as many spheres as possible. Data was gathered over a period of 1 minute, which allowed the user to cover most of the space within reach.



(a) Vew from head (b) The grid of spheres from a distance

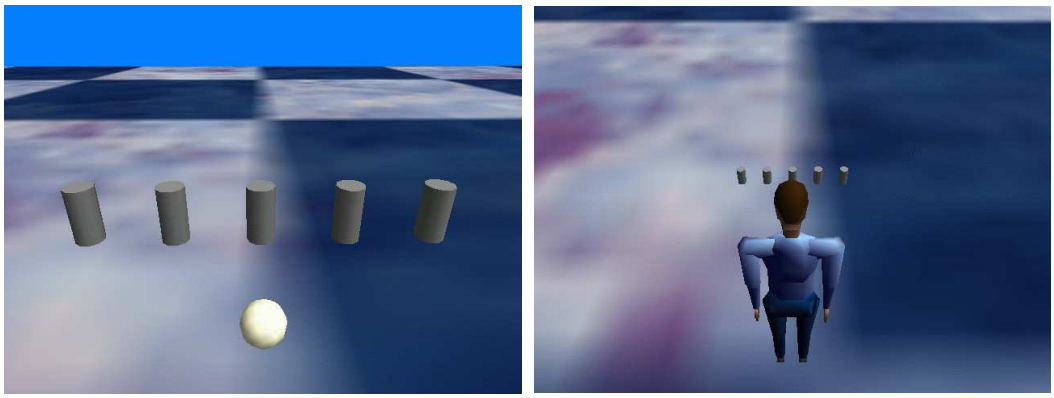
Figure 4.1: Random task.

4.2.2 Rotation Task

The rotation task is designed to produce the kind of structured movement that would be generated when performing a real-world task, such as the manipulation of radioactive containers or the assembly of a mechanical part.

The participant was placed in front of 5 cylinders hovering next to one another at chest height, with a small sphere hovering between the cylinders and their chest, see figure 4.2. The participant was asked to pick up the right most cylinder, rotate it 180 degrees and touch the top of it onto the sphere. On contact with the sphere the cylinder disappears. The participant was asked to repeat this for each cylinder in turn, moving right to left, until all 5 cylinders were removed. When they had done so, the participant was asked to place their

hand on their lap, at which point the 5 cylinders reappear. The participants were asked to repeat the above process, except this time moving from left to right. This was repeated another 2 times, each time moving the opposite way across the spheres. The movements are similar to the pouring action shown in figure 1.4, except they also involve displacement of hand position.



(a) View from head (b) Distant view

Figure 4.2: Rotation task.

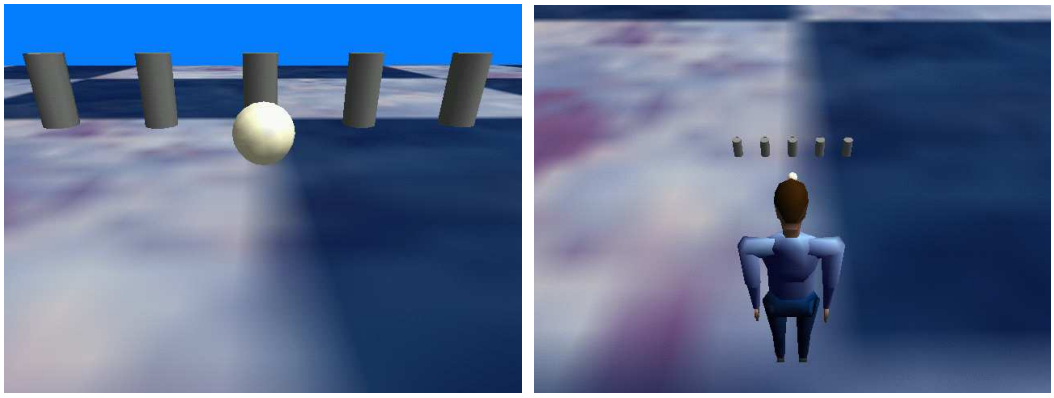
High Rotation Task

This task is nearly identical to the normal rotation task except the position of the objects have been raised by 1 meter. A comparison of the screen-shots from the normal rotation task in figure 4.2 and the high rotation task in figure 4.3 shows the difference between the height of the objects.

This task will produce arm movements similar to those generated when performing the normal rotation task, but in a different area of space. A similar relationship will exist between the hand data and the upper arm orientation data, but the values will be offset by a certain degree.

4.2.3 Procedure

The participant sits upright with their upper arm at their side and their lower arm pointing directly forwards, hand outstretched, as shown in figure 2.2.



(a) View from head (b) Distant view

Figure 4.3: High Rotation task.

The participant's shoulder width, upper arm length, and lower arm length are measured and stored in a file. The sensors are attached to their body (as shown in figure 2.3) and they are asked to sit upright as the sensors are initialised. They perform a trial run of each task to familiarise themselves with what is required.

Data was gathered in 2 sessions:

Session 1: Data was gathered from a total of 6 participants. Each was asked to perform the random task 3 times and the rotation task 3 times. This gives a total of 6 data files per participant. The data is used in all 3 experiments.

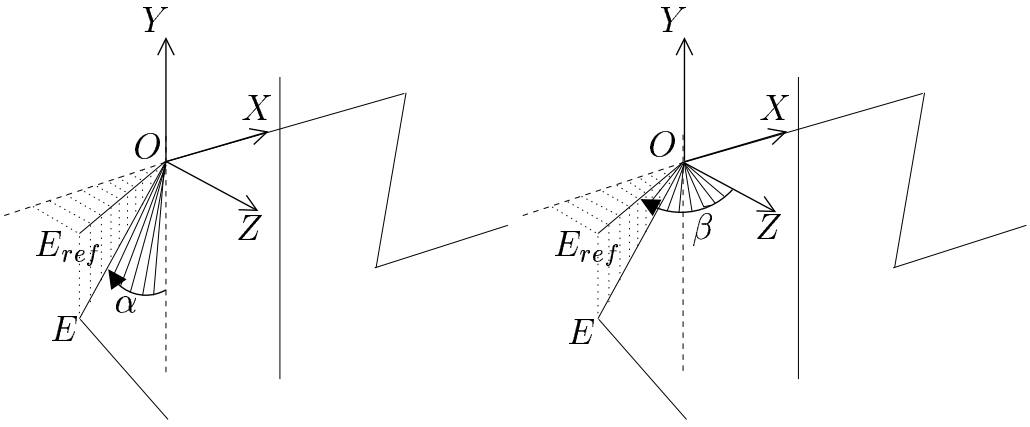
Session 2: The sensors were re-attached to the body and the offsets were re-measured. Data was gathered from a further 8 participants and again from the original 6. The participants were required to perform the random and the rotation tasks once each. Additionally, they were asked to perform the high rotation task once. This gives a total of 3 data files per new participant, and an additional 3 files per original participant. The data is only used in experiment 3.

Immersion in VR can be strenuous, as mentioned in section 2.2. This is particularly true of the high rotation task as it requires arm movement above

the head. The participants were free to stop for a rest if necessary. If any errors were made performing a task the participant was asked to simply continue and finish the run. They were not required to repeat the task. Accurate performance of the task was desired simply to allow the data sets gathered from different participants to exhibit a similar structure. Note that a slightly inaccurate performance of the task does not hinder testing and comparison of the networks.

Specification of Data Gathered

The data gathered differed from that required in the final implementation. Rather than having the origin at the back and gathering the upper arm yaw relative to the X axis, the origin was at the shoulder and the yaw was relative to the Z axis. Figure 4.4 shows the upper arm orientations gathered. A mathematical description of how the data was calculated will not be provided as it is essentially identical to the description in section 2.5.1 except that SE replaces E, and Z replaces X. Section 5.3.1 justifies the change in the specification.



(a) Pitch (b) Yaw

Figure 4.4: Data gathered.

Categorisation of Data Sets

To allow easy reference of the different data sets gathered, a categorisation scheme will be adopted. Data from the 6 initial subjects are categorised using the letters A-F, and the further 8 subjects using G-N. Data from all subjects is referred to with "all". These will be followed by "rand" to indicate random task data, "rot" to indicate rotation task data, or "highrot" to indicate high-rotation task data. "Both" indicates data from both the random and the rotation tasks. Finally, a number is used to indicate which of the performances the data was gathered from. For example, D-rand-2 indicates data from user D performing the random task for the second time, and A-both-1,2 indicates a set consisting of data from the first and second performances by user A of both the random and the rotation tasks.

Sets containing data from more than 1 participant are cut down in size to contain roughly the same number of patterns as the equivalent set from a single user. Likewise, sets containing data from more than 1 task are cut down in size to have the roughly the same number of patterns as a single task. So all-rand-1,2 contains the same number of patterns as C-rand-1,2.

4.3 Data Preparation

There are various steps needed to prepare the data before it can be presented to the networks for training and testing. The data must be cleaned, split into sets, and finally converted from its raw form into a form that is acceptable for the network. The values of the input patterns must range between 0 and 1, as must the targets. The steps required are outlined below.

4.3.1 Translation

The first experiment determined that it was necessary to apply a translation to the data gathered from the participants. Chapter 5 describes and justifies the translation required.

4.3.2 Duplicate Patterns

The data sets often contain duplicate patterns. Their removal is important to prevent the network over-learning them, which will have the effect of reducing the generalisation ability of the network.

The main cause of duplicate patterns is a sensor moving out of range, in which case the tracking daemon that polls the sensor returns identical values until it is moved back into range. It theoretically could also happen if the user remains so still that no change in sensor position or orientation is noticed by the tracker. This however, is unlikely.

4.3.3 Anomalous Patterns

Anomalous patterns, that do not represent sensible positions or orientations, must not be present in the data sets as their presentation to a network during training will partially "undo" the learning already achieved. It will adjust its weights to better classify the bad pattern, moving them away from those required to classify the other good patterns.

There are various reasons why the data sets may contain anomalous readings. A sensor moving out of range returns an anomalous reading rather than its position and orientation before it was out of range. Magnetic field interference can be created by the unexpected introduction of metal or a magnet, or by monitor emissions. This unlikely as the calibration of the tracker has accounted for the presence of any metal and monitors in the laboratory and I was careful that nothing potentially interfering passed within range of the tracker. Finally, the data could be corrupted by fluctuations in the bandwidth of the system, a problem I was unlikely to face but one that is especially relevant with the increased popularity of networked collaborative environments.

4.3.4 Normalisation

Before the data can be fed to the network, it must undergo normalisation to convert it into the form required by the network. This process was described in section 2.5.1.

4.3.5 Randomisation

Patterns close to one another are similar as they are generated by movements in the same part of a motion. This can cause catastrophic interference, where past knowledge learnt by the network is lost due to the presentation of new patterns [21]. The interleaving of data is known to prevent this occurring, so the order of the patterns in the training data file is randomised.

4.3.6 Forming Training, Validation, and Testing data sets.

The final step in data preparation is to combine the data gathered from the users into the sets needed for training, validation, and testing. The 6 data sets per user that were first gathered (section 4.2.3) are combined in different proportions depending on the experiment. The sets are described in chapters 5, 6, and 7 when the experiments themselves are detailed.

In all experiments, the training, validation, and testing sets do not share identical patterns. As mentioned in sections 3.3.1 and 3.4 it is important that the network is not validated or tested using data that has already been presented to it during training for risk of underestimating the error of the network in practical use.

Chapter 5

Experiment 1 - Initial Exploration

This experiment is an initial exploration of both the networks and the data.

5.1 Aim

This experiment aims to determine the extent to which the networks can approximate the data. The most promising network structures will be identified for further exploration in experiment 2. It also aims to determine whether there are any anomalies in the data that may effect the performance of the networks.

5.2 Description

In total, 10 different network architectures will be explored - 4 networks with a single hidden layer, and 6 with two hidden layers. The work carried out in the NeuroAnimator project [12, 11] determined that networks with 1-layer networks have difficulty modelling the dynamics of physical models. This is echoed by both Beverage [5] and Beeharee [3] who determined that 1-layer networks could not model the dynamics of tasks very similar to the ones performed in this project. In response to this a greater range of 2-layer networks

# Nodes in hidden layer 1	# Nodes in hidden layer 2
8	—
16	—
24	—
32	—
4	4
8	8
12	12
16	16
20	20
24	24

Table 5.1: Network structures for experiment 1.

are tested. A smaller number of 1-layer networks are also tested in order to validate their predicted failure. The structures are given in table 5.1.

The data was gathered in session 1. Only data from user A is used in this experiment and it is assumed to be representative of the other participants (experiment 2, described in chapter 6 explores training with different users). The data used for training, validation, and testing is tabulated in table 5.2. Each data set is split into two groups, 90% is used in the training process and 10% is testing data. 10% of the data reserved for the training process is used for validation. The data undergoes the preparation described in section 4.3, except for the translation step.

Task	Data set (participant A only)	Training process		Testing
		Training	Validation	
Random	A-rand-1,2,3	90%		10%
		90%	10%	
Rotation	A-rot-1,2,3	90%		10%
		90%	10%	
Both	A-both-1,2,3	90%		10%
		90%	10%	

Table 5.2: Data used in experiment 1.

The networks are trained for a total of 10000 epochs, quite sufficient to allow full observation of learning.

5.3 Initial Results

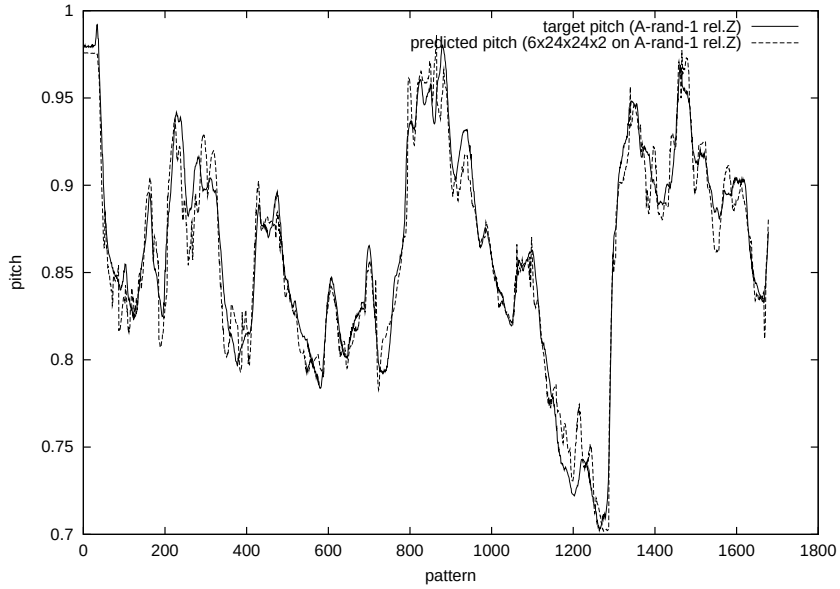
All of the networks have a similar performance, measured as the lowest validation error at any time during training (see section 3.3.1). The errors, accurate to ± 0.01, are tabulated in table 5.3. The network identified as having the most potential was the largest network (structure 6x24x24x2). The performance of the network on (un-randomised) data gathered from participant A's first run of the random task is shown in figure 5.1.

Network	Lowest validation error (MSE)		
	Random	Rotation	Both
6x8x2	0.015	0.010	0.013
6x16x2	0.010	0.012	0.012
6x24x2	0.007	0.013	0.010
6x32x2	0.009	0.012	0.010
6x4x4x2	0.015	0.012	0.014
6x8x8x2	0.009	0.010	0.010
6x12x12x2	0.006	0.010	0.009
6x16x16x2	0.007	0.009	0.009
6x20x20x2	0.007	0.009	0.009
6x24x24x2	0.005	0.002	0.008

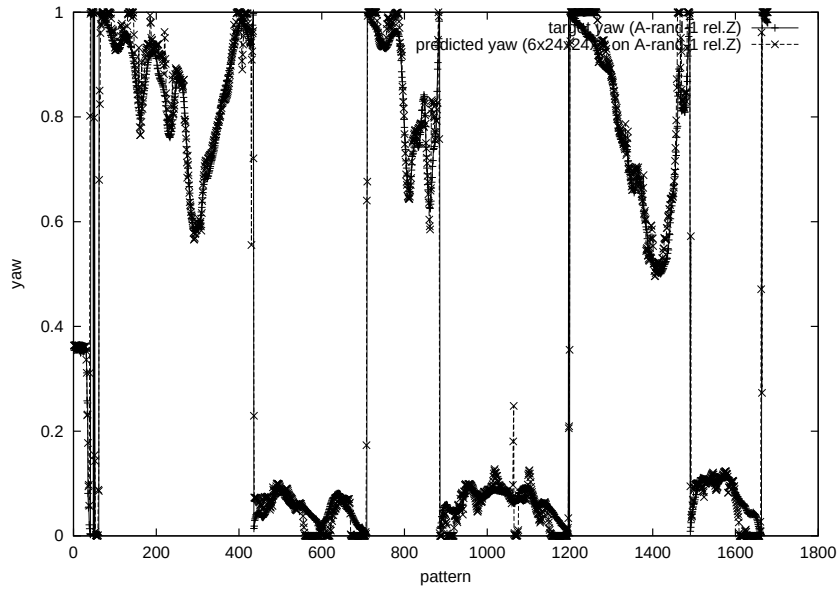
Table 5.3: Network errors for experiment 1, initial results.

A graph of the training, validation and testing errors of the 6x24x24x2 network on data from each task is shown in figure 5.2. You can see that there are large fluctuations in the training and validation error, and while the network has the potential to achieve a very low error it seems equally likely it could settle on a higher one. This effect is only observed with the 2-layer networks, with the exception of the smallest (6x4x4x2). Training of the 1-layer networks produces smooth error curves, an example of which can be seen in appendix A. The erratic nature of the 2-layer networks suggests that they are having difficulty classifying the data, and are possibly attempting to learn something that the 1-layer networks can not. It seems the dynamics of arm movement are too complex to be modelled by 1-layer networks and so their exploration ends here.

The problem was found to originate in the measurement of upper arm orien-



(a) Pitch



(b) Yaw

Figure 5.1: Initial performance of the 6x24x24x2 network on user A performing the random task.

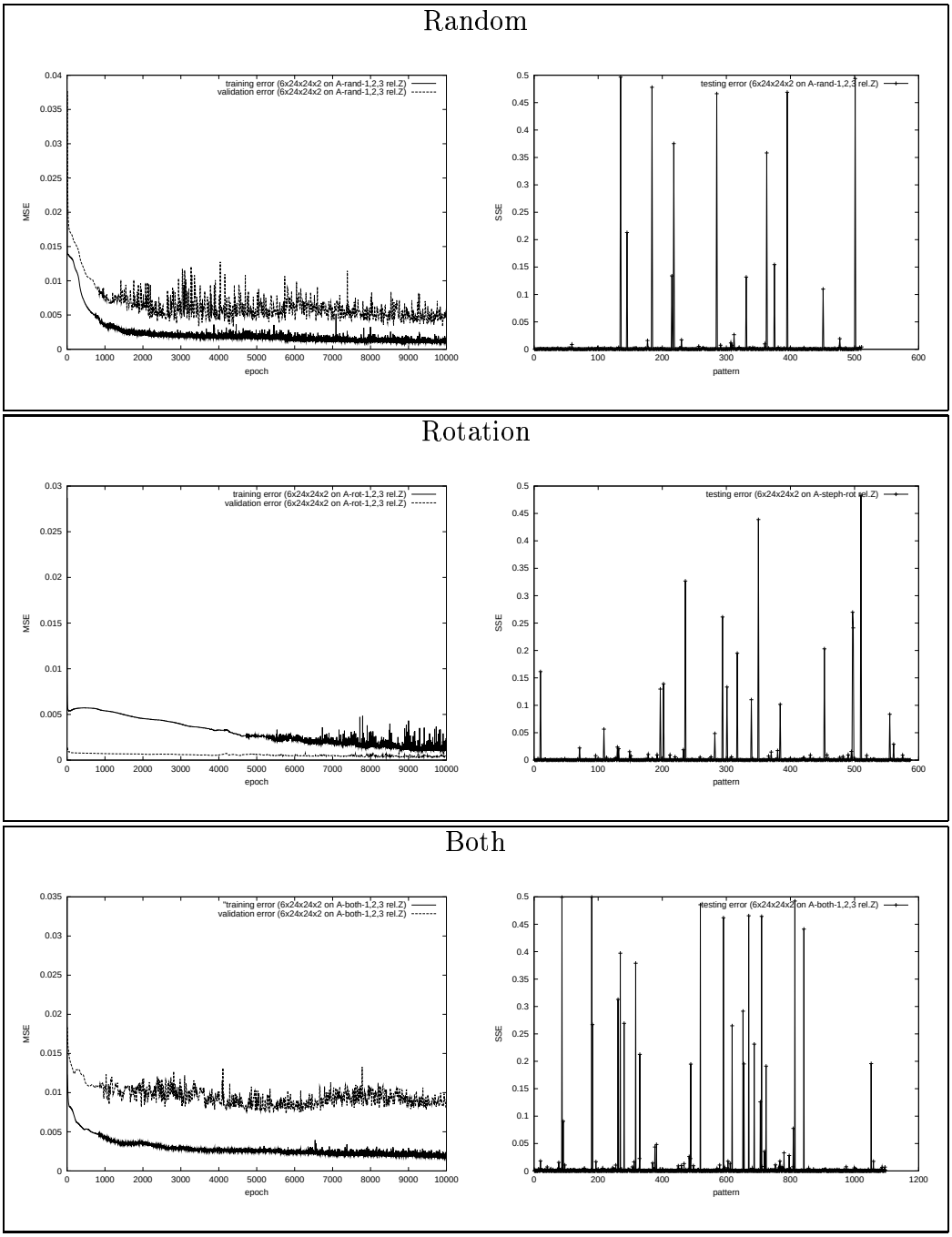
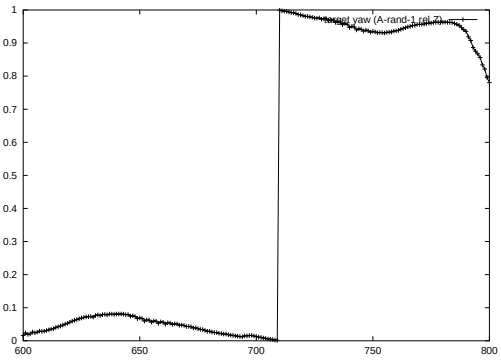
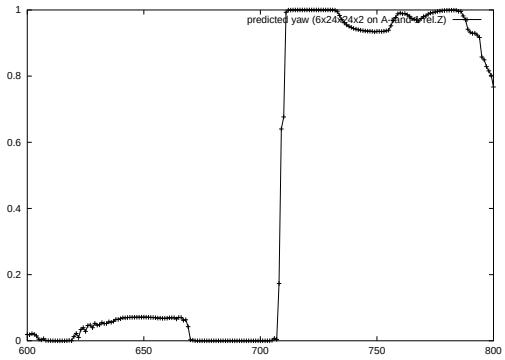


Figure 5.2: The best network of experiment 1 (Initial Results) - training and testing performance of the 6x24x24x2 network

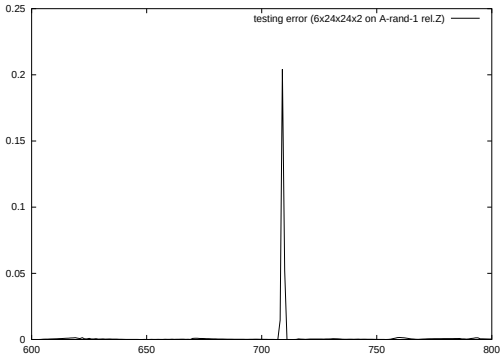
tation. You can see from the testing errors, shown in figure 5.2, that there are large error peaks with magnitudes much greater than any surrounding errors. This indicates that the network is having severe trouble predicting the desired outputs of a few select input patterns. Figure 5.3 displays the target yaw, the network's prediction, and the corresponding error peak on a small section of testing data. There is a surge in error when the target yaw flips from 0 to 1 (representing an angular change from 0° to 360°). The problem arises because the data originally gathered specifies the yaw of the upper arm relative to the Z axis, with the origin at the shoulder. When the upper arm sweeps across this axis the yaw jumps from 0° to 360° (or visa versa). The network performs "interpolation" and guesses values in between the two, resulting in a wildly wrong prediction.



(a) Target yaw



(b) Predicted yaw

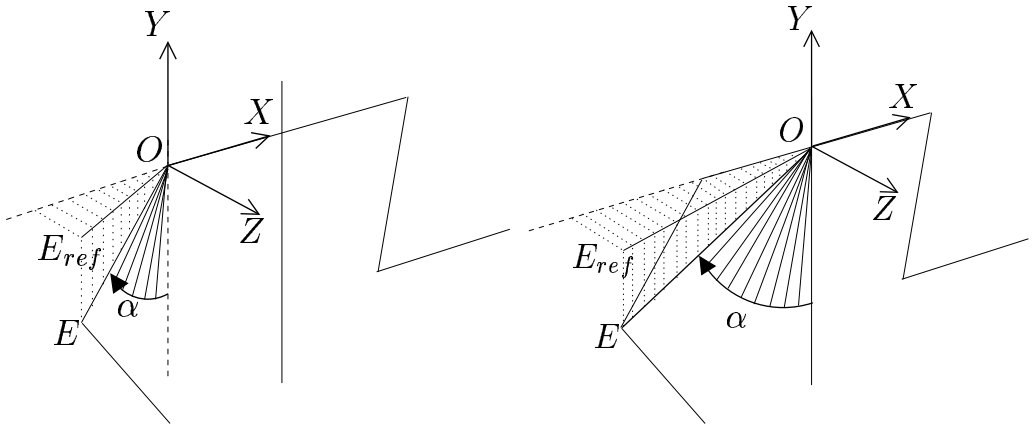


(c) Error

Figure 5.3: Desired yaw and the network's prediction.

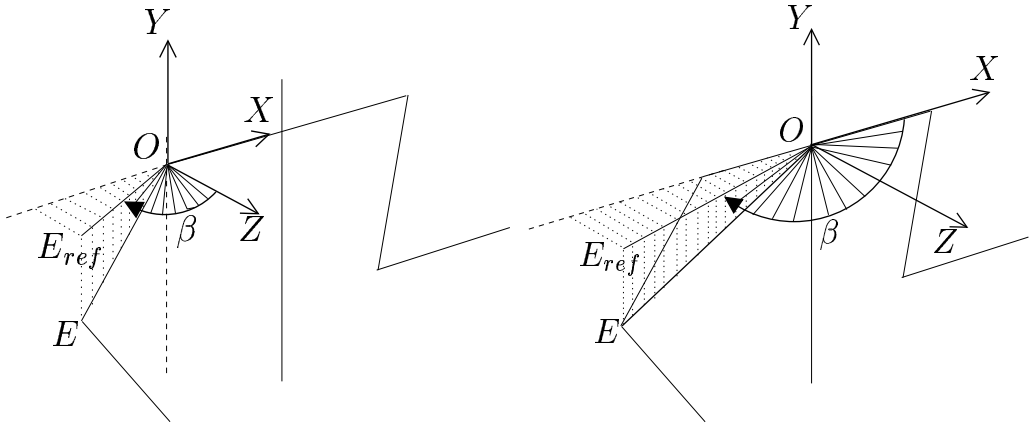
5.3.1 Arm Data Translation

The solution to this problem requires a translation of the arm data so that the yaw is relative to an axis for which an instant change from 0° to 360° is not possible. Figure 5.4 shows the change in specification of upper arm rotation. The origin is now at the center of the torso and instead of measuring the yaw from the Z axis it is measured from the X axis.



(a) OriginalpPitch

(b) New pitch



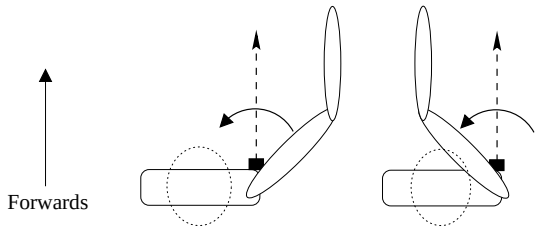
(c) Original yaw

(d) New yaw

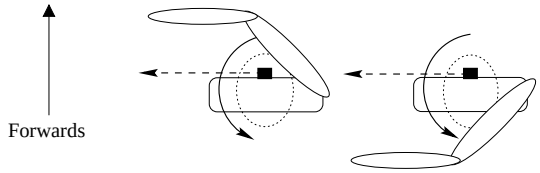
Figure 5.4: Change in upper arm rotation specification.

It is extremely difficult if not impossible for the user to move their right upper arm over the X axis. It would mean performing a very unnatural movement,

and one that is rarely used for everyday tasks. The movement required is shown in figure 5.5 compared to the movement that was required using the original specification.



(a) Original specification of yaw



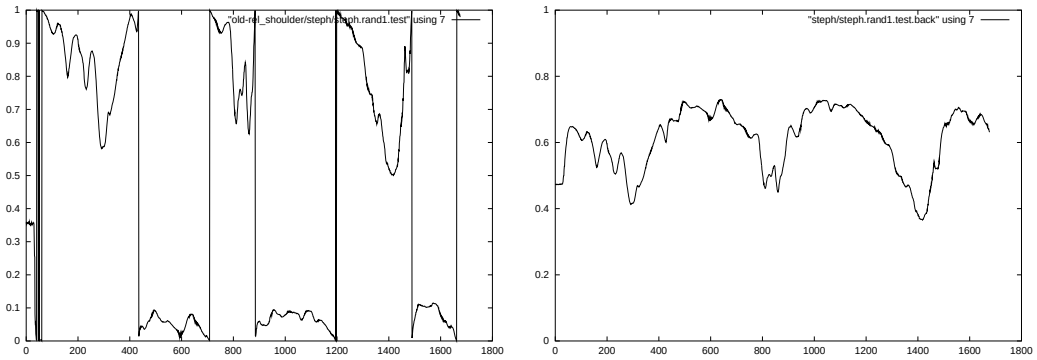
(b) New specification of yaw

Figure 5.5: Movement required to create jumps in yaw between 0 and 1.

Figures 5.6 shows the yaw of the original specification and the yaw of the translated specification. The translation has the effect of bringing the yaw away from the limits of its range and move it towards the middle. This prevents its value suddenly jumping from the largest possible to the smallest possible and should make it easier for the networks to learn the data.

5.4 Re-training

All arm data was transformed so that yaw is relative to the X axis and the origin is in the center of the torso (now the same as the measurement described in section 2.5.1). The larger 2-layer networks were re-trained to determine whether the transformation was beneficial. The results of re-training are given in table 5.4. For a visual comparison, the results of re-training the 6x24x24x2



(a) Original yaw (b) New yaw

Figure 5.6: Example of yaw data from the Random task.

network are shown in figure 5.7[1]. Note that the units have changed. The training and validation performance of the network is now measured in Root-Mean-Squared error (RMS) as opposed to Mean-Squared-Error (MSE). This is because there is a substantial reduction in error (by a factor of 10^{-3}) and RMS error gives more practical units for display.

Network	Lowest validation error (RMS)		
	Random	Rotation	Both
6x8x8x2	0.010	0.006	0.010
6x12x12x2	0.010	0.006	0.009
6x16x16x2	0.009	0.006	0.009
6x20x20x2	0.009	0.006	0.008
6x24x24x2	0.008	0.006	0.008

Table 5.4: Re-trained network errors for experiment 1.

The smooth training curves indicate that there has been a substantial change in the nature of the task. The network is no longer fluctuating between a high and low error but exhibits a smooth decrease in error, to a value lower than previously encountered. The testing graphs also show a large decrease in error.

The performance of the network on (un-randomised) data from participant A performing the random task (figure 5.8) shows the network's outputs are closer

[1]The training results file for A-both-1,2,3 ("Both") became corrupted, hence the end of the training graph is missing. This did not effect the results in any way.

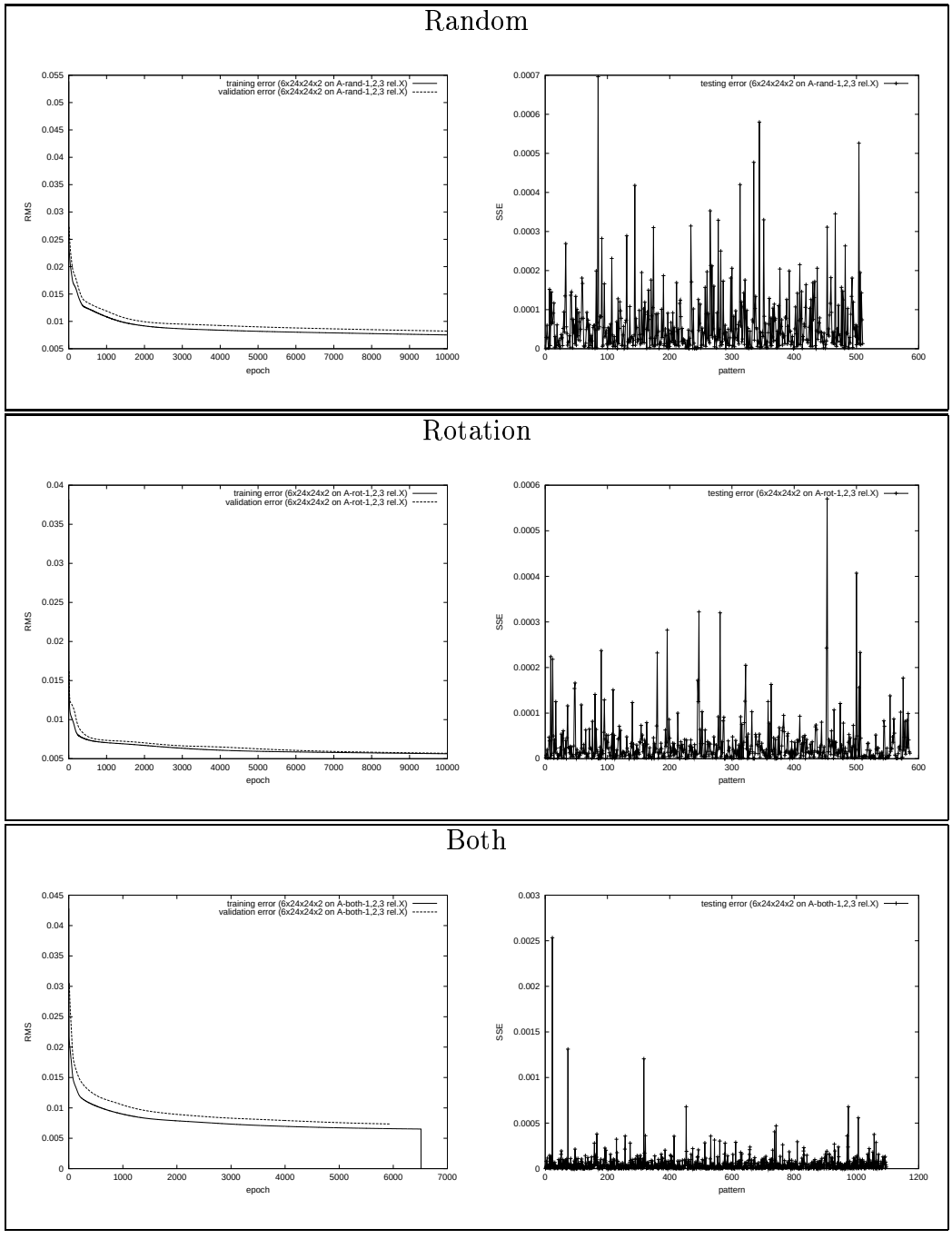
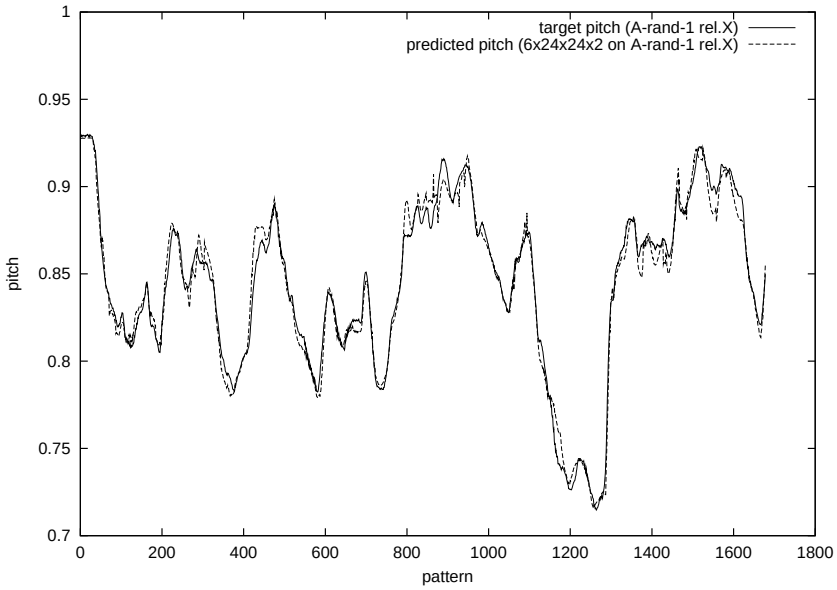


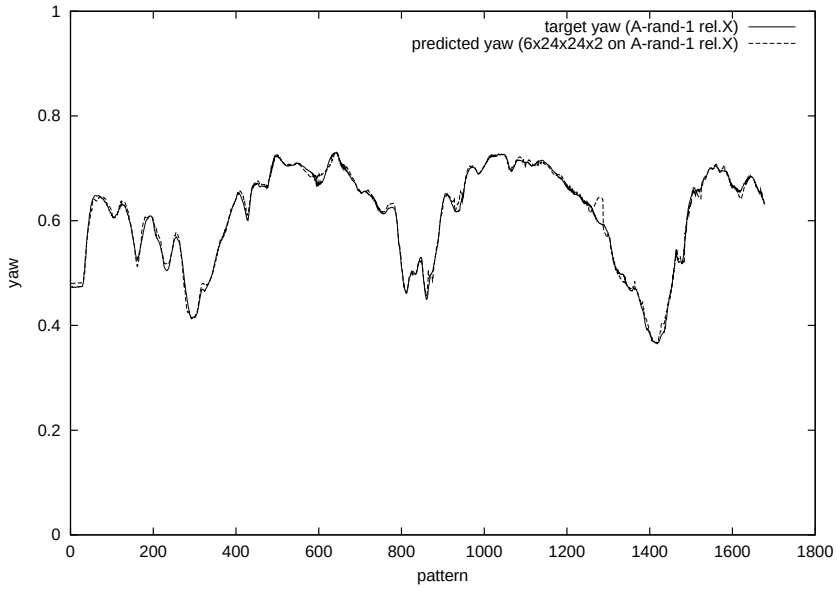
Figure 5.7: Results of re-training 6x24x24x2 network.

to the targets than with the original yaw specification and that the network no longer has difficulty with the prediction. Since the network was trained on the same data this accuracy is expected. But it does show that the network is fully capable of learning the task. The remaining experiments will determine if the network can generalise.

The results indicate a decrease in error as the network size increases. However, the degree of accuracy of the error measurements is such that a clear conclusion as to which 2-layer structure performs best can not be reached. To decrease the time required to complete testing, only the 6x8x8x2, 6x16x16x2, and 6x24x24x2 networks will be explored. These structures cover a suitably large range of complexity.



(a) Pitch



(b) Yaw

Figure 5.8: Performance of re-trained 6x24x24x2 net on A-rand-1 data.

Chapter 6

Experiment 2 - Exploration of Best network Structures

This is a full exploration of the most promising network structures identified in experiment 1.

6.1 Aim

This experiment aims to determine whether user-specific networks perform better than user-independent networks, and similarly, whether task-specific networks perform better than task-independent networks. The experiment will identify the best network structure to pursue for final testing in experiment 3.

6.2 Description

Experiment 1 determined that three 2-layer structures will be explored further. These are tabulated in table 6.1.

As with the first experiment, all data was gathered from session 1. The training, validation, and testing data sets are tabulated in table 6.2. Data from the third run of each task is not used in the training process but instead will be used for testing only. A user-independent set consisting of data from all 6

# Nodes in layer 1	# Nodes in layer 2
8	8
16	16
24	24

Table 6.1: Network structures for experiment 2.

subjects is created by joining the data from all users. This is cut down to the average size of the other data sets to allow accurate comparison of user-specific and user-independent networks.

Task	Data set (from participants A to F)	Training	Validation	Testing
Random	?-rand-1,2	90%	10%	—
	?-rand-3	—	—	100%
Rotation	?-rot-1,2	90%	10%	—
	?-rot-3	—	—	100%
Both	?-both-1,2	90%	10%	—
	?-both-3	—	—	100%

Table 6.2: Data used in experiment 2.

6.3 User-Specific vs User-Independent

It is expected that user-specific networks will perform best when tested on their corresponding user and worst when tested on a different user. A user-independent network will have a performance somewhere between the two. This hypothesis is based on the relative amount of information about the participant provided during training, i.e whether examples of data from the participant was presented to the network during training.

Only data from the rotation task is used in this part of experiment 1. For each of the structures that will be explored (table 6.1) 6 networks are each trained on data from a different participant, and 1 is trained on an equal sized set containing data from all participants. This gives 6 user-specific networks and 1 user-independent network for each structure.

Some illustrative results of the 6x8x8x2 network are shown in table 6.3. These results are echoed with all structures, see appendix C, section C.2.1. The table gives the positional error (defined in section 4.1.3) of each user-specific network on data from their corresponding participant. It also shows the mean positional error of both the user-specific and user-independent network tested on data from the other participants.

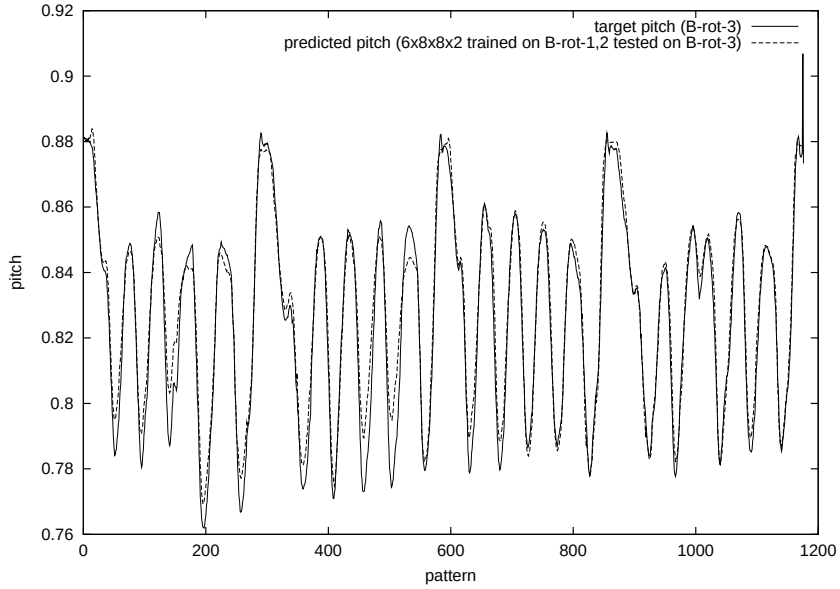
Trained on participant	Positional error on same participant (meters)	Mean positional error on other participant (meters)
A	0.034537	0.054452
B	0.008828	0.070014
C	0.014978	0.046059
D	0.020558	0.055799
E	0.019062	0.114728
F	0.008482	0.051637
All	—	0.035152

Table 6.3: Performance of 6x8x8x2 ANN.

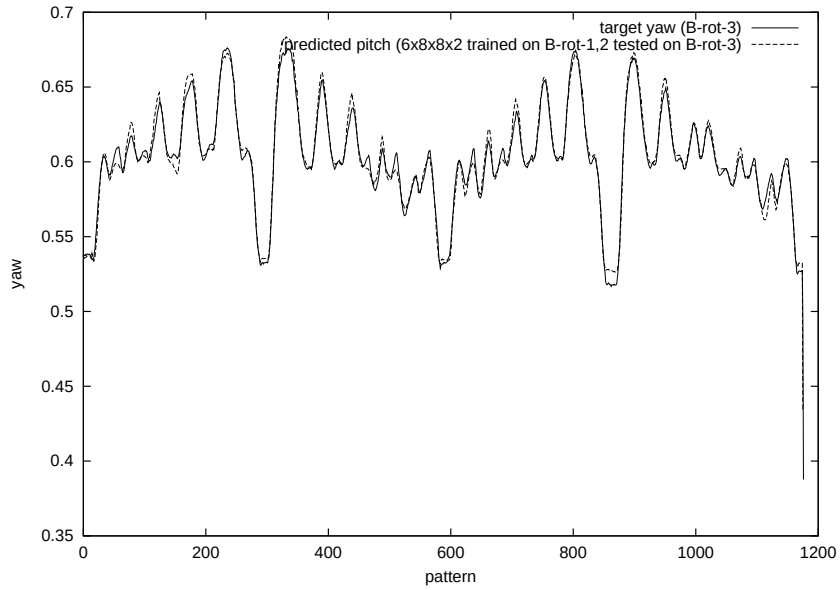
As expected, networks trained on user-specific data perform best when tested on their associated user and worst when tested on a different user. The average error of a user-specific network tested on data from its corresponding participant is just 1.7 centimeters. Tested on a different participant it is more than 3 times greater, with a value of 6.5 centimeters. User-independent networks, that have been trained on data from all users, have a performance in between the two, with an error of 3.5 centimeters.

As an example, figure 6.1 shows the results of testing a user-specific network on data from its corresponding participant (in this case, participant B). Graphs a and b show plots of the target output against the actual output for pitch and yaw. Figure 6.2 shows a plot of the positional (real) error. The user-specific network predicts yaw and pitch well. Its positional error ranges from 0 to 6 centimeters although it remains at the lower end most of the time.

In comparison, figure 6.3 shows the pitch and yaw prediction of a user-independent network tested on data from the same participant. The user-independent network is not as accurate, there is more deviation from the targets that is particularly noticeable in the predictions of pitch as its value oscillates (at the



(a) Pitch



(b) Yaw

Figure 6.1: Pitch and yaw prediction of 6x8x8x2 user-specific network on B-rot-3.

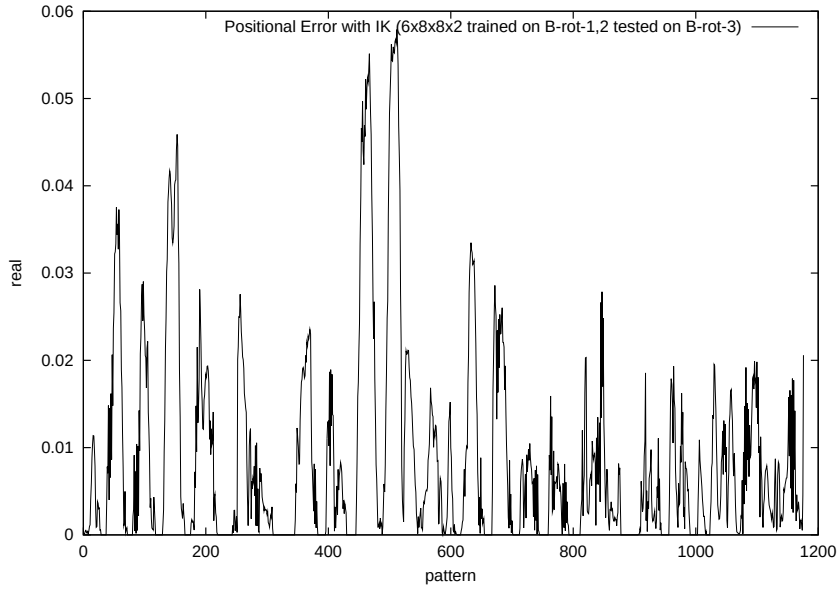


Figure 6.2: Positional error of 6x8x8x2 user-specific network on B-rot-3.

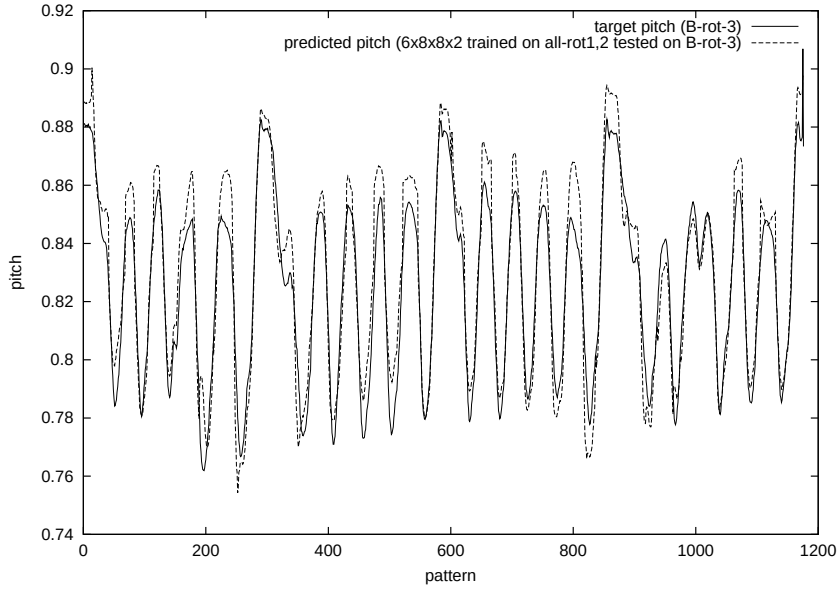
peaks and troughs). Figure 6.4 shows the positional error to be worse by 2 centimeters, with the error in elbow position ranging from 0 to 8 centimeters.

It is safe to conclude that a user-specific network will perform better than a user-independent one if tested on its corresponding user. But if there is a requirement for multiple users then a user-independent network will perform better on average than a user-specific one.

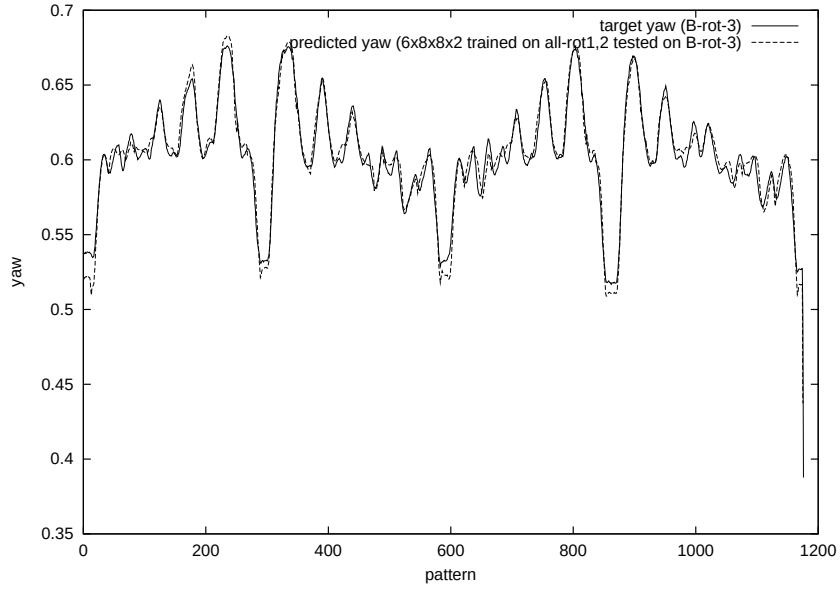
6.4 Task-Specific vs Task-Independent

Now we consider the performance of task-specific networks compared to that of task-independent ones. It is hypothesised that task-specific networks will perform better on the task than the task-independent networks. As with user-specific networks this is because of the relative amounts of information provided to the network during training. By its very definition, a network specific to a particular task will have been presented with more of the required arm movements during training than a task-independent one.

The 3 structures will be trained on user-independent data from either the random task or the rotation task (so either set all-rand-1,2 or all-rot-1,2). For

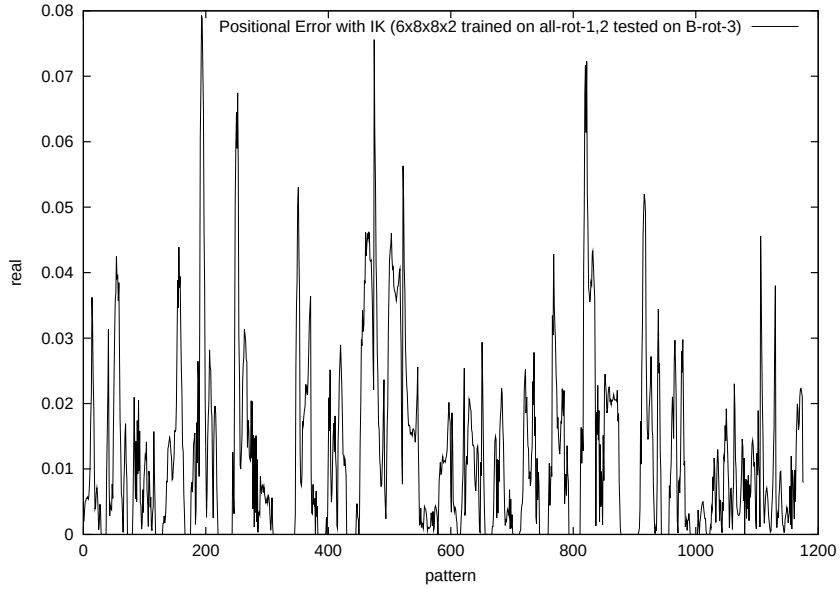


(a) Pitch



(b) Yaw

Figure 6.3: Pitch and yaw prediction of 6x8x8x2 user-independent network on B-rot-3.



(a) Positional error

Figure 6.4: Positional error of 6x8x8x2 user-independent network on B-rot-3.

an added comparison, the networks will also be trained on sets of an equal size containing data from both tasks (i.e. all-both-1,2). The results from the 6x8x8x2 network are tabulated in table 6.4, which shows the mean positional error of the networks on the random and rotation tasks. Full results are given in appendix C, section C.2.2.

Trained on task	Mean positional error on random task (meters)	Mean positional error on rotation task (meters)
Random	0.039020	0.068617
Rotation	0.066172	0.035152
Both	0.041548	0.034755

Table 6.4: Mean positional errors of 6x8x8x2 network.

The results validate the conclusion that a task-specific network will perform better on a specific task than a task-independent network. On the rotation task, the mean error of the task-specific network is half that of the task-independent network, at 3.9 centimeters rather than 6.9 centimeters. A net-

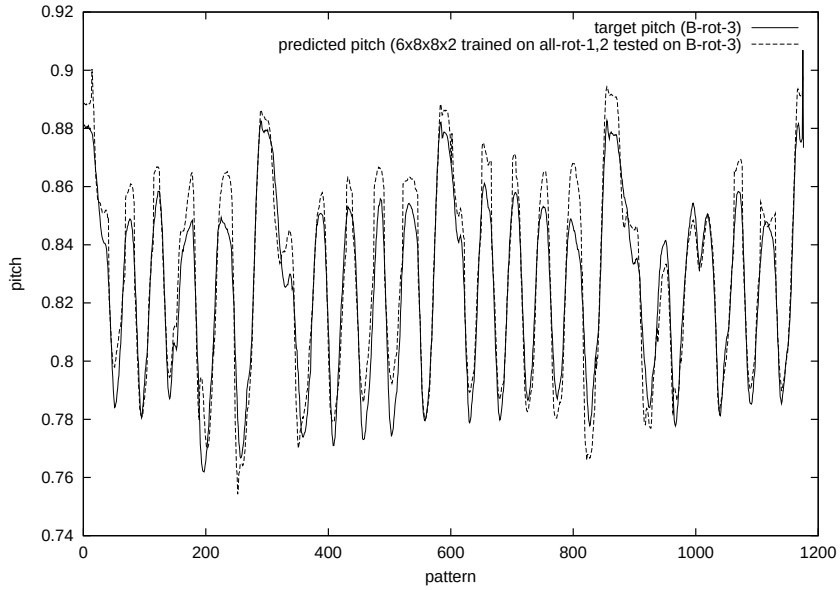
work trained on both random arm movements and data from a specific task has a performance between the two. Again, this is due to the amount of data present in the testing set that is similar to data in the testing set.

Another important point to note is that the task-independent network is better at predicting the movements of the rotation task than the task-specific one is at predicting random arm movements. The task-independent network can generalise better because data from the random task covers a much greater range of movement than data from the rotation task.

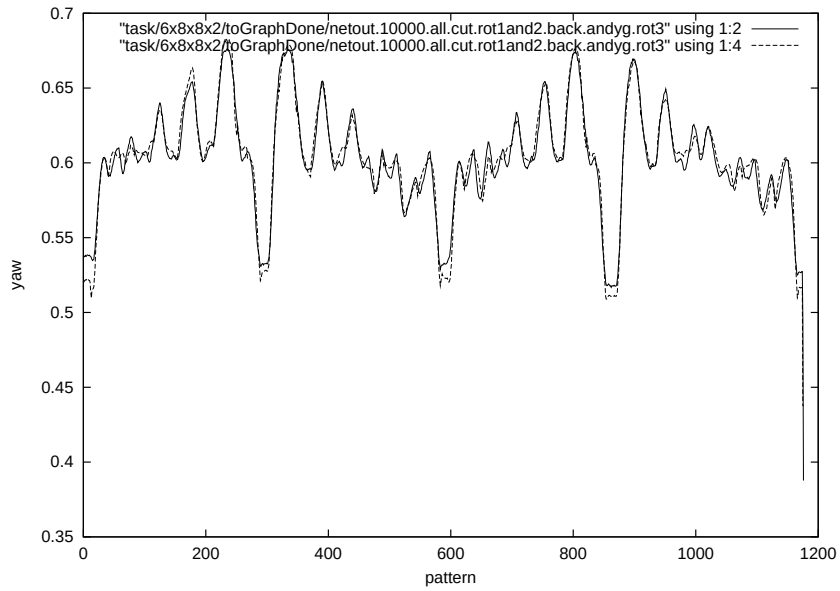
Figure 6.5 shows the predictions of pitch and yaw of the 6x8x8x2 task-specific network on data from participant B performing the rotation task. The performance is good. The paths of pitch and yaw are followed well, although it does have some difficulty when the pitch changes rapidly. Figure 6.6 shows the positional error, which peaks at 8 centimeters.

Figures 6.7 and 6.8 show the performance of the 6x8x8x2 task-independent network tested on the same data. In comparison, the task-independent network performs much worse. It has particular difficulty predicting pitch. The general shape of the movement is followed, but the range in the predicted values is simply too small. The error in predicting pitch is the main reason that the positional error is bad.

The difference in performance of the different network structure is marginal, so it seems that the complexity of the 2-layer networks does not effect their performance a great deal. It is reasonable to select the 6x8x8x2 network for further testing. This is the same structure determined by Beeharee [3] to have good performance on similar tasks.

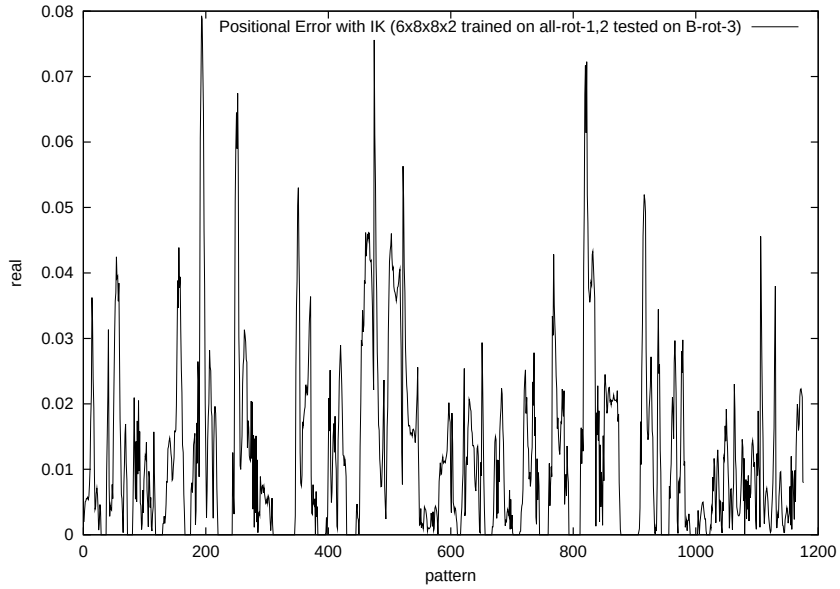


(a) Pitch



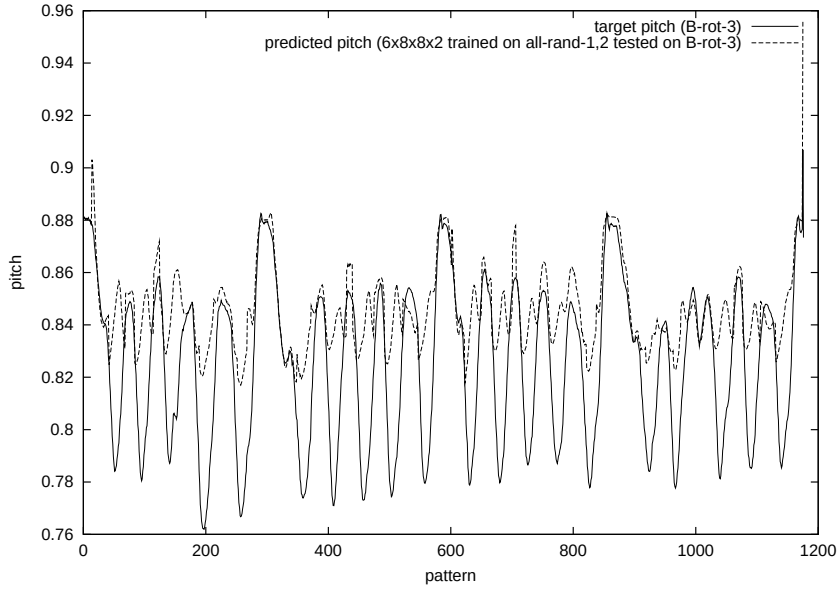
(b) Yaw

Figure 6.5: Performance of 6x8x8x2 task-specific network on B-rot-3.

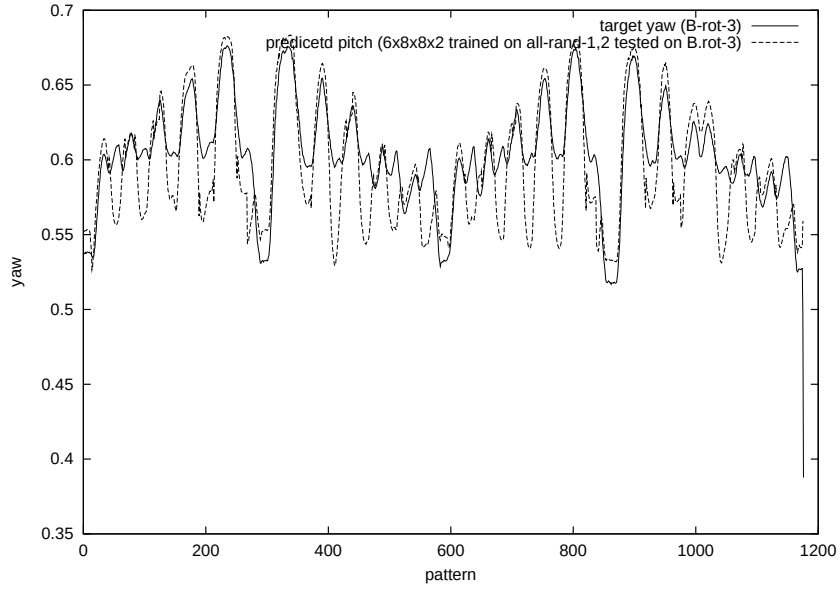


(a) Positional error

Figure 6.6: Positional error of 6x8x8x2 task-specific network on B-rot-3.

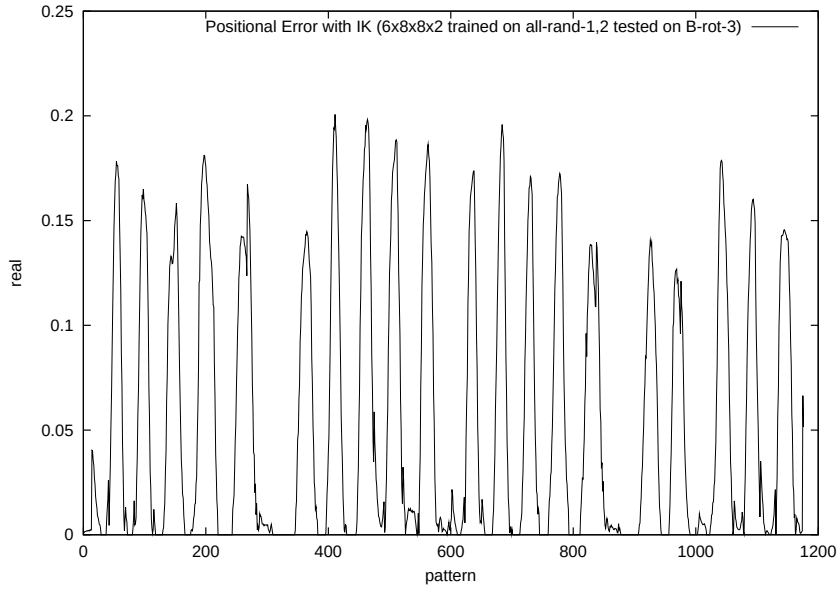


(a) Pitch



(b) Yaw

Figure 6.7: Pitch and yaw prediction of 6x8x8x2 task-independent network on B-rot-3.



(a) Positional error

Figure 6.8: Positional error of 6x8x8x2 task-independent network on B-rot-3.

Chapter 7

Experiment 3 - Testing Generalisation

This experiment tests the best network structure identified in experiment 2 on its generalisation to new users and new tasks.

7.1 Aim

This experiment aims to determine how well the network generalises to new users. It will also determine the ability of the network to generalise to changes in the task. The error caused by re-attaching the sensors can also be determined.

7.2 Description

Different applications have different requirements. A highly specialised task may only have a limited number of users capable of performing them, in which case it is plausible to train user-specific, task-specific networks. Other tasks may be more general, and designed for use by any person. In this case it may be better to train a user-independent, task-independent network. To explore the performance on a range of possible applications I will explore 3 scenarios:

1. The user has trained a user-specific network.
2. The user contributed in training a user-independent network.
3. The user has never used the system before (so they must use a user-independent network).

Each scenario has 2 parts. Part a tests a task-specific network, and part b tests a task-independent network. Only the 6x8x8x2 network, identified in experiment 2, is tested.

In all scenarios the network is trained on data gathered in the initial session (session 1). To determine the error introduced by re-attaching the sensor and re-measuring the offsets, the network is tested on data gathered in a different session (session 2). The sensors were re-attached and the offsets were re-measured, exactly as they would when a user is first set up to use the system. The error introduced by doing so can be determined by comparing the performance of the network on data from session 1 with its performance on data from session 2. Table 7.1 shows the data used for training. The data used for testing is given in table 7.2.

Scenario	Task	Data Set (from original participants A-F)	Training	Validation
1a	Rotation	?-rot-1,2	90%	10%
1b	Random	?-rand-1,2	90%	10%
2a	Rotation	all-rot-1,2	90%	10%
2b	Random	all-rand-1,2	90%	10%
3a	Rotation	all-rot-1,2	90%	10%
3b	Random	all-rand-1,2	90%	10%

Table 7.1: Training data in experiment 3.

7.3 Results

The performance of a 6x8x8x2 network in the scenarios is tabulated in table 7.3.

Scenario	Task	Data Set		Testing
		From original participants A-F	From new participants G-N	
1a	Rotation	?-rot-4	—	100%
		?-highrot-4	—	100%
1b	Random	?-rot-4	—	100%
		?-highrot-4	—	100%
2a	Rotation	?-rot-4	—	100%
		?-highrot-4	—	100%
2b	Random	?-rot-4	—	100%
		?-highrot-4	—	100%
3a	Rotation	—	?-rot-1	100%
		—	?-highrot-1	100%
3b	Random	—	?-rot-1	100%
		—	?-highrot-1	100%

Table 7.2: Testing data in experiment 3.

An increase in error of just under 1 centimeter is observed in most cases when the network is tested on data gathered in session 2. This can be mainly attributed to errors in the measurement of the sensor offsets. The users' dimensions remain the same as they are simply read back from a file in which the original measurements were stored.

For all scenarios the task-specific network (in part a) produces a lower error than the task-independent one (in part b), confirming the results of experiment 2. The best performance is seen in scenario 1a, where the network is both task-specific and user-specific. For the other task-specific scenarios the error ranges from 4.26 to 6.32 centimeters.

Comparing part b of all scenarios reveals that when the network is task-independent, whether it is user-independent or not makes little difference to its performance. The use of a task-independent scenario increases the errors, but also brings them closer to one another, to just under 8 centimeters.

The similarity in error of the task-independent networks reflects on their generalisation ability. In session 2, the difference in error of task-independent networks between the rotation task and the high-rotation task is in the range of only 0.06 to 0.72 centimeters. In comparison, for task-specific networks the

Table 7.3: 6x8x8x2 performance for the 3 scenarios.

Scenario	Same session	Different session	
	Mean positional error on rotation task (meters)	Mean positional error on rotation task (meters)	Mean positional error on high-rotation task (meters)
1a	0.017741	0.013978	0.051728
1b	0.068102	0.078419	0.077779
2a	0.035152	0.042601	0.059812
2b	0.068617	0.076433	0.070736
3a	—	0.055065	0.063245
3b	—	0.075304	0.082517

range is between 0.81 and 3.7 centimeters. This confirms that task-independent networks are better at generalising to new tasks than task-specific ones.

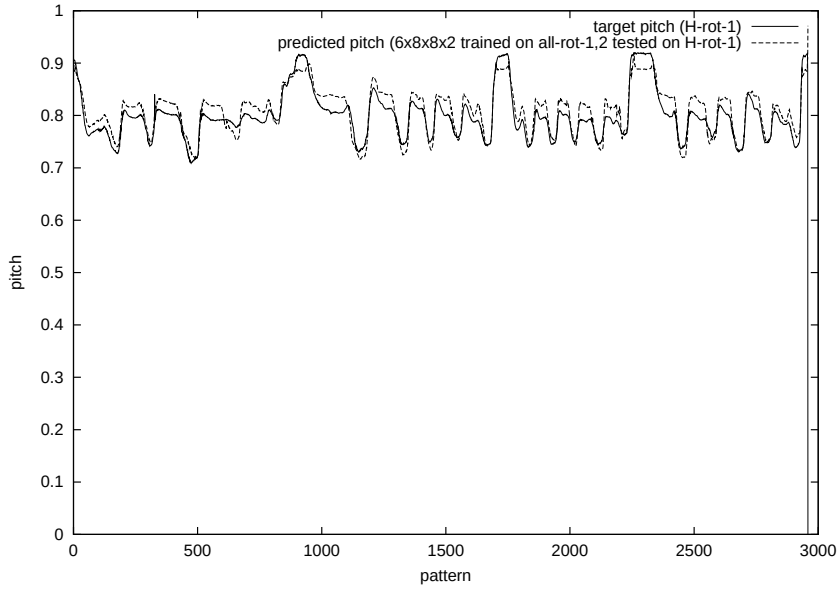
Interestingly, the comparative performance on new users shows a similar trend. In session 2, the change in error of a task-independent network between an existing participant and a new participant ranges from -0.11 to 1.46 centimeters (note that for the rotation task, the error on a new participant is marginally better than that on an original participant, with a decrease in error of 0.11 centimeters). The task-specific networks have a change in error that is in the region of 0.34 to 4.11 centimeters. This indicates that the task-independent network is also better at generalising to new users, perhaps because of the greater range of motion provided by the random task.

Despite the better generalisation ability of the task-independent networks, their performance remains consistently worse than the task-specific networks. The similarity of the high-rotation task to the rotation task means that the task-specific networks can still perform it with a reasonable degree of accuracy (in the range of 5.17 to 6.32 centimeters).

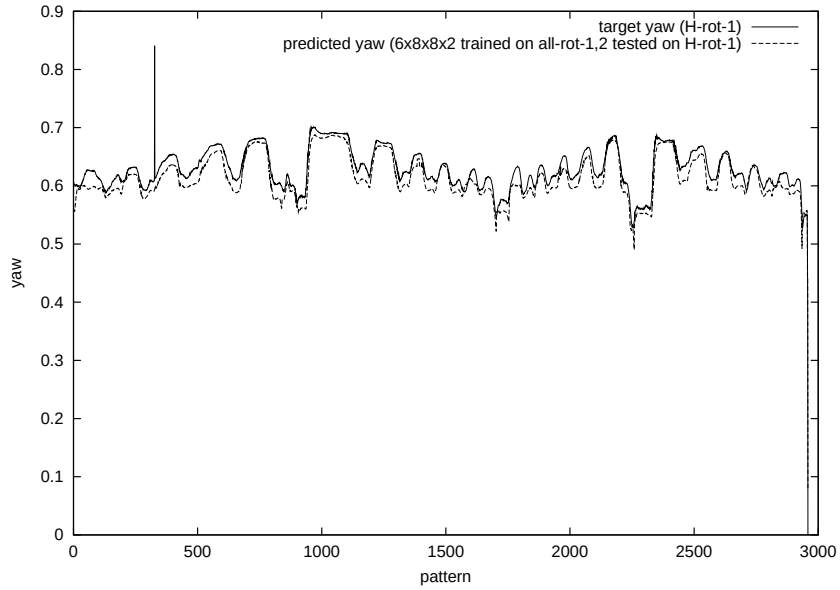
It is assumed that scenarios 3 is the most relevant to the majority of virtual reality applications. It is rare to find a system that can only be used by a handful of people. The best performance from the networks trained for scenario 3 is seen when the system is used by participant H. Figure 7.1 shows the results of the network from scenario 3a (user-independent and task-specific) at predicting the pitch and yaw of participant H's upper arm when performing the rotation task for the first time. You can see from graphs a and b that pitch and yaw is predicted well by the network, with deviations of at most 0.03 (or 10°). Figure 7.2 shows the corresponding positional (real) error of the network[1]. The maximum positional error is 1 centimeter, but this is not representative of the majority of the predictions. The changes in error between each pattern (i.e. between each time step) are small, leading to smooth movement.

Figure 7.3 shows the predictions of pitch and yaw of the network from scenario 3b (user-independent and task-independent) when tested on the same set from

[1]Please note that the error spike around pattern 350 is caused by an anomalous yaw value in the testing set.



(a) Pitch



(b) Yaw

Figure 7.1: Pitch and yaw prediction of (6x8x8x2) network from scenario 3a on H-rot-1.

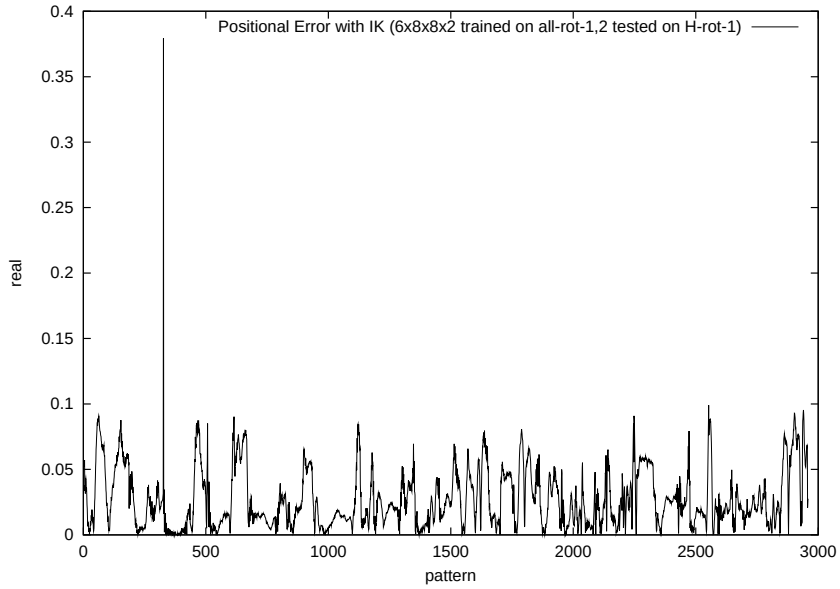
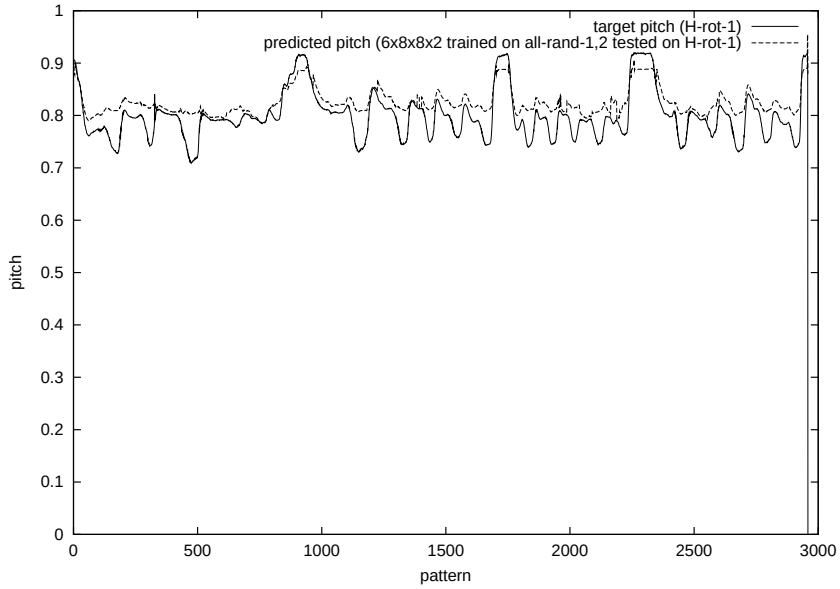


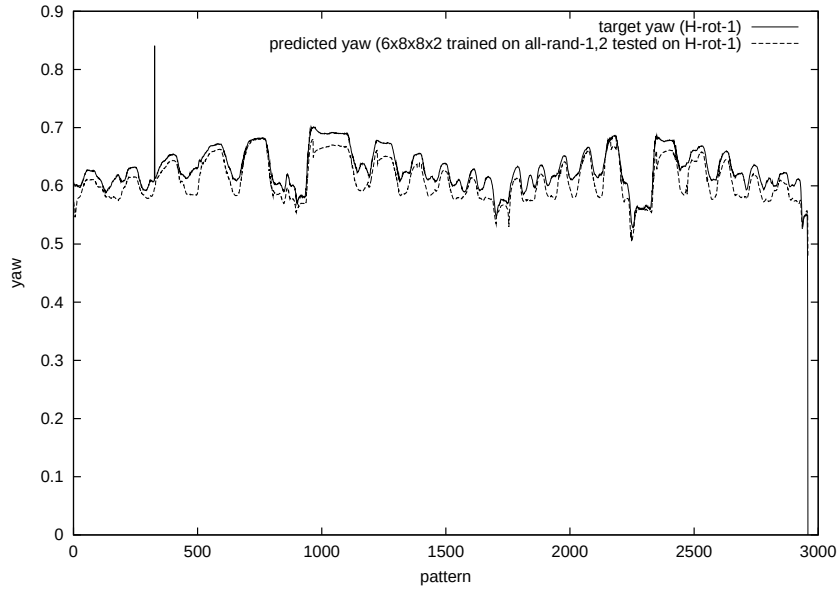
Figure 7.2: Positional error of (6x8x8x2) network from scenario 3a on H-rot-1.

participant H[2]. As would be expected, the prediction of pitch and yaw is worse. The predication of pitch in particular follows the peaks but not the troughs of the path. This results in an increase in the positional error, shown in figure 7.4, the maximum of which is now 2.5 centimeters. Again, the error of most of the prediction is lower than this, and the change in error at each time step is such that the movement is smooth.

[2]Again, the error peak at pattern 350 is caused by an anomalous yaw value in the testing set.



(a) Pitch



(b) Yaw

Figure 7.3: Pitch and yaw prediction of (6x8x8x2) network from scenario 3b on H-rot-1.

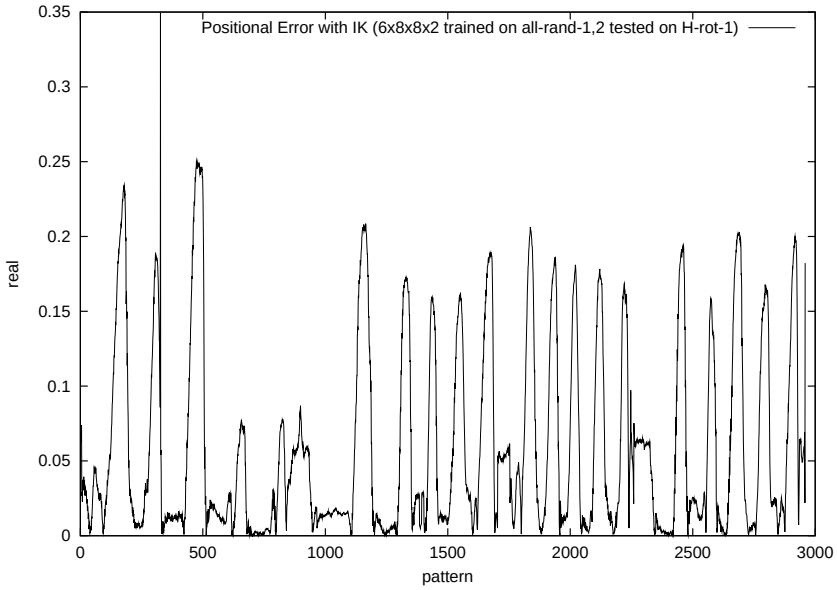


Figure 7.4: Positional error of (6x8x8x2) network from scenario 3b on H-rot-1.

Chapter 8

Conclusions

8.1 Discussion

This project has successfully explored the application of neural networks in predicting elbow position. A variety of neural network structures were tested on a range of users and tasks. The results of 3 experiments determined that a neural network with 2 hidden layers has the potential to predict elbow position with an average error of only 1.4 centimeters.

Experiment 1 determined that a network with only a single hidden layer can not learn the complexity of human arm movement. The high dimensionality of the data requires a network with some kind of hierarchical structure, such as a network with 2 hidden layers. It is likely that the input space is encoded by the first hidden layer to reduce its complexity before presentation to the second layer.

A suitable representation of upper arm orientation proved to be very important. The original specification could not be learnt successfully by any of the neural network structures. This problem was solved with the introduction of an altered specification.

Experiments 2 and 3 determined that a user-specific network performs better than a user-independent network, but does not generalise as well to different users. Similarly, a task-specific network performs better than a task-independent network but does not generalise as well to variations in the task.

Even so, the task-specific networks performed consistently better than the task-independent ones. The nature of the task-independent networks' inaccuracy in predicting pitch indicated that data gathered from random arm movements was not representative enough of the movements required by the structured task. A possible solution is to train the task-independent network on data gathered from a range of different tasks. The data would cover a wider range of arm movement and hence be more suitable for training a task-independent network.

The best performance can be achieved by a task-specific, user-specific network, which has an average error of only 1.4 centimeters. However, the majority of virtual reality applications have the requirement of multiple users. In this case, each user would have to perform an example of the task before using the system, so that a network could be trained specifically for them. This approach, while producing very accurate arm movement, violates my requirement of a system that is quick initialise and start using.

It is more likely that a system will want to allow multiple users. In this case, data gathered from several users performing movements common to the application would be used to train a network. The resulting task-specific, user-independent network will perform with an average error of 4.2 centimeters if the user contributed in the training data, and 5.5 centimeters if they didn't. The estimated tolerance range is ± 4 centimeters in the y-direction [3]. Given that the positional errors in this project are a measurement of displacement in *any* direction, the network should perform within the tolerance range most of the time. The use of a task-independent, user-independent network will produce errors of about 7 centimeters on average.

All of the resulting networks produce smooth movements that follow the path of the user's arm. Errors in the network's predictions are gradual, and have the effect of the arm not moving as much as it should, or moving too far. There are no sudden jumps in its position.

The error introduced by the initial set-up of a user was to found to be roughly 1 centimeter. This is surprisingly good, showing that a calibration technique, such as the one used by Beeharee [3], may not be necessary

8.2 Improvements

8.2.1 Yaw Specification

With the current specification of upper arm rotation, a sudden change in yaw from 0° to 360° is difficult but not impossible (see section 5.3.1). It is desirable to have a specification that is defined so this can never occur.

8.2.2 IK Correction

The IK correction algorithm (section 2.5.2) consistently reduces the error of the neural network prediction by several centimeters. However, the use of a more sophisticated technique, that perhaps takes into account past elbow position, could reduce the error further.

8.3 Future Work

8.3.1 Neural Network

Movements from one moment in time are inherently related to previous movements so the use of a system that takes into account past movement could prove beneficial. One example of such a system is the recurrent network. These are similar to MLPs, except information encoded in the network in one time step is fed back as inputs for the next time step. The NeuroAnimator system successfully uses recurrent networks to model the dynamics of physical systems [12, 11]. Elman networks are another example [10] and have been applied successfully in the field of natural language processing.

Modular networks [17, 13, 29] consist of a number of separate modules (each of which is a small neural network) governed by a so-called "gating network". Different areas of the input space are allocated to different modules for prediction. These types of network could prove particularly useful in predicting the movement of several body parts at the same time.

8.3.2 Legs

The existing system animates a person's upper body only. In most applications, movement of the legs is determined by a simple IK based algorithm, or by following the oscillation of a spline. But for applications that require more accuracy, it would be useful to extend the system to include neural networks capable of modelling leg movements.

8.4 Summary

This project has shown that neural networks are a viable approach for modelling human arm movement. They have the potential to predict arm movement to within only a couple of centimetres. Future work is needed to extend the approach to the whole body.

Appendix A

Calculations

A.1 User Measurements

Participant	Shoulder width l_s	Upper arm length l_{ua}	Lower arm length l_{la}
A (steph)	0.21	0.37	0.31
B (andyg)	0.225	0.34	0.26
C (andygo)	0.18	0.33	0.27
D (caz)	0.18	0.305	0.255
E (daniel)	0.18	0.29	0.265
F (helen)	0.17	0.29	0.24
G (ashwin)	0.185	0.34	0.275
H (barbara)	0.175	0.31	0.24
I (bertine)	0.175	0.33	0.26
J (dave)	0.1925	0.385	0.305
K (davel)	0.215	0.35	0.28
L (jonne)	0.20	0.375	0.30
M (kim)	0.1925	0.38	0.28
N (markku)	0.2025	0.335	0.285

A.2 Avatar Scaling

The scaling factor s of a part is calculated by simply looking at the ratio of the part's desired length l^d and the current length l^c,

$$s = \frac{l^d}{l^c}$$

This scaling factor is used to define a scaling matrix $[S]$,

$$[S] = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling is achieved by multiplying the part's position matrix $[P]$ by this scaling matrix,

$$[P^{scaled}] = [P][S]$$

It was decided to implement scaling in this manner because it does not require any modification of MAVERIK code (stated in section 2.4 as a desired feature of my code)[1]. Because the parts are relative to one another, parts further up the hierarchy scale with those below. To prevent this, and to allow a part to define its own scaling factor, the position matrices of parts higher up the hierarchy must undo the effects of the scaling of previous parts. This is done by simply applying the inverse of the previous parts scale matrix before applying the part's own. Scaling of the shoulders is achieved by scaling the hip part by the shoulder's scaling factor. This results in a scaling of the whole body. The arm parts undo this scaling and apply their own scaling factor to achieve accurate arm lengths. The scaling process is shown in figure 1.

[1] It should also be noted that the position matrices are not being changed in any other way, and should not be. A change in the positional components would result in parts detaching - as well as being physically inaccurate it would also probably be relatively disturbing!

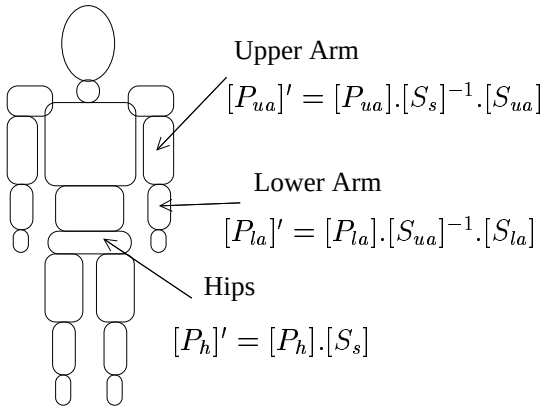


Figure 1: Avatar scaling.

A.3 Tracking Details

A.3.1 Initialisation

The user is asked to sit upright when the sensors are initialised so that a transformation can be calculated to correct the orientation of the torso sensor. The torso sensor is not mounted vertically upright on the users chest, but on a slope. The continuous application of this transformation during tracking ensures that the avatar is upright when the user is, even though the torso sensor is not. It only corrects for orientation and not position, so it is still important that the sensor is mounted in the center of the users chest. This translation step is unnecessary for the other sensors as their orientations are already aligned with the body part they are tracking.

Define a translation $[T]$ that maps from an upright orientation $[U]$ to the torso sensor's initial orientation $[I]$ as,

$$[T] = [U]^{-1}[I]$$

where the positional components of $[U]$ and $[I]$ are all zero.

The translation required to correct the torso sensor orientation, $[C]$, is simply the inverse of this,

$$[C] = [T]^{-1}$$

The application of $[C]$ effectively rotates the torso sensor by the difference between the initial orientation $[I]$ and the upright orientation $[U]$, as shown in figure 2.

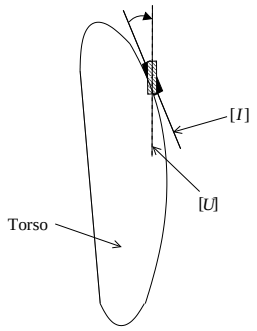


Figure 2: Torso correction.

A.3.2 Tracking

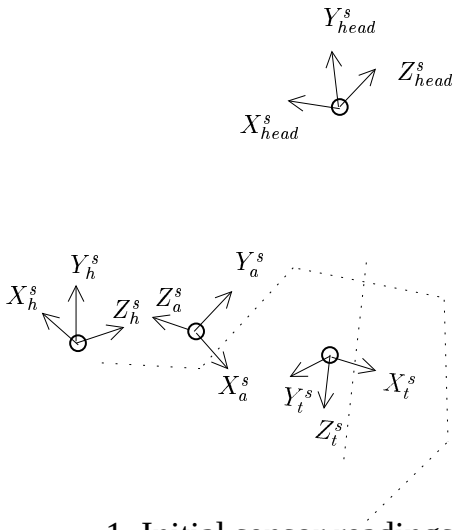
Figure 3 shows the steps necessary to offset the sensors to the positions they represent. The offsetting of all sensors but the upper arm sensor is done in avatar co-ordinate system. The application of such a translation to the upper arm sensor is unnecessary as we are only interested in its position and never require its orientation. Each sensor has an associated translation that maps from sensor co-ordinates to avatar co-ordinates.

These steps are illustrated using the torso sensor as an example.

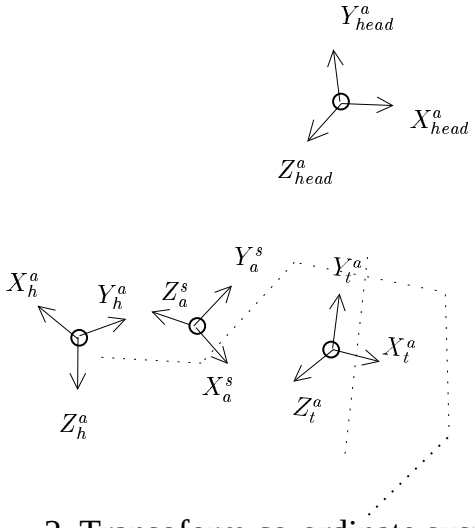
Step 1: The sensors are read to determine their positions and orientations in world co-ordinates. For each sensor this gives a matrix $[M^s]$ from which you can determine[2] the position, specified by 3 co-ordinates x^s, y^s, and z^s, and the orientation, specified by the 3 axes X^s, Y^s, and Z^s.

Step 2: For each sensor a matrix $[A]$ is defined to translate from sensor to avatar co-ordinates. The translation swaps the axes of the sensor so that they

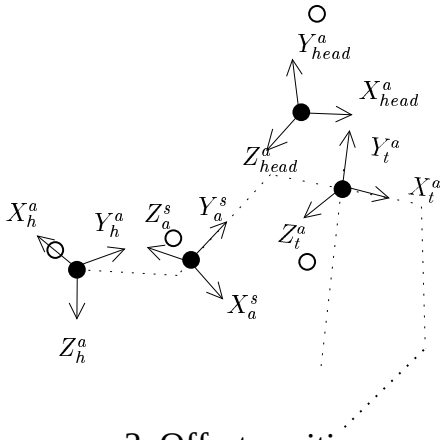
[2]MAVERIK provides functions for querying and manipulating matrices and vectors [15].



1. Initial sensor readings



2. Transoform co-ordinate systems



3. Offset positions

Figure 3: Offset calculations.

represent the axes of the corresponding avatar part. The sensor's matrix is multiplied with $[A]$ to achieve the desired translation,

$$[M^a] = [M^s][A]$$

For example, for the torso sensor the desired axes changes are,

$$X_t^a = X_t^s$$

$$Y_t^a = -Z_t^s$$

$$Z_t^a = Y_t^s$$

So $[A_t]$ is defined as,

$$[A_t] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is applied to produce the torso sensor's matrix in avatar co-ordinates[3],

$$[M_t^a] = [M_t^s][A_t]$$

Step 3: The position of the sensors are offset so that their positions line up with the user's bones. Generally, the desired effect is,

$$x^o = x^a + ox$$

$$y^o = y^a + oy$$

$$z^o = z^a + oz$$

where x^a, y^a, and z^a are the position components of $[M^a]$, and ox, oy, and oz are the offsets in the X^a, Y^a and Z^a directions respectively.

[3]Actually, the torso sensor also has the correction of its orientation applied, so the full calculation is $[M_t^a] = [M_t^s].[C].[A]$. This only applies to the torso sensor and so it is not illustrated in the main text for the sake of clarity.

A matrix $[O]$ is defined to perform this offsetting,

$$[O] = \begin{bmatrix} 1 & 0 & 0 & ox \\ 0 & 1 & 0 & oy \\ 0 & 0 & 1 & oz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with which $[M]^a$ is multiplied,

$$[M^o] = [M^a][O]$$

For example, for the torso, oz_t is negative and there is no offset along the x-axis,

$$x_t^o = x_t^a$$

$$y_t^o = y_t^a + oy_t$$

$$z_t^o = z_t^a + oz_t$$

so $[O_t]$ is,

$$[O_t] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & oy_t \\ 0 & 0 & 1 & oz_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is applied to produce the final offset torso sensor matrix,

$$[M_t^o] = [M_t^a][O_t]$$

The result of this process gives us a final matrix $[M^o]$ corresponding to each of the sensors, from which the positions and orientations of the user's body parts can be determined to calculate a model of the upper body, described in section 2.4.3.

A.4 Determining Lower Arm Roll

First, the vector EH is used to calculate the y-axis of the lower arm,

$$Y_a = -\frac{EH}{|EH|}$$

Next, the normal between the z-axis of the hand and the y-axis of the upper arm gives us the x-axis of the upper arm,

$$X_a = Z_h \times Y_a$$

Finally, the cross product between the x-axis and y-axis of the lower arm gives us its z-axis,

$$Z_a = X_a \times Y_a$$

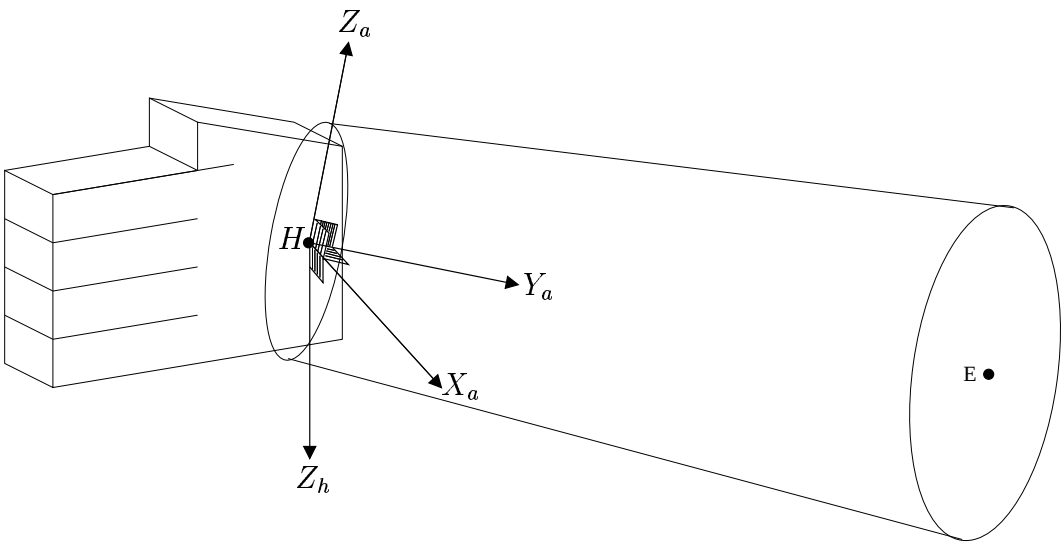


Figure 4: Determining lower arm roll.

Appendix B

Backpropagation

Before training begins the weights of the neural network are initialised to small random values (between -0.5 and 0.5).

Training consists of a forward and a backward pass through the network. The forward pass propagates a pattern $(\overrightarrow{x}, \overrightarrow{t})$ through the network and calculates the output o_u of every node u in the MLP.

The backward pass propagates the error backwards through the network. The error δ_k for each output unit k is,

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{1}$$

where t_k is the target of node k and o_k is its actual output.

The error for each hidden unit h is,

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in ouput} w_{kh}\delta_k$$

where w_{kh} is the weight between node h and node k.

The error is used to adjust each weight w_{ji} in the network by adjusting them by an proportional amount,

$$w_{ji} \leftarrow w_{ji} + \triangle w_{ji}(n)$$

where

$$\triangle w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \triangle w_{ji}(n-1)$$

where $\triangle w_{ji}(n)$ is the weight update in the nth epoch of training, x_{ji} in the input from node i into node j.

This occurs for all of the patterns in the training set, for every epoch of training. The process repeats until some stopping criteria has been met, such as the when the maximum number of epochs has been reached, or the validation error is low enough (see section 3.3.1).

Appendix C

Full Results

C.1 Experiment 1

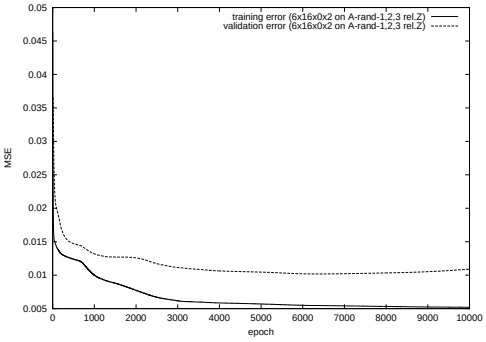
C.1.1 Initial Results

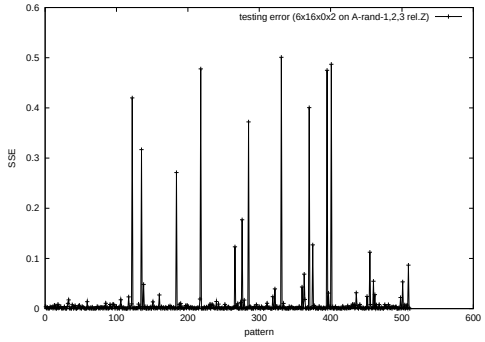
Experiment 1 Summary

Network	Lowest validation error (MSE)		
	A-rand-1-2-3	A-rot-1-2-3	A-both-1-2-3
6x8x2	0.015	0.010	0.013
6x16x2	0.010	0.012	0.012
6x24x2	0.007	0.013	0.010
6x32x2	0.009	0.012	0.010
6x4x4x2	0.015	0.012	0.014
6x8x8x2	0.009	0.010	0.010
6x12x12x2	0.006	0.010	0.009
6x16x16x2	0.007	0.009	0.009
6x20x20x2	0.007	0.009	0.009
6x24x24x2	0.005	0.002	0.008

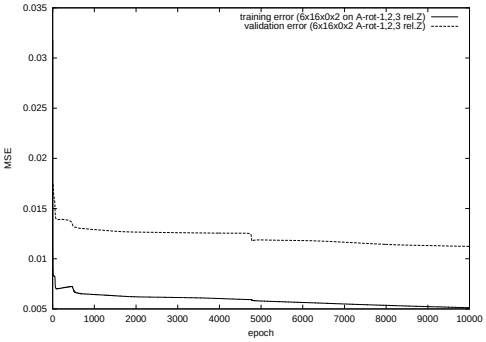
Example of 6x16x0x2 Training and Testing

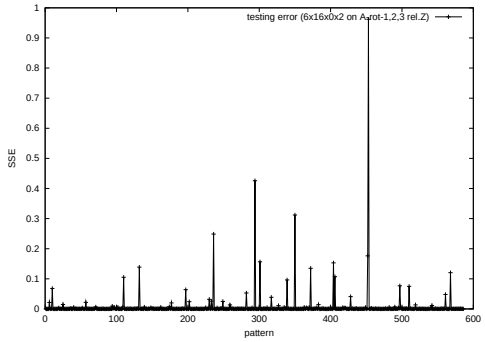
Random



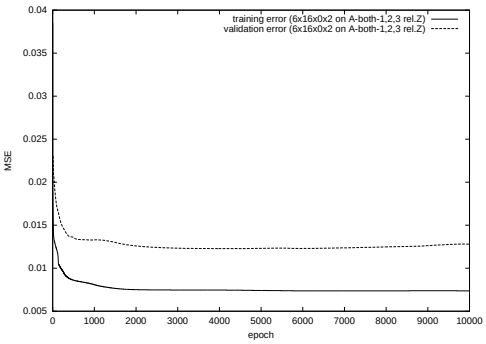


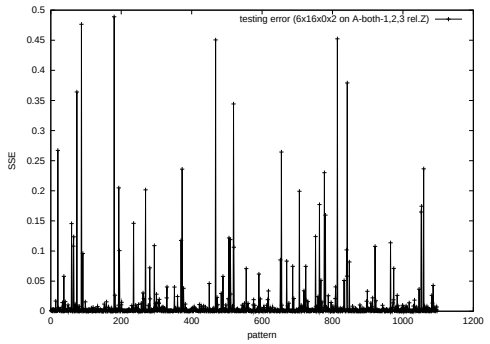
Rotation





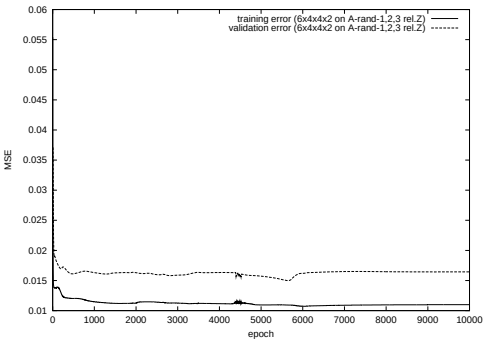
Both

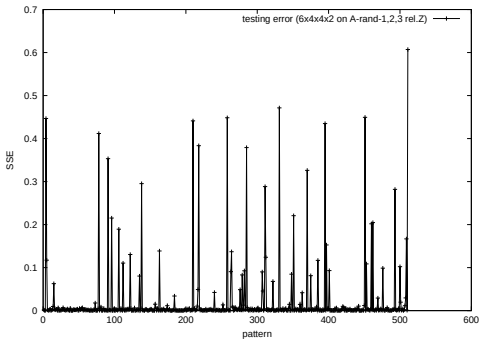




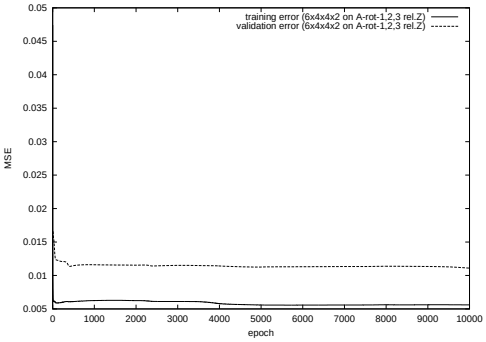
Example of 6x4x4x2 Training and Testing

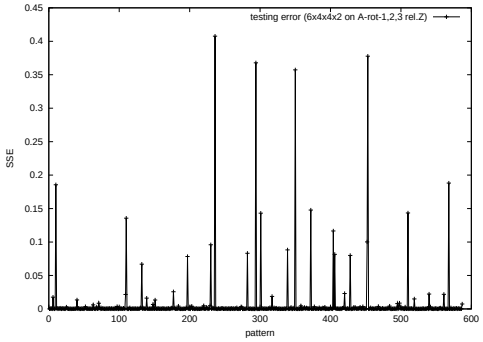
Random



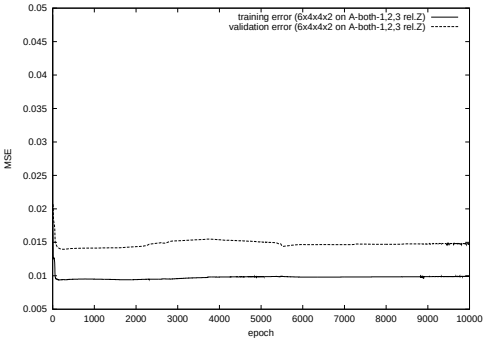


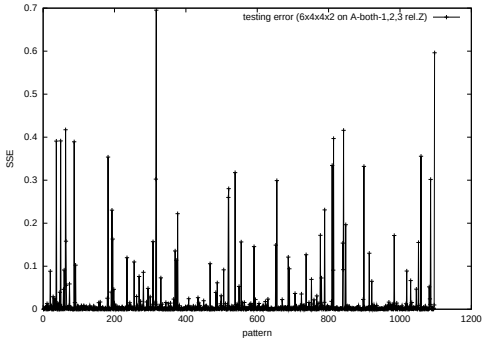
Rotation





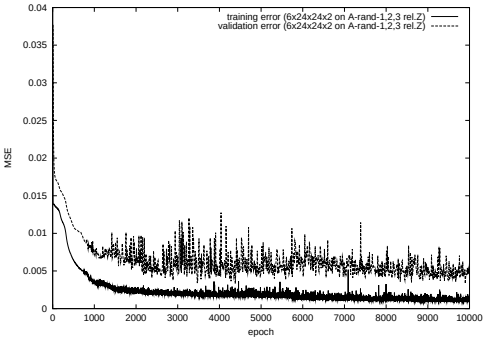
Both

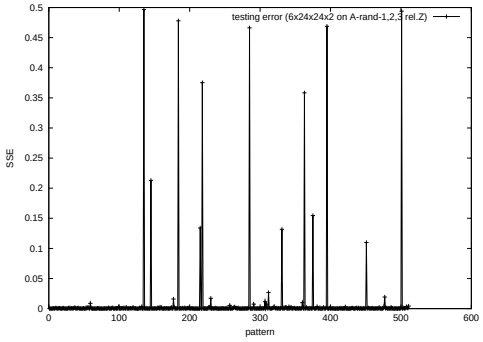




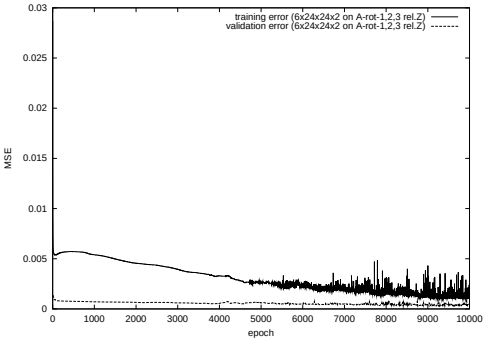
Example of 6x24x24x2 Training and Testing

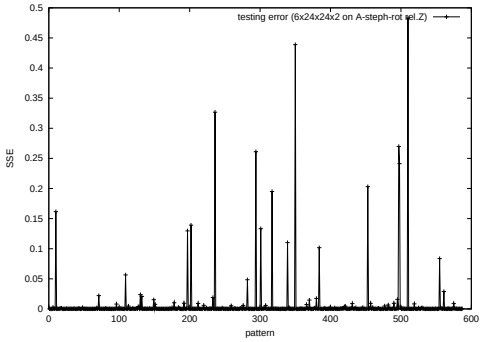
Random



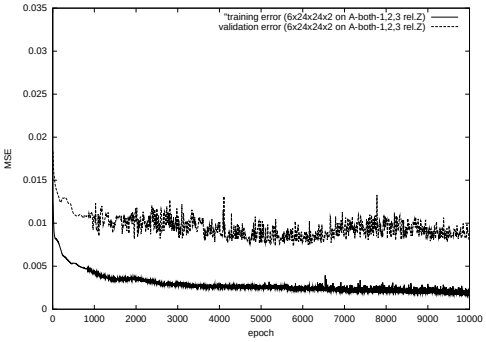


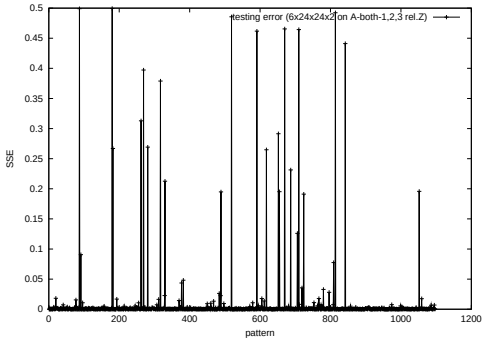
Rotation



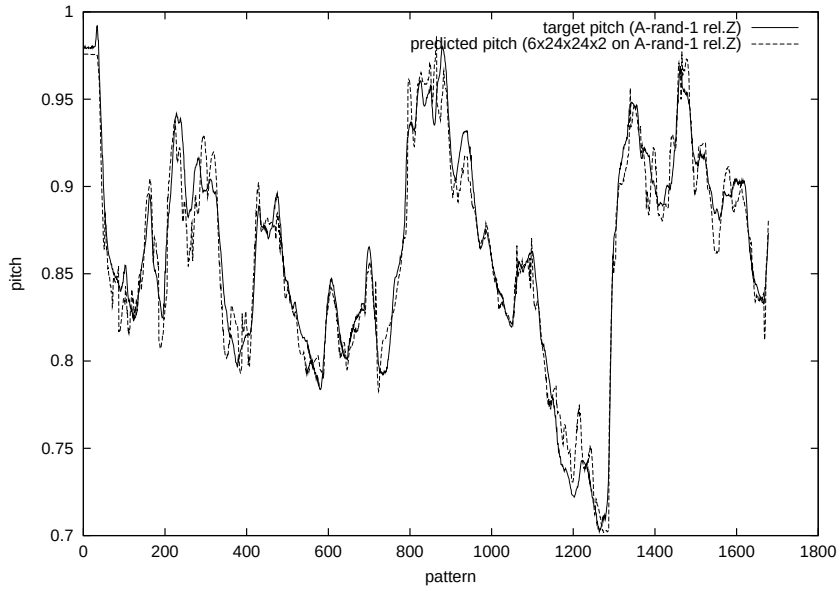


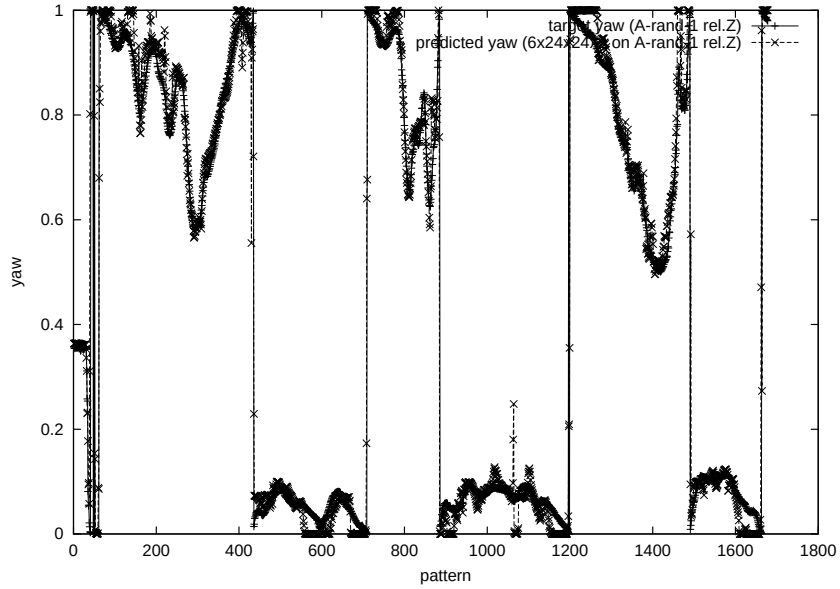
Both





Example of 6x24x24x2 Pitch and Yaw Prediction





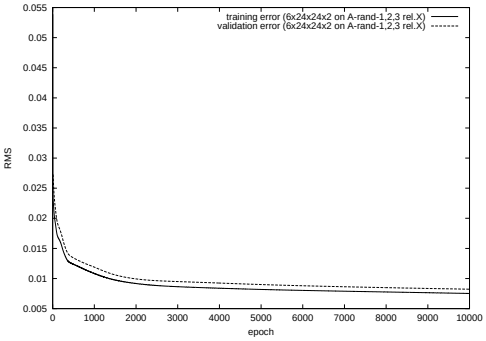
C.1.2 Re-training

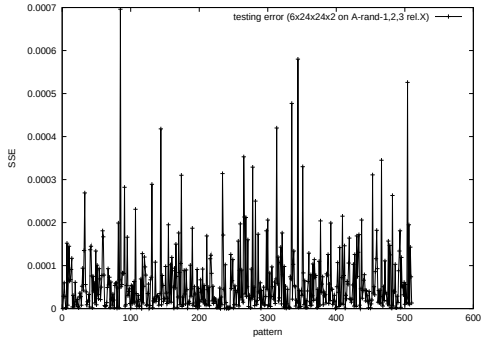
Experiment 1 Re-training Summary

Network	Lowest validation error (RMS)		
	A-rand-1-2-3	A-rot-1-2-3	A-both-1-2-3
6x8x8x2	0.010	0.006	0.010
6x12x12x2	0.010	0.006	0.009
6x16x16x2	0.009	0.006	0.009
6x20x20x2	0.009	0.006	0.008
6x24x24x2	0.008	0.006	0.008

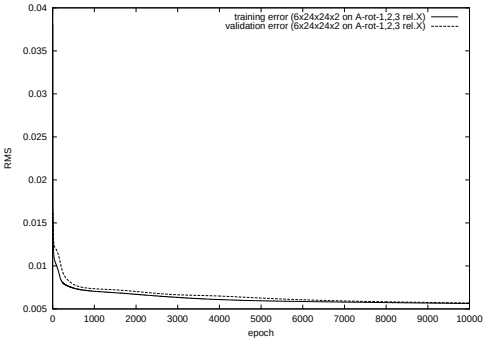
Example of 6x24x24x2 Training and Testing

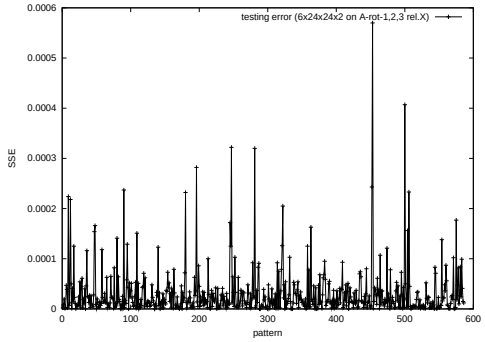
Random



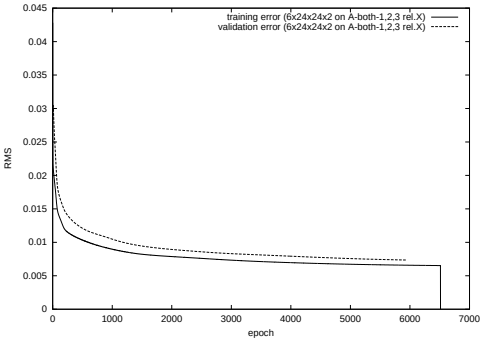


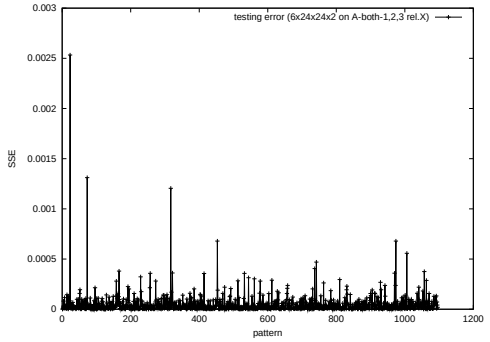
Rotation



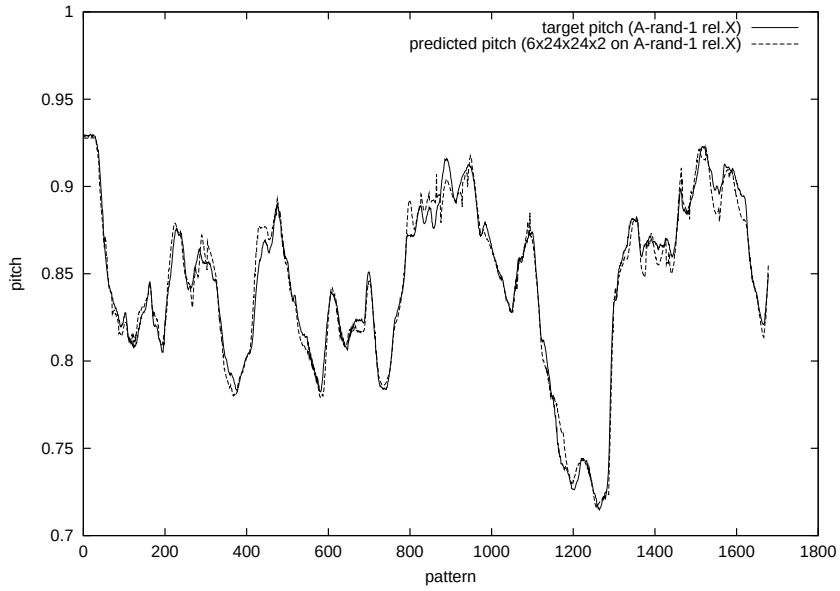


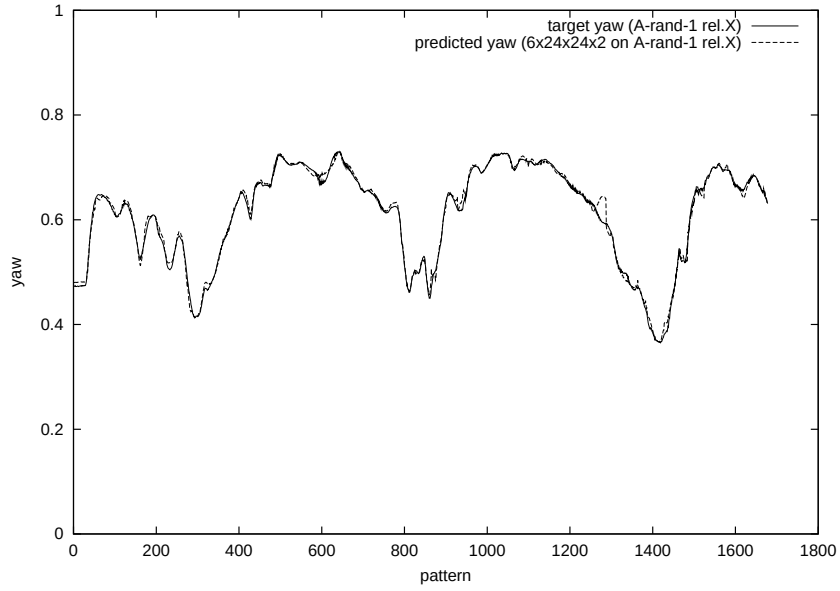
Both





Example of re-trained 6x24x24x2 Pitch and Yaw Prediction





C.2 Experiment 2

C.2.1 User-Specific vs User-Independent

User-Specific/Independent 6x8x8x2 Results

Trained on participant	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
A	A-rot-1,2	A-rot-3	0.040119	0.034537
	"	B-rot-3	0.082413	0.041151
	"	C-rot-3	0.062992	0.045343
	"	D-rot-3	0.072478	0.051334
	"	E-rot-3	0.136126	0.103207
	"	F-rot-3	0.075702	0.031227
B	B-rot-1,2	A-rot-3	0.121719	0.105702
	"	B-rot-3	0.018160	0.008828
	"	C-rot-3	0.101151	0.078292
	"	D-rot-3	0.081131	0.052756
	"	E-rot-3	0.115567	0.080347
	"	F-rot-3	0.061989	0.032971
C	C-rot-1,2	A-rot-3	0.058483	0.038762
	"	B-rot-3	0.074687	0.029693
	"	C-rot-3	0.020500	0.014978
	"	D-rot-3	0.062150	0.031291
	"	E-rot-3	0.133438	0.098235
	"	F-rot-3	0.083534	0.032313
D	D-rot-1,2	A-rot-3	0.083640	0.064084
	"	B-rot-3	0.128572	0.070447
	"	C-rot-3	0.079723	0.036920
	"	D-rot-3	0.032585	0.020558
	"	E-rot-3	0.132812	0.089298
	"	F-rot-3	0.044935	0.018248

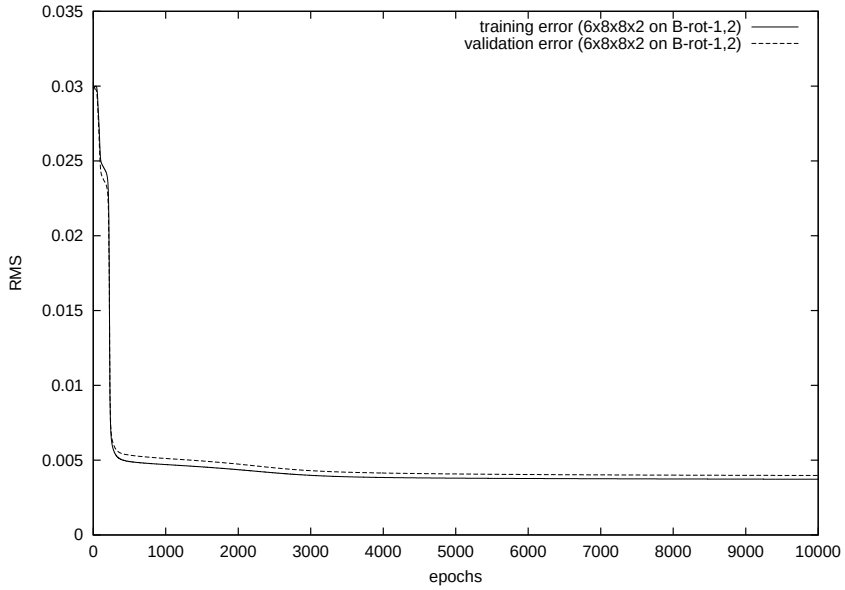
Trained on participant	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
E	E-rot-1,2	A-rot-3	0.196447	0.162808
	"	B-rot-3	0.116020	0.061366
	"	C-rot-3	0.170377	0.146013
	"	D-rot-3	0.142878	0.112901
	"	E-rot-3	0.029210	0.019062
	"	F-rot-3	0.158870	0.090553
F	F-rot-1,2	A-rot-3	0.095653	0.054545
	"	B-rot-3	0.079513	0.030097
	"	C-rot-3	0.114594	0.034189
	"	D-rot-3	0.068562	0.040472
	"	E-rot-3	0.144544	0.098880
	"	F-rot-3	0.019058	0.008482
All	All-rot-1,2	A-rot-3	0.077139	0.066033
	"	B-rot-3	0.029742	0.011905
	"	C-rot-3	0.042105	0.032734
	"	D-rot-3	0.043317	0.030064
	"	E-rot-3	0.076091	0.053525
	"	F-rot-3	0.041604	0.016650

User-Specific/Independent 6x8x8x2 Summary

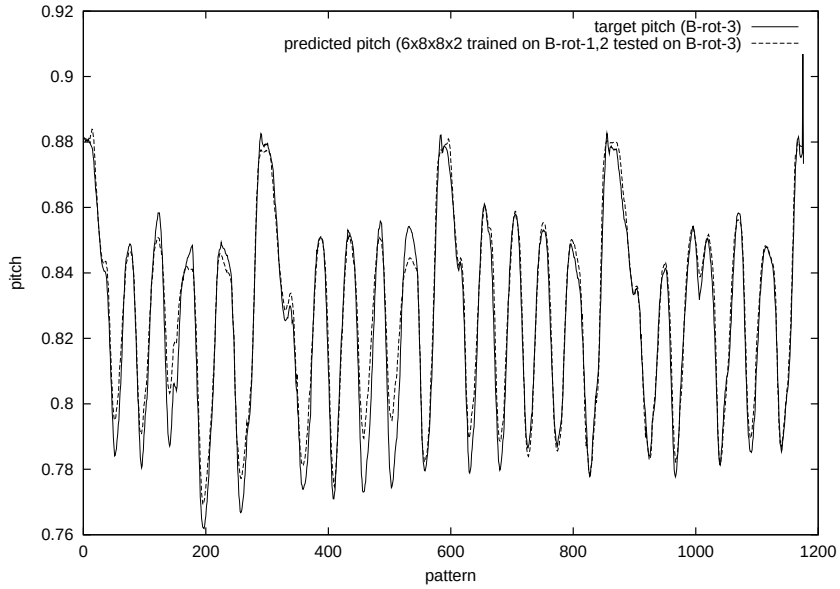
Trained on participant	Positional error on same participant (meters)		Mean positional error on other participants (meters)	
	Without IK	With IK	Without IK	With IK
A	0.040119	0.034537	0.071619	0.054452
B	0.018160	0.008828	0.096311	0.070014
C	0.020500	0.014978	0.082458	0.046059
D	0.032585	0.020558	0.093936	0.055799
E	0.029210	0.019062	0.156918	0.114728
F	0.019058	0.008482	0.100573	0.051637
All	—	—	0.051666	0.035152

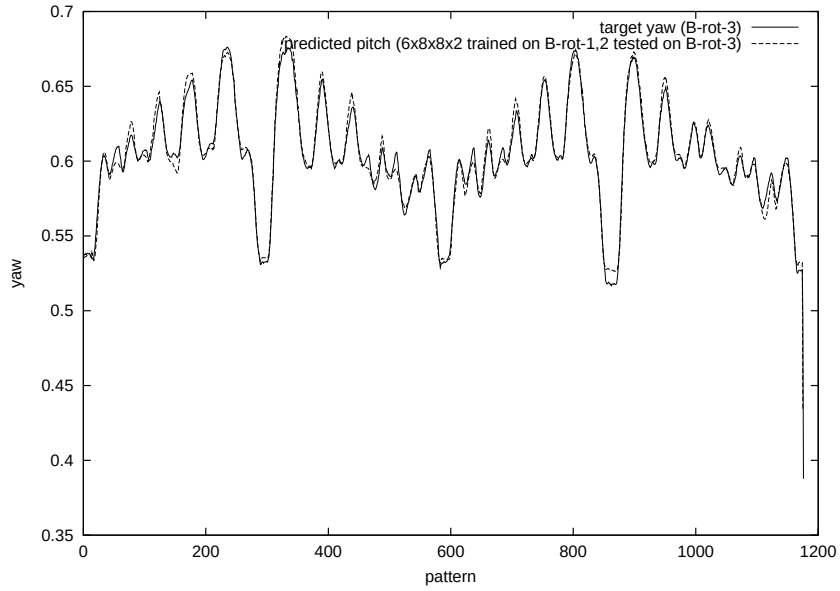
User-Specific 6x8x8x2 Performance (Trained on B-rot-1,2. Tested on B-rot-3.)

Training and Validation Error

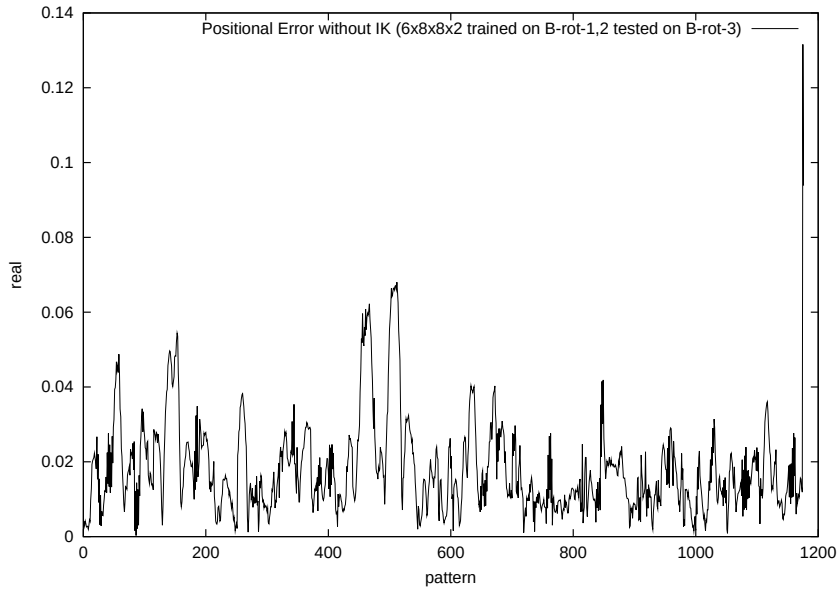


Pitch and Yaw Prediction

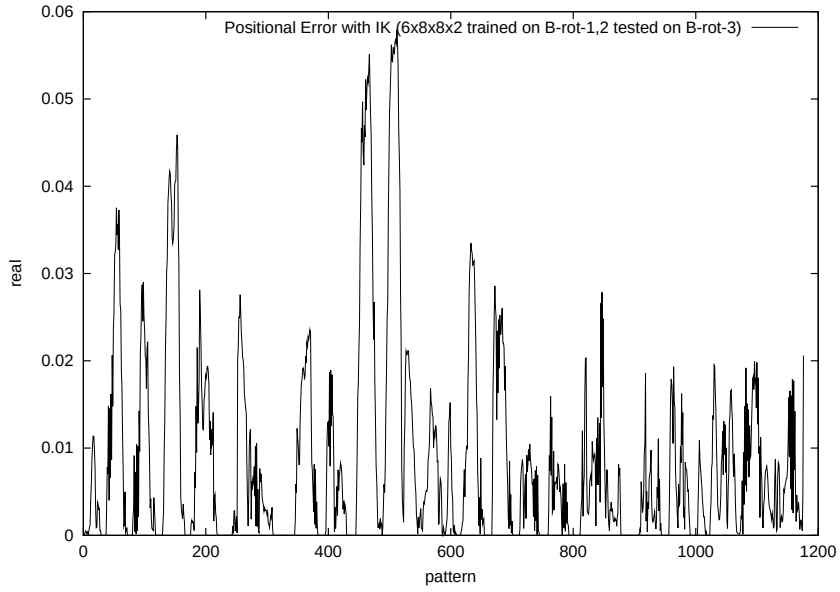




Positional Error (without IK)

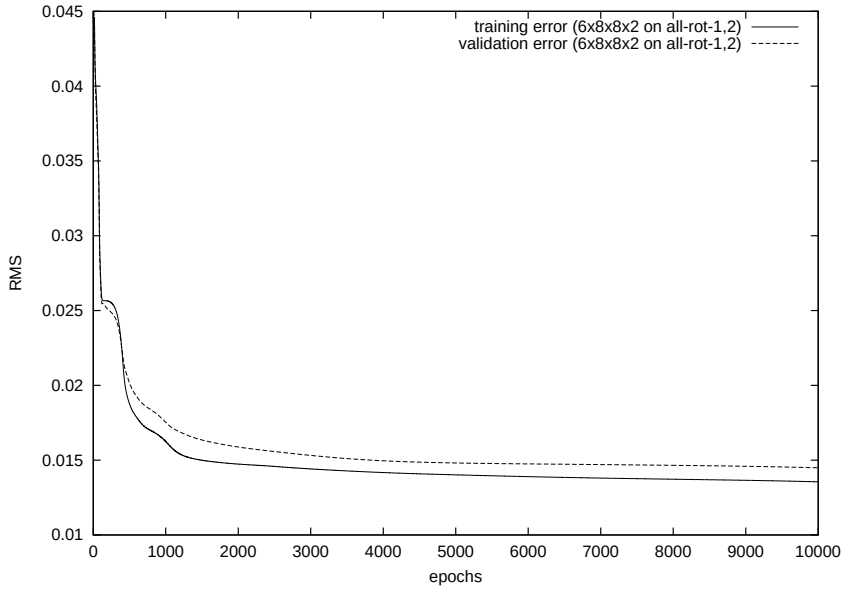


Positional Error (with IK)

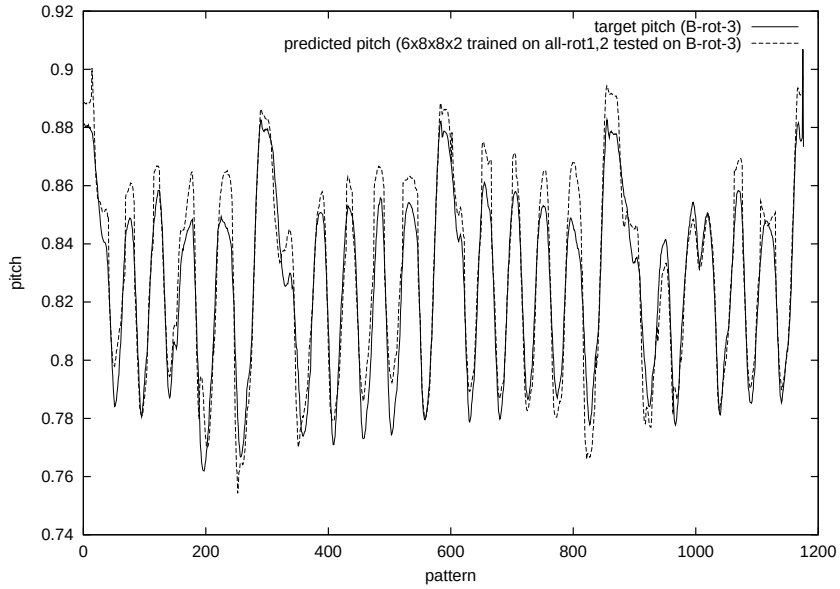


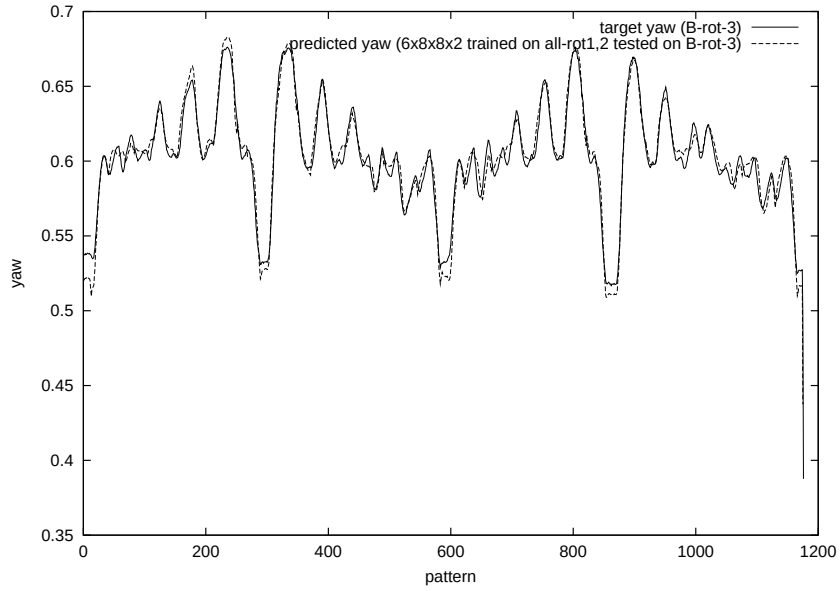
User-Independent 6x8x8x2 Performance (Trained on All-rot-1,2. Tested on B-rot-3.)

Training and Validation Error

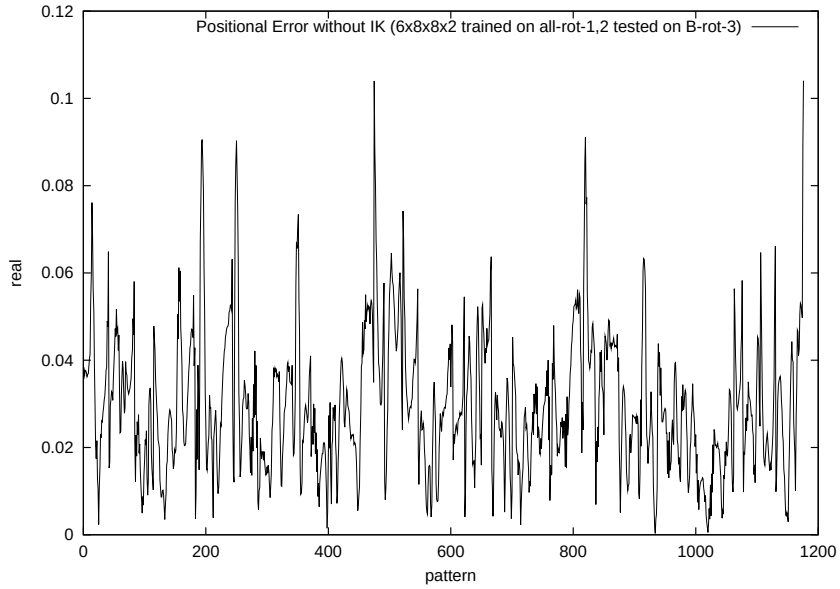


Pitch and Yaw Prediction

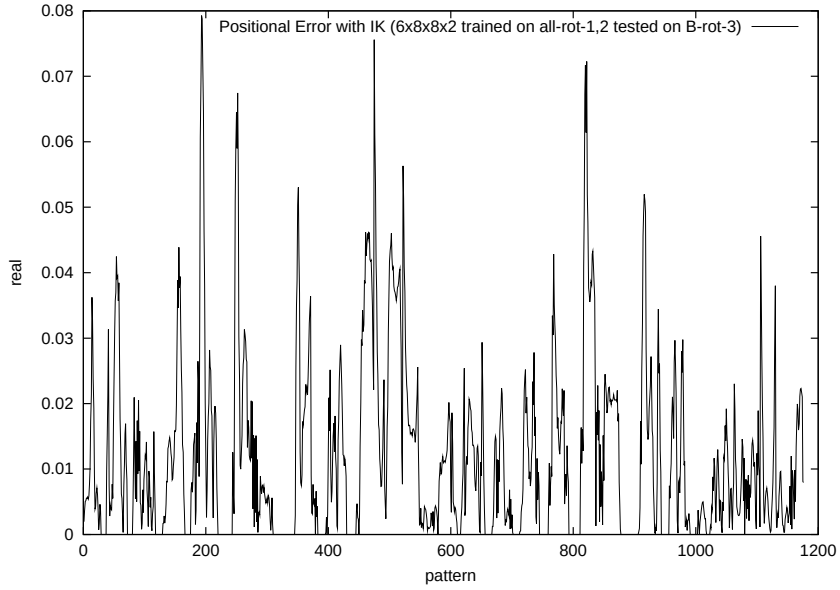




Positional Error (without IK)



Positional Error (with IK)



User-Specific/Independent 6x16x16x2 Results

Trained on participant	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
A	A-rot-1,2	A-rot-3	0.040173	0.034579
	"	B-rot-3	0.083277	0.041219
	"	C-rot-3	0.073310	0.046916
	"	D-rot-3	0.061362	0.042859
	"	E-rot-3	0.127312	0.095770
	"	F-rot-3	0.063113	0.028955
B	B-rot-1,2	A-rot-3	0.123143	0.106932
	"	B-rot-3	0.016790	0.008909
	"	C-rot-3	0.103995	0.086467
	"	D-rot-3	0.085726	0.059995
	"	E-rot-3	0.111052	0.075793
	"	F-rot-3	0.071386	0.043477
C	C-rot-1,2	A-rot-3	0.072485	0.056337
	"	B-rot-3	0.082735	0.034976
	"	C-rot-3	0.021138	0.016561
	"	D-rot-3	0.059686	0.033041
	"	E-rot-3	0.117124	0.080691
	"	F-rot-3	0.079918	0.035712
D	D-rot-1,2	A-rot-3	0.082799	0.067170
	"	B-rot-3	0.136402	0.077772
	"	C-rot-3	0.077418	0.037493
	"	D-rot-3	0.035564	0.022266
	"	E-rot-3	0.136536	0.091058
	"	F-rot-3	0.057128	0.022778

Trained on participant	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
E	E-rot-1,2	A-rot-3	0.204102	0.168075
	"	B-rot-3	0.107889	0.056517
	"	C-rot-3	0.169742	0.144862
	"	D-rot-3	0.156036	0.122648
	"	E-rot-3	0.027790	0.017991
	"	F-rot-3	0.156397	0.088604
F	F-rot-1,2	A-rot-3	0.076653	0.031865
	"	B-rot-3	0.087992	0.040966
	"	C-rot-3	0.110658	0.035539
	"	D-rot-3	0.070529	0.043833
	"	E-rot-3	0.149322	0.103429
	"	F-rot-3	0.017592	0.008003
All	All-rot-1,2	A-rot-3	0.079740	0.069071
	"	B-rot-3	0.031210	0.012496
	"	C-rot-3	0.040004	0.031303
	"	D-rot-3	0.039618	0.026007
	"	E-rot-3	0.078588	0.056827
	"	F-rot-3	0.037277	0.013988

User-Specific/Independent 6x16x16x2 Summary

Trained on participant	Positional error on same participant (meters)		Mean positional error on other participants (meters)	
	Without IK	With IK	Without IK	With IK
A	0.040173	0.034579	0.081675	0.051144
B	0.016790	0.008909	0.099060	0.074533
C	0.021138	0.016561	0.082390	0.048151
D	0.035564	0.022266	0.098057	0.059254
E	0.027790	0.017991	0.158833	0.116141
F	0.017592	0.008003	0.099031	0.051126
All	—	—	0.061287	0.034949

User-Specific/Independent 6x24x24x2 Results

Trained on participant	Training (and validation) Data	Testing data	Positional error (meters)	
			Without IK	With IK
A	A-rot-1,2	A-rot-3	0.038565	0.033348
	"	B-rot-3	0.090902	0.049774
	"	C-rot-3	0.063754	0.046366
	"	D-rot-3	0.072555	0.049528
	"	E-rot-3	0.121019	0.095486
	"	F-rot-3	0.076505	0.031269
B	B-rot-1,2	A-rot-3	0.118075	0.100803
	"	B-rot-3	0.016331	0.008619
	"	C-rot-3	0.108280	0.080242
	"	D-rot-3	0.086948	0.059533
	"	E-rot-3	0.116111	0.078847
	"	F-rot-3	0.067038	0.045172
C	C-rot-1,2	A-rot-3	0.063135	0.032395
	"	B-rot-3	0.098925	0.058410
	"	C-rot-3	0.019484	0.016268
	"	D-rot-3	0.062870	0.035313
	"	E-rot-3	0.131571	0.098696
	"	F-rot-3	0.075218	0.023476
D	D-rot-1,2	A-rot-3	0.065771	0.046207
	"	B-rot-3	0.104772	0.049985
	"	C-rot-3	0.076670	0.042409
	"	D-rot-3	0.038547	0.021536
	"	E-rot-3	0.130729	0.087677
	"	F-rot-3	0.061155	0.021946

Trained on participant	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
E	E-rot-1,2	A-rot-3	0.200366	0.163905
	"	B-rot-3	0.113674	0.062262
	"	C-rot-3	0.187697	0.146950
	"	D-rot-3	0.155120	0.123400
	"	E-rot-3	0.026750	0.018357
	"	F-rot-3	0.159763	0.089825
F	F-rot-1,2	A-rot-3	0.096382	0.036872
	"	B-rot-3	0.094554	0.043437
	"	C-rot-3	0.106438	0.028198
	"	D-rot-3	0.070543	0.044002
	"	E-rot-3	0.151082	0.104660
	"	F-rot-3	0.016391	0.008058
All	All-rot-1,2	A-rot-3	0.088615	0.078928
	"	B-rot-3	0.027242	0.010946
	"	C-rot-3	0.036957	0.028644
	"	D-rot-3	0.038611	0.025706
	"	E-rot-3	0.072780	0.049824
	"	F-rot-3	0.038455	0.015918

User Specific/Independent 6x24x24x2 Summary

Trained on participant	Positional error on same participant (meters)		Mean positional error on other participants (meters)	
	Without IK	With IK	Without IK	With IK
A	0.038565	0.033348	0.084947	0.054485
B	0.016331	0.008619	0.099290	0.072919
C	0.019484	0.016268	0.086344	0.049658
D	0.038547	0.021536	0.087819	0.049645
E	0.026750	0.018357	0.163324	0.117268
F	0.016391	0.008058	0.103800	0.051434
All	—	—	0.050443	0.034994

C.2.2 Task-Specific vs Task-Independent

Task-Specific/Independent 6x8x8x2 Results

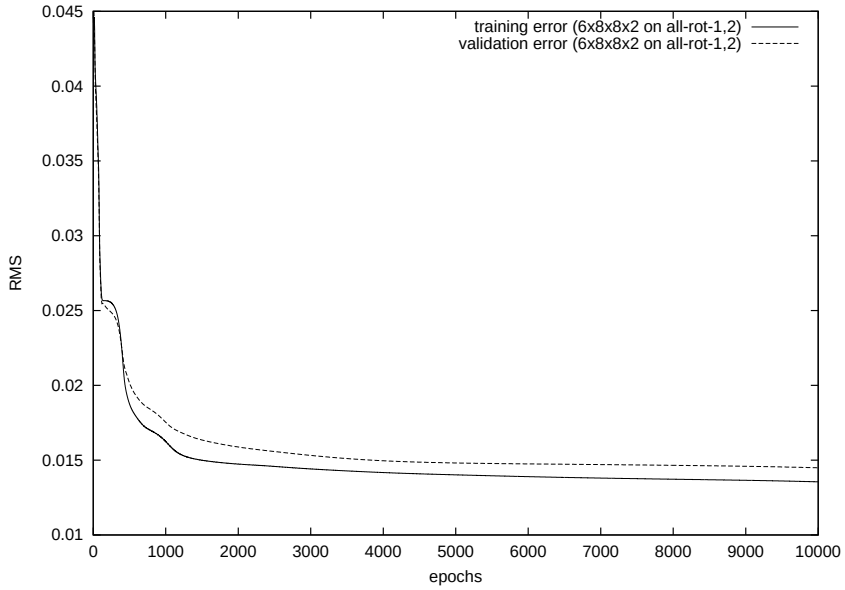
Trained on task	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
Random	All-rand-1,2	A-rand-3	0.073481	0.053016
	"	B-rand-3	0.059895	0.025831
	"	C-rand-3	0.053332	0.026738
	"	D-rand-3	0.090498	0.046550
	"	E-rand-3	0.074750	0.053525
	"	F-rand-3	0.070369	0.028459
	"	A-rot-3	0.098374	0.073343
	"	B-rot-3	0.087209	0.053920
	"	C-rot-3	0.085394	0.063139
	"	D-rot-3	0.089019	0.066801
	"	E-rot-3	0.156425	0.128187
	"	F-rot-3	0.068344	0.026313
Rotation	All-rot-1,2	A-rand-3	0.144560	0.085220
	"	B-rand-3	0.121745	0.047435
	"	C-rand-3	0.133073	0.065574
	"	D-rand-3	0.145361	0.084855
	"	E-rand-3	0.141204	0.059909
	"	F-rand-3	0.132472	0.054040
	"	A-rot-3	0.077139	0.066033
	"	B-rot-3	0.029742	0.011905
	"	C-rot-3	0.042105	0.032734
	"	D-rot-3	0.043317	0.030064
	"	E-rot-3	0.076091	0.053525
	"	F-rot-3	0.041604	0.016650

Trained on task	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
Both	All-both-1,2	A-rand-3	0.066444	0.046048
	"	B-rand-3	0.070885	0.031548
	"	C-rand-3	0.069559	0.037013
	"	D-rand-3	0.094528	0.053266
	"	E-rand-3	0.075415	0.051687
	"	F-rand-3	0.073018	0.029724
	"	A-rot-3	0.062754	0.050928
	"	B-rot-3	0.033698	0.012110
	"	C-rot-3	0.041985	0.032331
	"	D-rot-3	0.048637	0.030512
	"	E-rot-3	0.083637	0.056151
	"	F-rot-3	0.052108	0.026199

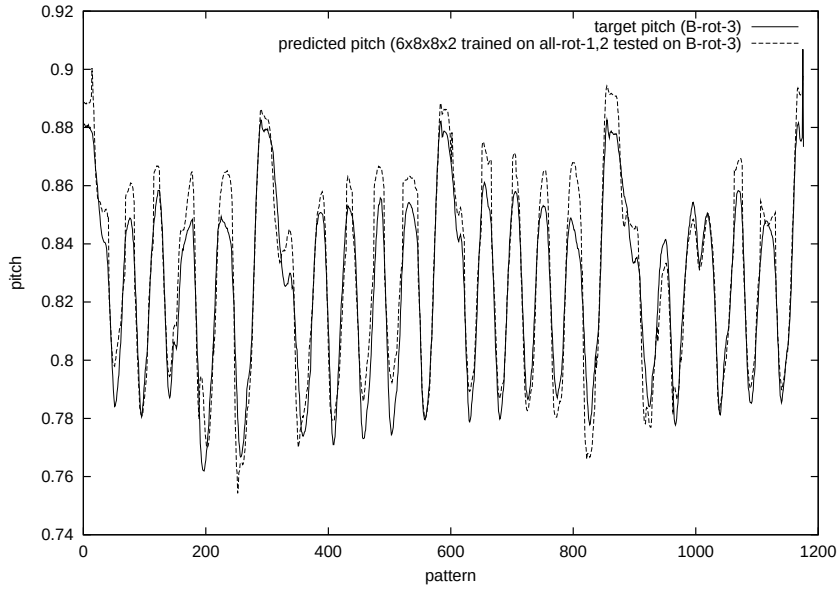
Trained on task	Mean positional error on random task (meters)		Mean positional error on rotation task (meters)	
	Without IK	With IK	Without IK	With IK
Random	0.070388	0.039020	0.097461	0.068617
Rotation	0.136403	0.066172	0.051666	0.035152
Both	0.074975	0.041548	0.053803	0.034755

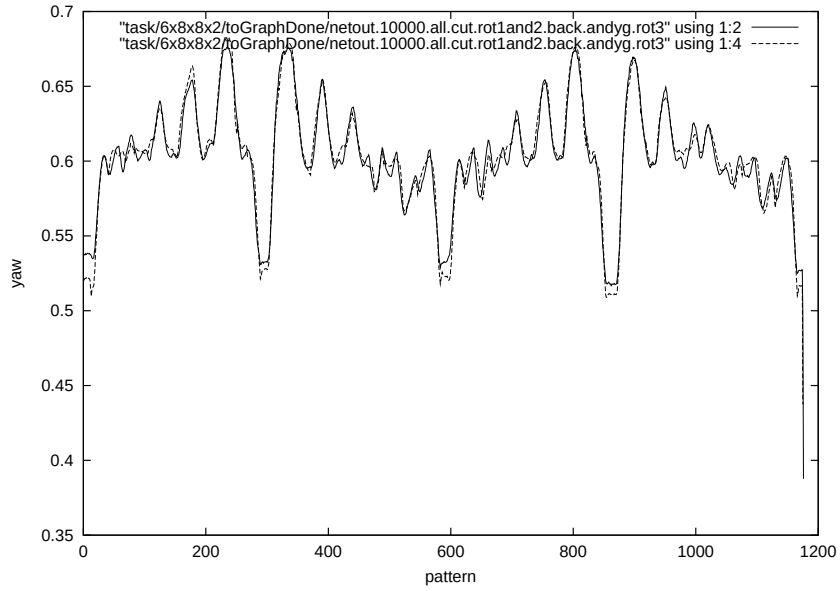
Task-Specific 6x8x8x2 Performance (Trained on All-rot-1,2. Tested on B-rot-3.)

Training and Validation Error

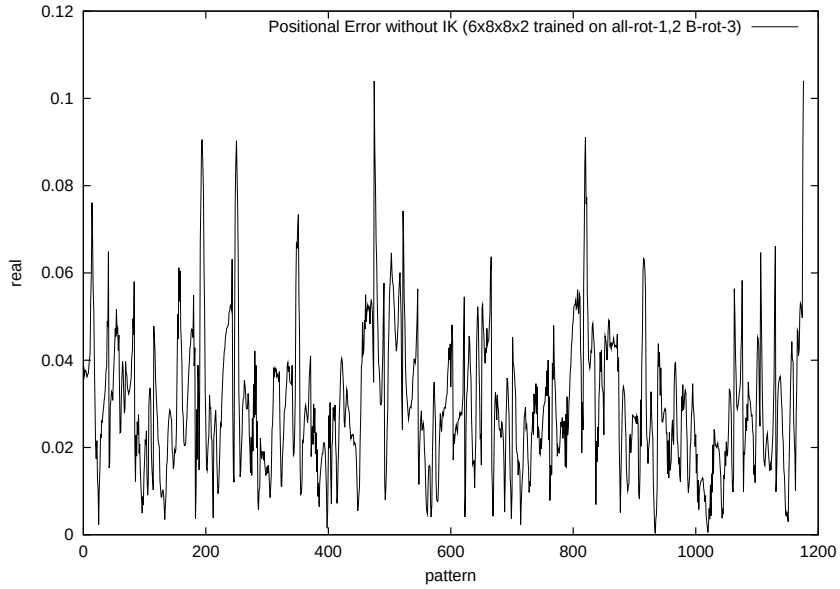


Pitch and Yaw Prediction

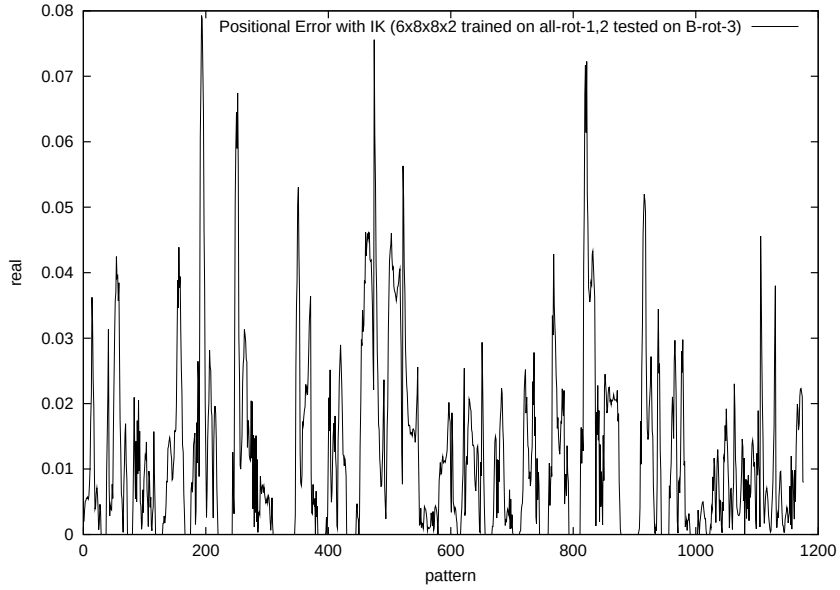




Positional Error (without IK)

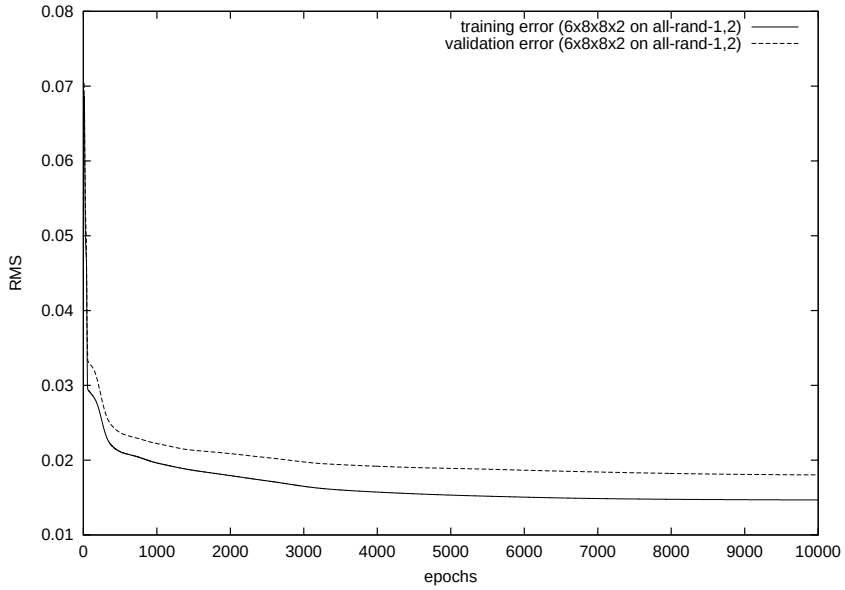


Positional Error (with IK)

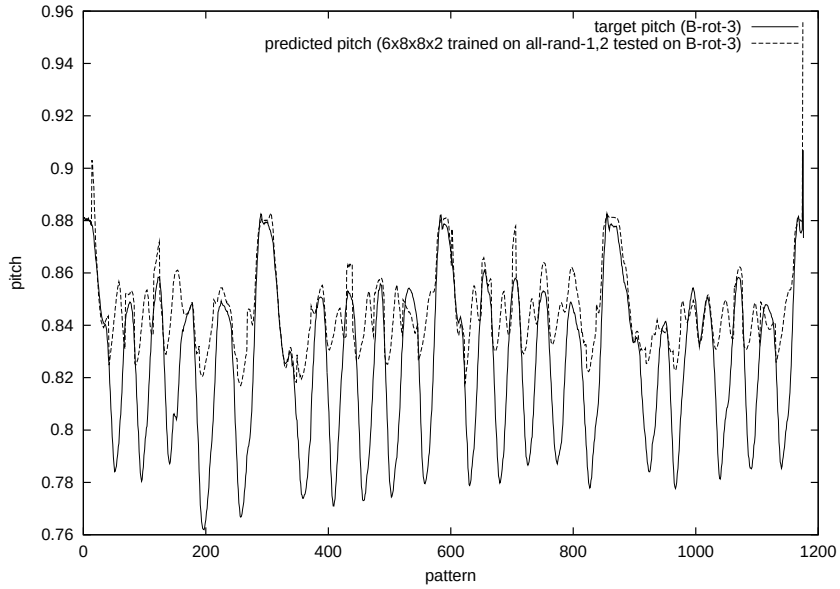


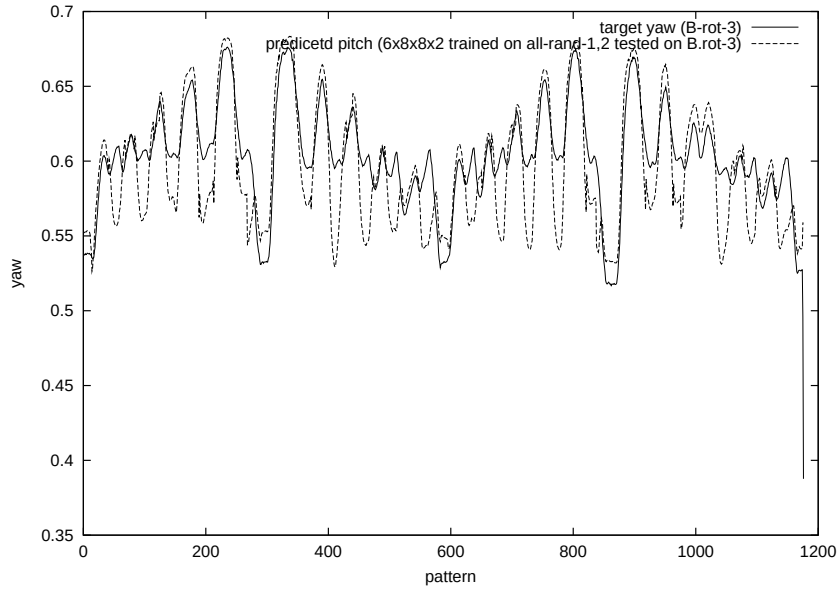
Task-Independent 6x8x8x2 Performance (Trained on All-rand-1,2. Tested on B-rot-3.)

Training and Validation Error

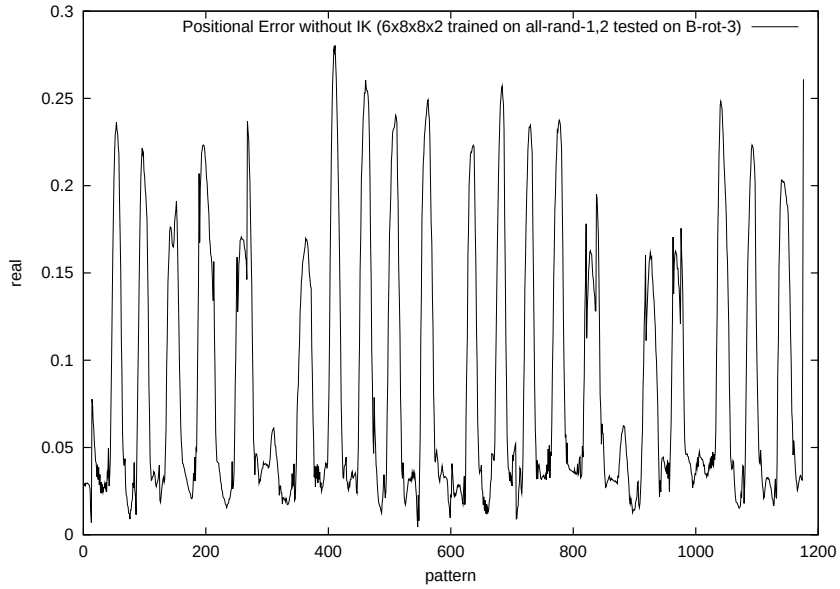


Pitch and Yaw Prediction

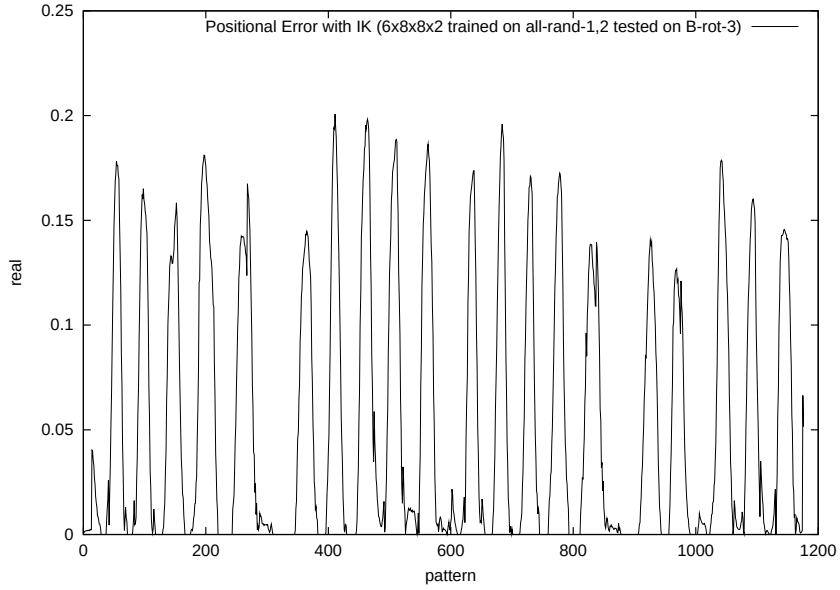




Positional Error (without IK)



Positional Error (with IK)



Task-Specific/Independent 6x16x16x2 Results

Trained on task	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
Random	All-rand-1,2	A-rand-3	0.075143	0.055433
	"	B-rand-3	0.050327	0.024599
	"	C-rand-3	0.049021	0.025084
	"	D-rand-3	0.093072	0.050899
	"	E-rand-3	0.070445	0.051606
	"	F-rand-3	0.072144	0.028861
	"	A-rot-3	0.100249	0.084700
	"	B-rot-3	0.070851	0.048705
	"	C-rot-3	0.101835	0.069681
	"	D-rot-3	0.095847	0.070222
	"	E-rot-3	0.157215	0.127296
	"	F-rot-3	0.069154	0.023221
Rotation	All-rot-1,2	A-rand-3	0.147951	0.092124
	"	B-rand-3	0.122075	0.043515
	"	C-rand-3	0.111610	0.052208
	"	D-rand-3	0.137553	0.090457
	"	E-rand-3	0.137541	0.058960
	"	F-rand-3	0.113446	0.050330
	"	A-rot-3	0.079740	0.069071
	"	B-rot-3	0.031210	0.012496
	"	C-rot-3	0.040004	0.031303
	"	D-rot-3	0.039618	0.026007
	"	E-rot-3	0.078588	0.056827
	"	F-rot-3	0.037277	0.013988

Trained on task	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
Both	All-both-1,2	A-rand-3	0.063185	0.045241
	"	B-rand-3	0.064296	0.028284
	"	C-rand-3	0.066479	0.037333
	"	D-rand-3	0.088564	0.046422
	"	E-rand-3	0.078624	0.053906
	"	F-rand-3	0.069313	0.026646
	"	A-rot-3	0.086236	0.077202
	"	B-rot-3	0.027752	0.011282
	"	C-rot-3	0.044815	0.037205
	"	D-rot-3	0.041584	0.025509
	"	E-rot-3	0.079584	0.054929
	"	F-rot-3	0.044385	0.016022

Trained on task	Mean positional error on random task (meters)		Mean positional error on rotation task (meters)	
	Without IK	With IK	Without IK	With IK
Random	0.063359	0.039414	0.099192	0.070638
Rotation	0.128363	0.064599	0.051073	0.034949
Both	0.071743	0.039639	0.054059	0.037025

Task-Specific/Independent 6x24x24x2 Results

Trained on task	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
Random	All-rand-1,2	A-rand-3	0.069459	0.050463
	"	B-rand-3	0.052375	0.022782
	"	C-rand-3	0.045609	0.021571
	"	D-rand-3	0.091267	0.047511
	"	E-rand-3	0.072442	0.051539
	"	F-rand-3	0.067791	0.025609
	"	A-rot-3	0.114680	0.086604
	"	B-rot-3	0.089244	0.058217
	"	C-rot-3	0.096432	0.057730
	"	D-rot-3	0.091298	0.062340
	"	E-rot-3	0.167966	0.130771
	"	F-rot-3	0.083933	0.027583
Rotation	All-rot-1,2	A-rand-3	0.193513	0.131815
	"	B-rand-3	0.106075	0.039413
	"	C-rand-3	0.161860	0.078213
	"	D-rand-3	0.179457	0.124242
	"	E-rand-3	0.126074	0.096726
	"	F-rand-3	0.180229	0.082491
	"	A-rot-3	0.088615	0.078928
	"	B-rot-3	0.027242	0.010946
	"	C-rot-3	0.036957	0.028644
	"	D-rot-3	0.038611	0.025706
	"	E-rot-3	0.072780	0.049824
	"	F-rot-3	0.038455	0.015918

Trained on task	Training (and validation) data	Testing data	Positional error (meters)	
			Without IK	With IK
Both	All-both-1,2	A-rand-3	0.064891	0.047345
	"	B-rand-3	0.067553	0.030882
	"	C-rand-3	0.066299	0.035790
	"	D-rand-3	0.094576	0.051606
	"	E-rand-3	0.071934	0.049081
	"	F-rand-3	0.070988	0.024868
	"	A-rot-3	0.075835	0.064613
	"	B-rot-3	0.029788	0.011457
	"	C-rot-3	0.040196	0.032859
	"	D-rot-3	0.043116	0.026606
	"	E-rot-3	0.072734	0.050486
	"	F-rot-3	0.044904	0.016155

Trained on task	Mean positional error on random task (meters)		Mean positional error on rotation task (meters)	
	Without IK	With IK	Without IK	With IK
Random	0.066491	0.0365792	0.107259	0.070541
Rotation	0.157868	0.092150	0.504433	0.034994
Both	0.072707	0.039929	0.051104	0.033696

C.3 Experiment 3

There are 3 scenarios:

1. user-specific (existing user)
    (a) task-specific
    (b) task-independent
2. user-independent (existing user)
    (a) task-specific
    (b) task-independent
3. user-independent (new user)
    (a) task-specific
    (b) task-independent

C.3.1 Scenario 1

1a: User-Specific, Task-Specific (existing user)

Scenario	Training data	Same session			Different session		
		Testing data	Positional error (meters)		Testing data	Positional error (meters)	
			Without IK	With IK		Without IK	With IK
1a	A-rot-1,2	A-rot-3	0.040119	0.034537	A-rot-4	0.039115	0.022109
	"	—	—	—	A-highrot-1	0.062230	0.050472
	B-rot-1,2	B-rot-3	0.018160	0.008828	B-rot-4	0.046256	0.017068
	"	—	—	—	B-highrot-1	0.060389	0.020558
	C-rot-1,2	C-rot-3	0.020500	0.014978	C-rot-4	0.029144	0.023282
	"	—	—	—	C-highrot-1	0.081761	0.059929
	D-rot-1,2	D-rot-3	0.032585	0.020558	D-rot-4	0.063186	0.049094
	"	—	—	—	D-highrot-1	0.064406	0.033832
	E-rot-1,2	E-rot-3	0.029210	0.019062	E-rot-4	0.134668	0.116665
	"	—	—	—	E-highrot-1	0.126602	0.107709
	F-rot-1,2	F-rot-3	0.019058	0.008482	F-rot-4	0.065751	0.035651
		—	—	—	F-highrot-1	0.089287	0.037866

1b: User-Specific, Task-Independent (existing user)

Scenario	Training data	Same session			Different session		
		Testing data	Positional error (meters)		Testing data	Positional error (meters)	
			Without IK	With IK		Without IK	With IK
1b	A-rand-1,2	A-rot-3	0.125922	0.118206	A-rot-4	0.176038	0.157846
	"	—	—	—	A-highrot-1	0.189838	0.169818
	B-rand-1,2	B-rot-3	0.065246	0.030730	B-rot-4	0.104031	0.059136
	"	—	—	—	B-highrot-1	0.104335	0.058651
	C-rand-1,2	C-rot-3	0.092435	0.067971	C-rot-4	0.160492	0.113941
	"	—	—	—	C-highrot-1	0.189634	0.101640
	D-rand-1,2	D-rot-3	0.091527	0.062793	D-rot-4	0.081204	0.042448
	"	—	—	—	D-highrot-1	0.095298	0.040094
	E-rand-1,2	E-rot-3	0.146762	0.119170	E-rot-4	0.085322	0.055533
	"	—	—	—	E-highrot-1	0.078098	0.052258
	F-rand-1,2	F-rot-3	0.040720	0.009740	F-rot-4	0.052926	0.041612
		—	—	—	F-highrot-1	0.075174	0.044210

C.3.2 Scenario 2

2a: User-Independent, Task-Specific (existing user)

Scenario	Training data	Same session			Different session		
		Testing data	Positional error (meters)		Testing data	Positional error (meters)	
			Without IK	With IK		Without IK	With IK
2a	All-rot-1,2	A-rot-3	0.077139	0.066033	A-rot-4	0.072219	0.056237
	"	—	—	—	A-highrot-1	0.043783	0.028989
	"	B-rot-3	0.029742	0.011905	B-rot-4	0.048714	0.023349
	"	—	—	—	B-highrot-1	0.059460	0.025365
	"	C-rot-3	0.042105	0.032734	C-rot-4	0.045707	0.037639
	"	—	—	—	C-highrot-1	0.114606	0.111875
	"	D-rot-3	0.043317	0.030064	D-rot-4	0.068261	0.050608
	"	—	—	—	D-highrot-1	0.098846	0.087739
	"	E-rot-3	0.076091	0.053525	E-rot-4	0.053819	0.048960
	"	—	—	—	E-highrot-1	0.048500	0.037620
	"	F-rot-3	0.041604	0.016650	F-rot-4	0.053531	0.038810
	"	—	—	—	F-highrot-1	0.085290	0.067283

2b: User-Independent, Task-Independent (existing user)

Scenario	Training data	Same session			Different session		
		Testing data	Positional error (meters)		Testing data	Positional error (meters)	
			Without IK	With IK		Without IK	With IK
2b	All-rand-1,2	A-rot-3	0.098374	0.073343	A-rot-4	0.129546	0.105667
	"	—	—	—	A-highrot-1	0.130520	0.121578
	"	B-rot-3	0.087209	0.053920	B-rot-4	0.107295	0.068951
	"	—	—	—	B-highrot-1	0.106581	0.068811
	"	C-rot-3	0.085394	0.063139	C-rot-4	0.123239	0.099243
	"	—	—	—	C-highrot-1	0.117097	0.077028
	"	D-rot-3	0.089019	0.066801	D-rot-4	0.066801	0.074448
	"	—	—	—	D-highrot-1	0.095779	0.063256
	"	E-rot-3	0.156425	0.128187	E-rot-4	0.072617	0.053218
	"	—	—	—	E-highrot-1	0.090840	0.057472
	"	F-rot-3	0.068344	0.026313	F-rot-4	0.091423	0.057071
	"	—	—	—	F-highrot-1	0.078663	0.036272

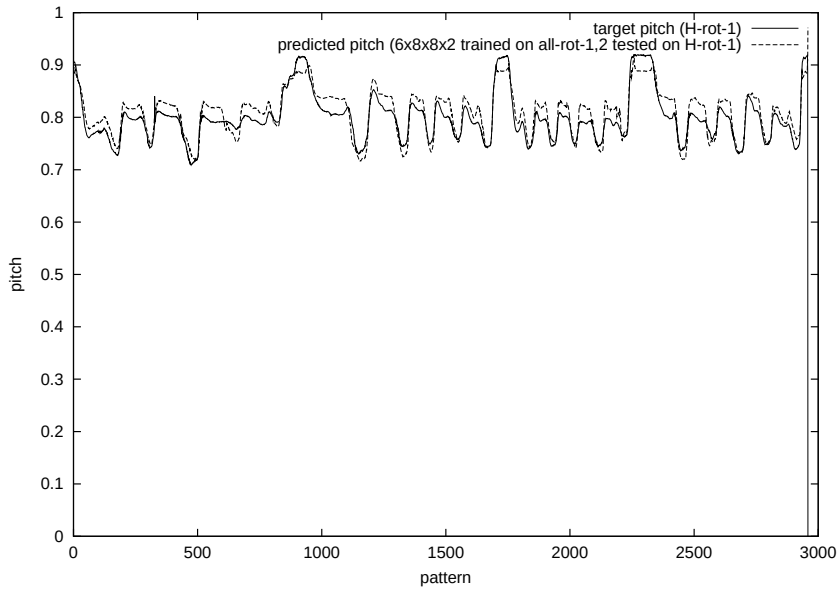
C.3.3 Scenario 3

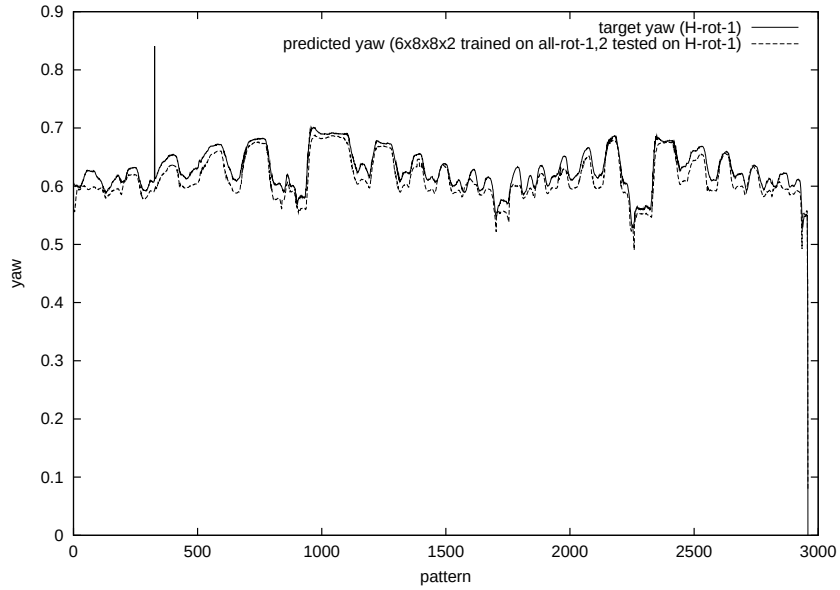
3a: User-Independent, Task-Specific (new user)

Scenario	Training data	Same session			Different session		
		Testing data	Positional error (meters)		Testing data	Positional error (meters)	
			Without IK	With IK		Without IK	With IK
3a	All-rot-1,2	—	—	—	G-rot-1	0.078528	0.051229
	"	—	—	—	G-highrot-1	0.064281	0.034091
	"	—	—	—	H-rot-1	0.062852	0.027418
	"	—	—	—	H-highrot-1	0.100837	0.054026
	"	—	—	—	I-rot-1	0.067401	0.051699
	"	—	—	—	I-highrot-1	0.085606	0.074262
	"	—	—	—	J-rot-1	0.076965	0.068759
	"	—	—	—	J-highrot-1	0.077418	0.068556
	"	—	—	—	K-rot-1	0.132448	0.110777
	"	—	—	—	K-highrot-1	0.119939	0.113859
	"	—	—	—	L-rot-1	0.078319	0.044119
	"	—	—	—	L-highrot-1	0.095375	0.047329
	"	—	—	—	M-rot-1	0.056771	0.033929
	"	—	—	—	M-highrot-1	0.080057	0.055770
	"	—	—	—	N-rot-1	0.076067	0.052592
	"	—	—	—	N-highrot-1	0.080709	0.058065

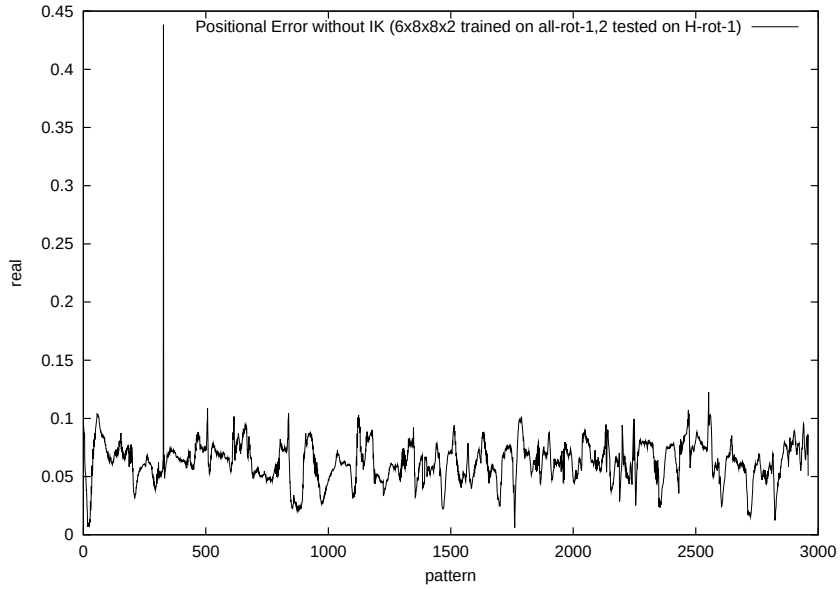
3a Performance (Trained on All-rot-1,2. Tested on H-rot-1.)

Pitch and Yaw Prediction

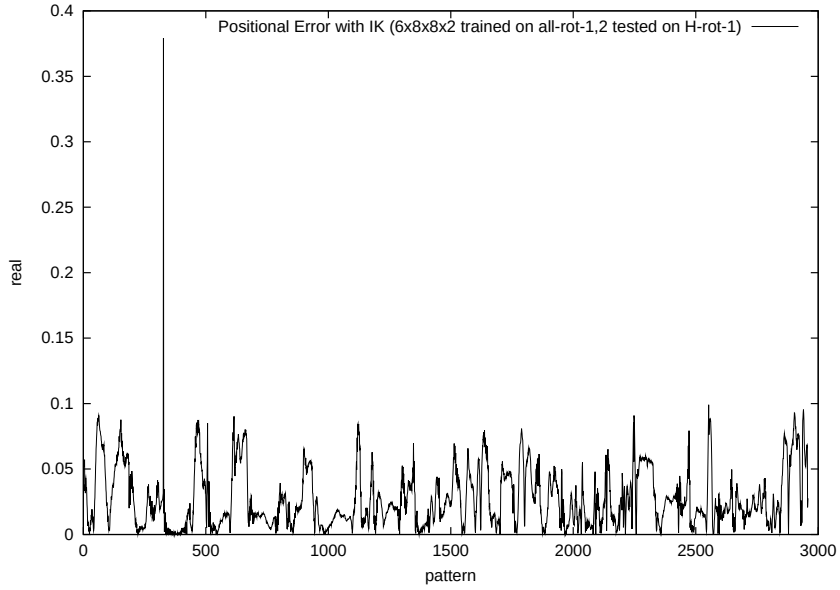




Positional Error (without IK)



Positional Error (with IK)

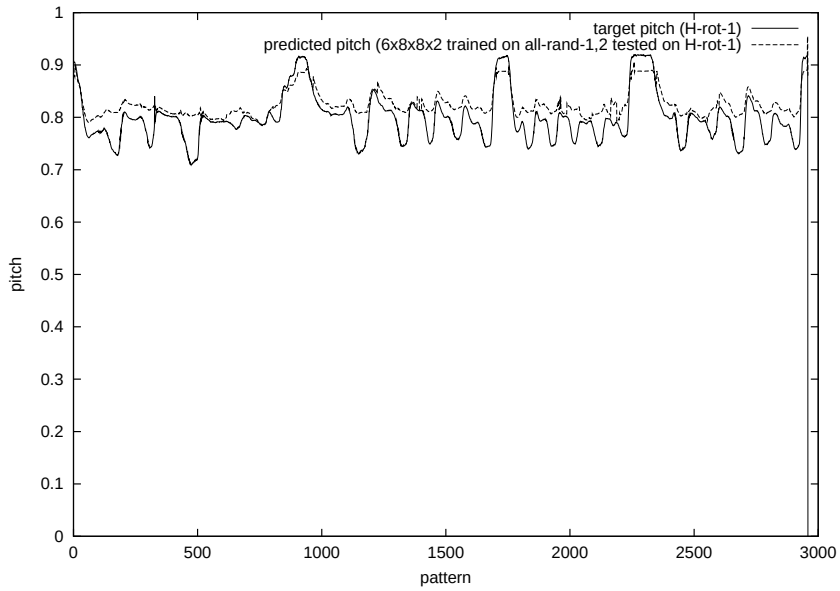


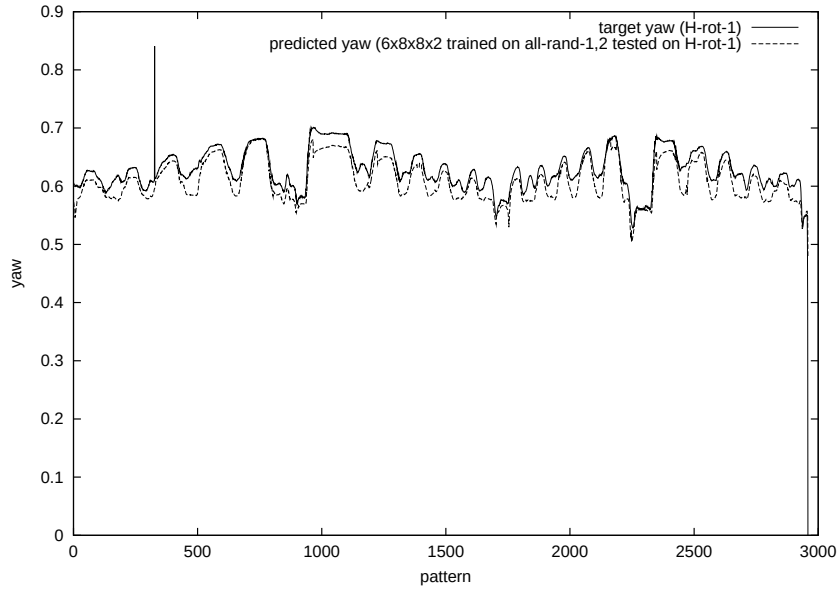
3a: User-Independent, Task-Independent (new user)

Scenario	Training data	Same session			Different session		
		Testing data	Positional error (meters)		Testing data	Positional error (meters)	
			Without IK	With IK		Without IK	With IK
3b	All-rand-1,2	—	—	—	G-rot-1	0.095449	0.072806
	"	—	—	—	G-highrot-1	0.119594	0.098030
	"	—	—	—	H-rot-1	0.092144	0.059724
	"	—	—	—	H-highrot-1	0.110812	0.070257
	"	—	—	—	I-rot-1	0.104535	0.085991
	"	—	—	—	I-highrot-1	0.088878	0.055725
	"	—	—	—	J-rot-1	0.102838	0.092251
	"	—	—	—	J-highrot-1	0.125206	0.094081
	"	—	—	—	K-rot-1	0.078813	0.055260
	"	—	—	—	K-highrot-1	0.099897	0.073900
	"	—	—	—	L-rot-1	0.111385	0.074725
	"	—	—	—	L-highrot-1	0.104252	0.064595
	"	—	—	—	M-rot-1	0.124584	0.095775
	"	—	—	—	M-highrot-1	0.157361	0.121750
	"	—	—	—	N-rot-1	0.087160	0.065902
	"	—	—	—	N-highrot-1	0.107097	0.081795

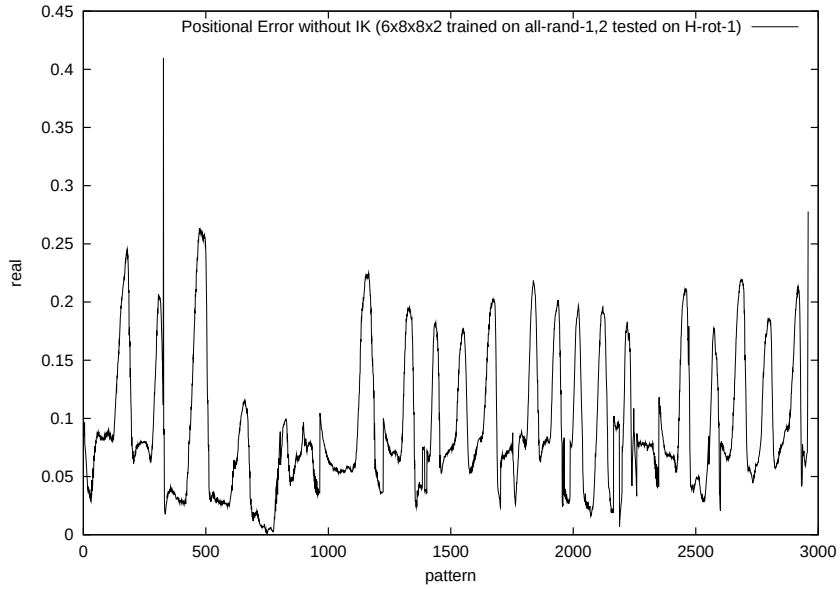
3b Performance (Trained on All-rand-1,2. Tested on H-rot-1.)

Pitch and Yaw Prediction

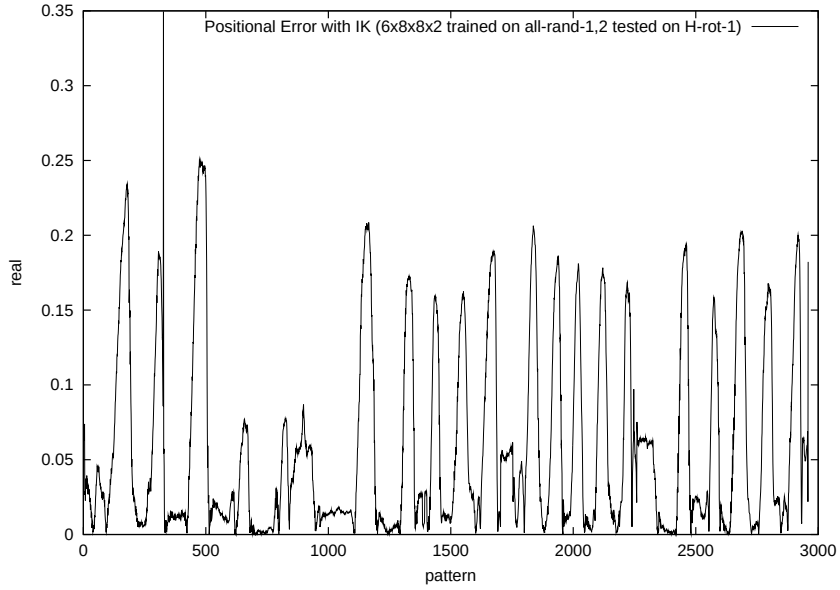




Positional Error (without IK)



Positional Error (with IK)



C.3.4 Experiment 3 Summary

Scenario	Same session		Different session			
	Mean positional error on rotation task (meters)		Mean positional error on rotation task (meters)		Mean positional error on high-rotation task (meters)	
	Without IK	With IK	Without IK	With IK	Without IK	With IK
1a	0.026772	0.017741	0.063020	0.043978	0.080779	0.051728
1b	0.093769	0.068102	0.110002	0.078419	0.122063	0.077779
2a	0.051666	0.036152	0.057042	0.042601	0.075081	0.059812
2b	0.097461	0.068617	0.098487	0.076433	0.103247	0.070736
3a	—	—	0.078669	0.055065	0.088028	0.063245
3b	—	—	0.099614	0.075304	0.114137	0.082517

Bibliography

[1] H. Amin and R. A. Earnshaw. Enhanced avatar control using neural networks. *Virtual Reality*, (5):47–53, 2000.

[2] N. Badler. Real-time virtual humans. In *Pacific Graphics*, pages 4–14, 1997.

[3] Ashweeni Beeharee. Modelling arm movements in a virtual environment using neural networks. Master's thesis, Department of Computer Science, School of Science and Engineering, University of Manchester, 1999.

[4] Ashweeni Beeharee and Roger Hubbold. Real-time avatar control with a neural network. Advanced Interfaces Group, Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK. http://aig.cs.man.ac.uk, 1999.

[5] Alan Beverage. Modelling arm movements in a virtual environment using neural networks. Master's thesis, Department of Computer Science, School of Science and Engineering, University of Manchester, 1998.

[6] Bobby Bodenheimer, Charles F. Rose, Seth Rosenthal, and John Pella. The process of motion capture: Dealing with the data. *Computer Animation and Simulation '97*, pages 3–18, 1997.

[7] R. Boulic, P. Becheiraz, L. Emering, and D. Thalmann. Integration of motion control techniques for virtual human and avatar Real-Time animation. In *ACM Symposium on Virtual Reality Software and Technology*, New York, NY, 1997. ACM Press.

[8] R. Boulic and D. Thalmann. Combined direct and inverse kinematic control for articulated figure motion editing. *Computer Graphics Forum*, 11(4):189–202, 1992.

[9] Ronan Boulic, Nadia Magnenat Thalmann, and Daniel Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6(6):344–358, 1990.

[10] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, (14):179–211, 1990.

[11] R. Grzeszczuk. *NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models.* PhD thesis, Dept. of Computer Science, University of Toronto, May 1998.

[12] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Computer Graphics, Annual Conference Series Proc. SIGGRAPH '98*, pages 9–20, july 1998.

[13] Simon Haykin. *Neural Networks: A comprehensive foundation.* Prentice Hall, second edition, 1999.

[14] Roger Hubbold, Jon Cook, Martin Keates, Simon Gibson, Toby Howard, and Alan Murta. Gnu/maverik: A micro-kernel for large scale virtual environments. In *Symposium on Virtual Reality Software and Technology*, December 1999.

[15] Roger Hubbold, Martin Keates, Simon Gibson, Alan Murta, Steve Pettifer, and Adrian West. Maverik programmer's guide, 2001. Advanced Interfaces Group, Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK. http://aig.cs.man.ac.uk.

[16] Roger J. Hubbold and M. J. Keates. Real-time simulation of a stretcher evacuation in a large-scale virtual environment. *Computer Graphics Forum*, 19(2):123–134, 2000.

[17] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, (3):79–87, 1991.

[18] Caroline Jay. Exagerated head movements in a head mounted display. Master's thesis, Department of Computer Science, School of Science and Engineering, University of Manchester, 2001.

[19] Shih kai Chung and James K. Hahn. Animation of human walking in virtual environments. In *Computer Animation*, pages 4–15, 1999.

[20] ChanSu Lee, SangWon Ghyme, ChanJong Park, and KwangYun Wohn. The control of avatar motion using hand gesture. *Virtual Reality Software and Technology '98*, pages 59–66, November 1998. Taipei, Taiwan.

[21] J.L McClelland, B.L. McNaughton, and R.C. O'Riley. Why there are complimentary learning systems in the hippocampus and neocortex: Insights from the success and failures of connectionist models of learning and memory. *Psychology Review*, 102(3):419–457, 1995.

[22] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[23] Jean-Christophe Nebel. Keyframe animation of articulated figures using autocollision-free interpolation. In *Eurographics UK*, pages 13–15, Cambridge, UK, April 1999.

[24] Jean-Christophe Nebel. Realistic collision avoidance of upper limbs based on neuroscience models. *Computer Graphics Forum*, 19(3), August 2000.

[25] Luciana Porcher Nedel, Tom Molet, and Daniel Thalmann. Animation of virtual human bodies using motion capture devices. In *Brazilian Workshop on Virtual Reality WRV'99*, November 1999. Marília, SP.

[26] U. Nehmzow and T.Smithers. Mapbuilding using self-organising networks in really useful robots. *Simulation of adaptive behaviour*, 1991.

[27] S. Razzaque, S. Kohn, and M. Whitton. Redirected walking. Technical Report TR01-007, University of North Carolina, 2001.

[28] Jeff Shufelt. Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3891.

[29] F.J. Smieja. Multiple network systems (minos) modules: Task division and module discrimination. In *8th AISB conference on Artificial Intelligence*, 16-19 April 1991.

[30] D. Thalmann. Physical, behavioral,and sensor-based animation. In *Graphicon '96*, pages 214–221, 1996.

[31] B. Webber and N. Badler. Animation through reactions, transition nets and plans. In *International Workshop on Human Interface Technology*, October 1995.

[32] Douglas J. Wiley and James K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, November/December 1997.

[33] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, 1994.

[34] Victor B. Zordan and Jessica K. Hodgins. Tracking and modifying upper-body human motion data with dynamic simulation. In *Eurographics '99: Computer Animation and Simulation Workshop*, pages 13–22, 1999.