University of Manchester
School of Computer Science
Project Report April 2013
BSc Computer Science with Industrial Experience

Gesture Recognition using Microsoft's Kinect

Author: Laura Howarth-Kirke

Supervisor: Dr. Gavin Brown

Abstract

Gesture Recognition using Microsoft's Kinect

Author: Laura Howarth-Kirke

Gesture recognition is becoming increasing popular since it's reappearance in the gaming industry recently. The main influence has been the depth camera in Microsoft's Kinect which is now being utilised in unconventional industries such as to provide support during surgical operations. This means that gesture recognition needs to be as accurate as possible to avoid delays in what might be critical situations.

The aim of this project is to discover the most suitable approach to generalised gesture recognition. This begins with an extensive review of the literature, from historical 'landmark' papers to the present day. The review also includes university theses and both commercial and open source applications. The input devices, data to represent gestures and machine learning algorithms are briefly explained, with detailed discussions on the aspects which appear to be most popular or have the most potential.

An attempt is made to incorporate the lesson know quaternion as a gesture representation with one of the most popular machine learning algorithms, the (discrete) Hidden Markov Model. This paper serves to inform readers of the advantages, pitfalls and misconceptions surrounding quaternions as well as proposing solutions to discovered problems.

A piece of software is also created to aid in the implementation of gesture recognition.

Supervisor: Dr. Gavin Brown

Acknowledgements

I would like to thank my supervisor Gavin Brown for sparking my interest in machine learning and for the years of encouragement and support he has given me.

I would also like to thank my family for all the love and kindness that has gotten me to where I am today.

A special thanks goes to Christopher Lee for his comprehensive video lectures on Hidden Markov Models.

Contents

List of Figures

Chapter 1

Introduction

A gesture is a physical human movement [73] of the face, body [81] and joints [117] with the purpose to convey information or meaning [68][96]. There are two main types of gestures: static gestures and dynamic gestures [122]. A static gesture is a particular configuration, pose [122] or still position [46] made by a human. A dynamic gesture is comprised of a sequence of static gestures presented over a period of time [68]. A similar, but lesser known type of gesture, is a continuous gesture. The gesture is tracked [46] and continually mapped to some output [58]. To clarify the difference between static, dynamic and continuous gestures, consider the following scenario. A person is helping to manoeuvre a car by giving the driver visual instructions from outside the vehicle. The helper holds their hand vertically outright with fingers together and palm facing towards the car. This is a static gesture instructing the driver to stop. From this hand position, the helper clenches their fist leaving only the forefinger vertically extended and rotates it in a circular motion, indicating that the driver should turn the car around. This is a dynamic gesture. Finally, the helper turns their hand so as to face their own palm and repeatedly bends the extended forefinger in a beckoning motion. This is a continuous gesture because every bend of the forefinger indicates that the driver should continue to move in the direction of the helper. These are well known gestures in the UK but it should be noted that they may be interpreted differently in other countries or cultures [48]. The act of interpreting gestures is known as gesture recognition. For computers, gestures must be tracked, recognised and interpreted using mathematical algorithms [113].

1.1 Motivation

The most obvious use for gesture recognition is as an alternative input device [10]. It has been used for the navigation of slides in a power point presentation [30][81] and channels on a smart TV's [116]. In operating theatres, surgeons can scroll through digital images without touching the computer and compromising the sterile environment [119][52][40]. In the future surgeons may be able to control robotic scrub nurses. This would reduce delays caused by mishearing and miscommunication with human nurses, ultimately increasing procedure efficiency [49]. Gesture recognition is being used to help the physically impaired and elderly. Although there are no commercially available sign language interpreters, there have been many theses [82][83][121][131][51][57][95] and own projects [59][79][38][39] on the topic. The first com-

mercial product, PSLT (Portable Sign Language Translator), is estimated to be available this year [99]. The creators, Technabling (a spin out company from the University of Aberdeen), have used normal phone and laptop cameras to capture the gesture so the software can be used on portable devices. They hope this will empower sign language users and improve employment opportunities by lowering the barrier between dead and hearing people [45]. Socially assistive robots encourage social, emotional and cognitive growth in people, including those with social and cognitive deficits [33]. The robots need to be able to recognise gestures made by the humans they are interacting with as well as detect their emotional state [98]. They support the elderly by encouraging physical exercise [56]. The most commonly seen use of gesture recognition is in the gaming industry. In 2003, PlayStation released the EyeToy [69] for PlayStation 2, a 2D full body motion tracking camera to be situated facing the user. It was the first of it's kind to put the user's image on screen and translate body movements into on screen interactions [87]. It's successor, the PlayStation Eye, was released in 2007 for the PlayStation 3 with a better quality camera [112], but has mainly served to track the movement of the PlayStation Move controller since the release of PlayStation Move in 2010 [71]. According to PlayStation, the combination of both spatial and button input gives the user physical interaction as well as precise control and a simple, fast, reliable way to trigger actions [87]. In 2006, Nintendo released the Wii console [111] where gestures are tracked using a 3-axis accelerometer and infrared camera (to track infrared light emitted from the Wii console sensor bar) in the Wii Remote Controller [118]. Also in 2010, Microsoft released the Kinect for the XBOX 360 [70], licensed hardware from Iraeli company, Primesense [64]. Similar to the PlayStation Eye, it is a camera that sits facing the user. In addition to a 2D colour camera, the Kinect has an infrared emitter and infrared camera (receiver) that, used in combination, can map the depth of the environment and user in 3D [90] based on the length of time it takes for the emitted light to return. More general gesture recognition software is available commercially [27][34][11] and open source [58] to allow users to control a variety of devices across different operating systems and platforms using either a provided, own or device integrated camera. Some more abstract gesture recognition is to dynamically make music [129] or manipulate objects in augmented reality [120]. The lack of commercial software could be explained through a quote by TV analyst Paul Gagnon at this years CES event: "Most interaction I've had with gesture and voice control... it's not real great right now. Right now, a lot of people in the industry are just trying to explore the possibilities." [124].

1.2 Aims and Objectives of the Project

There are a variety of input devices, gesture representational data and machine learning algorithms used in various combinations to achieve gesture recognition. The aim of this project is to discover, through research and implementation, the most suitable approach to generic gesture recognition. This aim can be dissected as follows:

- Recognise static gestures
- Recognise dynamic gestures
- Find the most appropriate data to represent gestures
- Experiment with machine learning algorithms and their parameters

- Recognise gestures in real time
- Map recognised gestures to a game (proof of concept)

As the project progressed, it was evident that a system would be needed to make gesture input, training and testing easier. As well as assisting to reach the above goals, the system will be useful as a future teaching/learning tool.

1.3 Overview of the Report

Chapter 2 provides an overview of approaches to gesture recognition taken by related works. Chapter 3 discusses some of these approaches in more detail and explains the decisions made in choosing the input device, gesture representations and machine learning algorithms for this project. Particularly complex implementation details are explained in chapter 4. The software created is demonstrated in chapter 5. The results of testing and evaluation follow in chapter 6. The report concludes in chapter 7 with a comparison to the initial project aims and suggestions for further work.

Chapter 2

Background

Gesture recognition shares similarities with many predecessor recognition tasks such as speech recognition, bio-informatics [64] and hand writing recognition [104][62]. Hand writing recognition is similar to static gesture recognition because they are based on data (usually an image) from a single moment in time. Speech recognition is more like dynamic gesture recognition because the data varies over time.

2.1 Approaches to Gesture Recognition

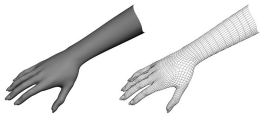
The approach to gesture recognition can be summarised as the input device/method used to record the gesture, the data used to represent the gesture and the machine learning algorithm to recognise the gesture.

2.1.1 Input methods

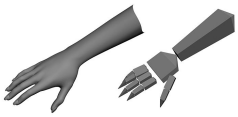
Gestures can be captured using either a vision based or sensor based approach [68]. In the vision based approach, an input image is captured by a 2D or 3D camera. Depending on the way in which the image is analysed, one of two children approaches can be taken. In the appearance based approach, features are extracted from the visual appearance of the image such as colour, size and shape [37]. Local (within the image) positional and rotational data can be extracted by projecting the image into the 2D plane [121][81]. In the 3D model based approach, the image and depth of field is used to create a volumetric representation such as a mesh [100][60] or to try and infer parameters like the position and pose of the user. As this approach is considered computationally expensive [101], a simplified Skeletal based version containing only key joints can be used instead.

The sensor approach is where mechanical or optical sensors are attached to the user to directly output data such as position, angles and orientations. This technique is found in data gloves [8] with sensors for the hand and each finger. It is worn by the user and is popular for object manipulation in virtual reality [8]. The sensors are usually a combination of accelerometers, gyroscopes and magnetometer [106] [28] [29]. An accelerometer measures the force of acceleration, by gravity or other movement. Therefore, it can measure velocity and position [29].

(a) Visual appearance based



(b) Visual 3D model based



(c) Visual 3D model, skeleton based

Figure 2.1: Capturing Gestures in the Visual Based Approach

However, the acceleration is local [130] so the same gesture performed in different directions will classify the same. If the object is stationary, the orientation (pitch and roll only, see figure 3.5) can also be calculated, but upon movement, the acceleration is added to the gravitational acceleration and becomes indistinguishable. To overcome the problem of movement, gyroscopes are added. Gyroscopes measure the rate of rotation around an axis, so used with the accelerometer, can provide the absolute angle of rotation around an axis. As yaw is orthogonal to gravity, accelerometers cannot calculate the angle around the yaw axis given the angular rate of change from the gyroscope. A magnetometer (which can sense the Earth's magnetic fields), GPS or similar can provide an absolute orientation for the gyroscope data to reference.

It can be seen that the choice of input device dictates the data that can be obtained and subsequently used to represent the gesture. However, positional data is often used because "although positional data is not important for posture recognition, positional data is required for gesture recognition because the trajectory of the hand must be tracked" [95].

2.1.2 Machine Learning Algorithms

Various machine learning models can be trained and subsequently used for classification depending on the data collected from the input device. Here follows a brief explanation of the most common models.

A Hidden Markov Model (HMM) can accept time-sequential data [73] and are therefore suitable for dynamic gesture recognition. First, consider a HMM to be a collection of states connected by transitions where each state represents an observable, physical event [105]. The system can be in any one state at a single moment in time, where each state depends only on the state preceding it (the Markov property). Now consider a system where the states are not directly observable (hidden), but some observation, dependant upon the state, is visible (see figure 3.7). As a concrete example, a coin is being tossed in another room and only the result of 'heads' or 'tails' is being relayed back to you. It is unknown in what state the coin was in (i.e. fair or biased) when it was being tossed. There are two variants of HMMs that work on discrete (DHMM) or continuous input data (CHMM).

Dynamic Time Warping (DTW) is especially applicable to signals that vary in length of time [74]. The sequences are warped to match each other in length by repeating one or more elements in either sequence (see figure 3.10). The distance between each sequence (a measure of

how comparable they are) is calculated by summing the 'distance' between corresponding elements [117]. For classification the incoming gesture is compared to all other known instances of gestures. A machine learning algorithm such as Nearest Neighbour (3.3.2) is needed, where the incoming gesture is classified as the gesture of the sequence it is closest to [57].

A neural network consists of a number of parallel processing units with multiple inputs and one output [95], see figure 2.2. There are connections between the nodes (neurons) with weights (importance) given to them [122]. Each piece of input is fed to a node in the starting layer. As the inputs are passed along to the next nodes, the input is multiplied by the weights of the connections they follow. When the weighted inputs reach a node in the next layer, they are summed and passed along through the rest of the network, if the current value is high enough. Upon reaching the output layer, classification is usually by the node with the highest value [95]. Neural networks can also be used to identify groups of data (similar data has the same node) [122][51].

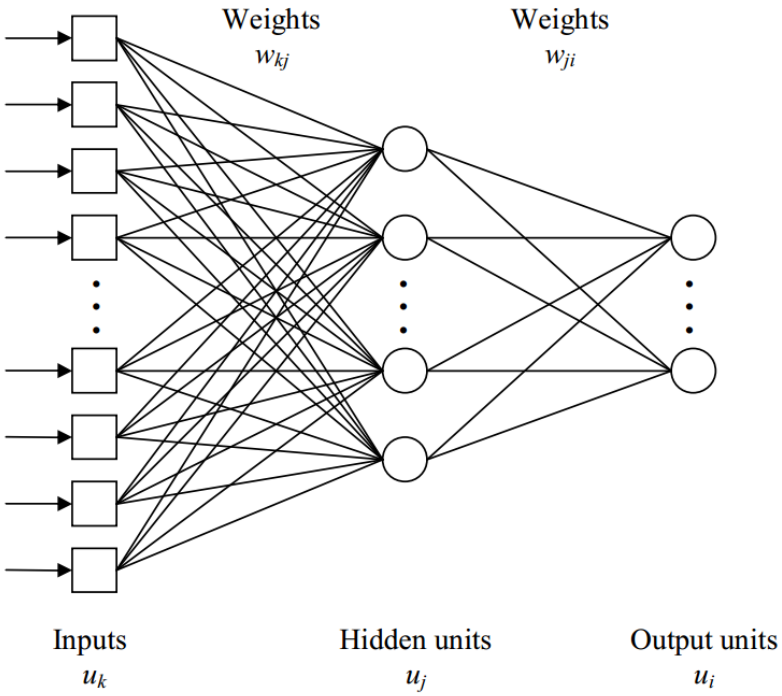


Figure 2.2: A Neural Network [131]

2.2 Literacy Review

The literacy review shows the variety of combinations in approach.

2.2.1 Visual Appearance based

In 1992, Yamato et al. recognised tennis actions (forehand stroke, backhand stroke, forehand volley, backhand volley, smash and serve) with over 90% accuracy [73]. Using a video camera,

gestures are captured as time-sequential images. Each image is first binarised to extract the user from the background, as in common in visual based approaches. The binary image is then divided into blocks where the ratio of pixels (number of black pixels divided by the number of pixels in the block) is calculated as a feature, sometimes known as a mesh feature [126], see figure 2.3.

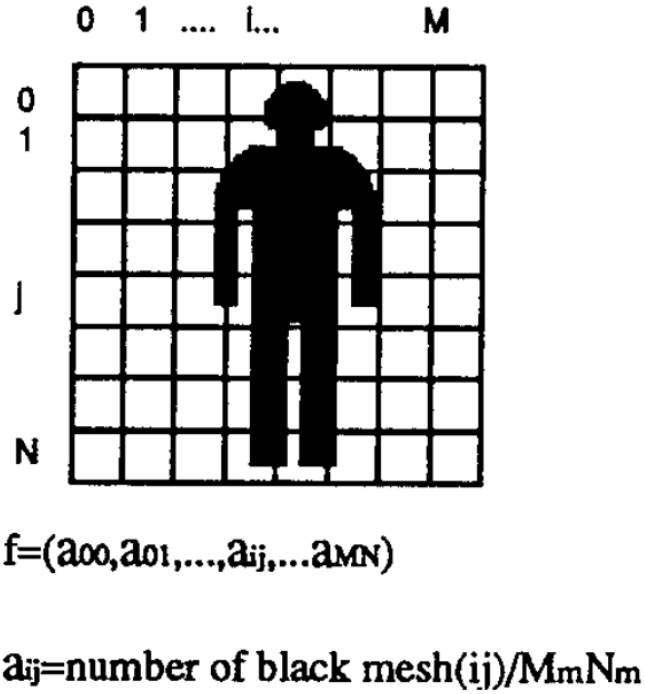


Figure 2.3: The (mesh) features of a gesture [73]

To be able to use this type of continuous data in DHMMs, vector quantisation is applied to assign the mesh feature vector to one of the DHMM output symbols. This involves dividing the feature space into the appropriate number of clusters and representing each cluster centre by one of the DHMM output symbols. The mesh feature vector is assigned the output symbol whose cluster centre is closest to it in the feature space. In summary, a sequence of (mesh) feature vectors are extracted from the image sequence and transformed to a symbol sequence by vector quantisation. This is a common process when using DHMMs and is shown pictorially in figure 2.4.

A DHMM is trained for each gesture. Incoming gestures are tested against each DHMM and recognised as the gesture whose corresponding model produced the highest probability, indicating the best match.

In 1999, Lee and Kim recognised 10 single hand gestures to navigate a power point presentation in real time with over 93% accuracy. Similarly to Yamato et al. the image is first binarised and the hand region detected. This is used, across all time-sequential images, to build up a hand trajectory that is projected into the 2D plane. Between sequential images, the direction of the trajectory (normalised directional unit vector) is calculated as a feature vector (contain 2 values, the sin and cos) using the hand position coordinates. The feature vectors are assigned a DHMM symbols using vector quantisation, see figure 2.5. The sequence of symbols is used as input to

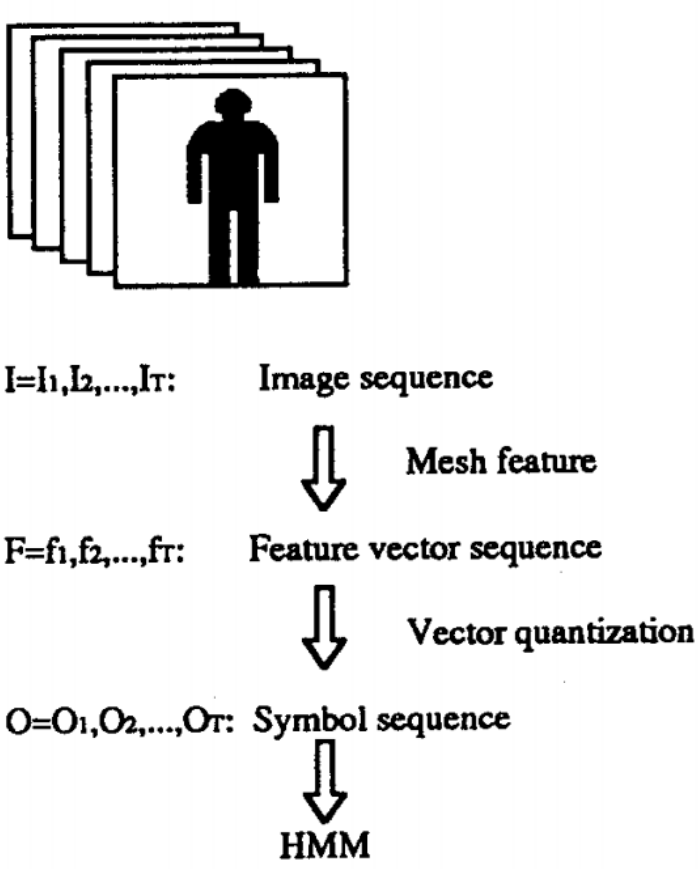


Figure 2.4: Process of using continuous data in DHMMs [73]

the DHMM.

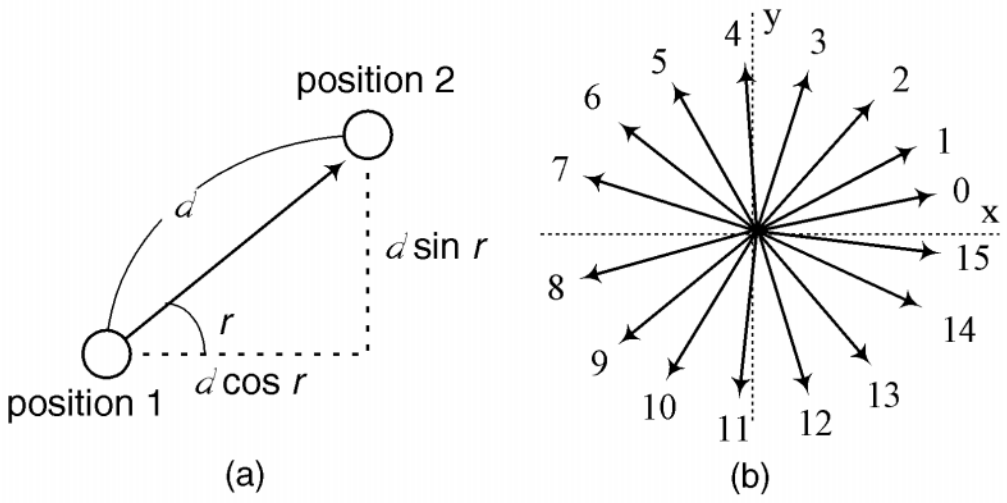


Figure 2.5: (a) Hand trajectory as a feature vector $(\cos\theta, \sin\theta)$ and (b) The vectors representing HMM output symbols used during vector quantisation [81]

Starner and Pentland recognised sentences formed by combining 42 different words (without

finger signing) with over 90% accuracy in 1995 [121]. The input image is binarised and the hands (wearing distinctly coloured gloves) are detected. Using the hand position coordinates, some concept of the shape and angle of the hand (relative to the horizontal) can be calculated by bounding the hand region with a minimum volume enclosing ellipsoid [94]. The shape is represented by the major axis (the longest diameter [109]) and the angle is a measure of the eccentricity (how off-circular it is [108]). This 8 element feature vector is used as direct input to a CHMM.

Although the following approaches are still in the visual appearance domain, they only aim to recognise static gestures through images at a single moment in time.

A Master's thesis by Symeonidis in 2000 recognised words and letters from American Sign Language through finger spelling [131]. The feature vector for each static gesture is an orientation histogram [127], specifically edge orientation [125]. The edges of the hand (shown in figure 2.6c) are detected in both the horizontal and vertical directions as seen in figure 2.6.

(a) Horizontal edge detection

(b) Vertical edge detection

(c) Image containing a static hand gesture/pose

Figure 2.6: Edge detection of a static hand pose [122]

Both resulting images are split in to blocks and the direction and intensity of the gradient for each block are calculated. The corresponding block gradient from each image are combined giving an overall gradient orientation. The orientations are then counted (in bins i.e. 0°-30°...330°-360°) to form a histogram shown in figure 2.7. The histogram values are used as input to the nodes of a neural network.

A laboratory exercise from the University of California extracts the hand from the image, makes it greyscale and uses the colour intensity of the pixels as input to a neural network [131].

Stergiopoulou and Papamarkos identify raised fingers as a necessity towards sign language interpreters in 2009. The images have plained white backgrounds and so the right hand is extracted based on skin tone/colour [51]. As opposed to using neural networks as a classification

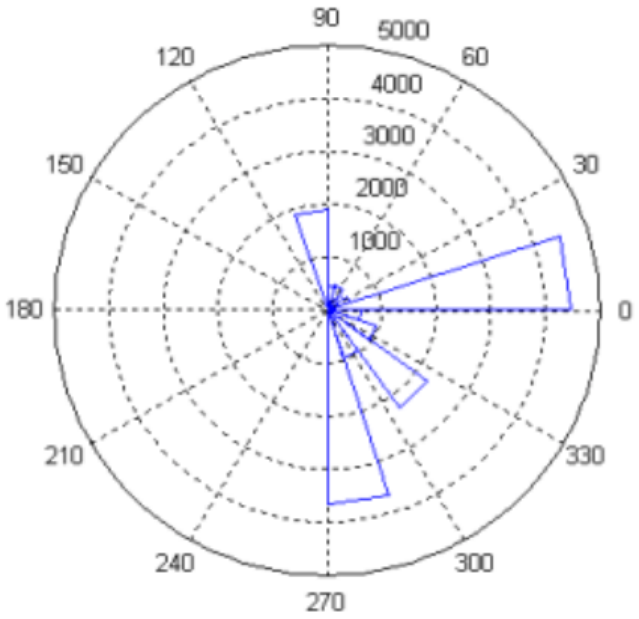


Figure 2.7: Orientation histogram for the gesture in figure 2.6 [122]

algorithm, Stergiopoulou and Papamarkos use a Self-Growing and Self-Organised Neural Gas (SGONG) neural network to approximate the shape of the hand (see figure 2.8). Input to the SGONG is a small number of random samples of hand pixel coordinates and the output is a network of neuron coordinates and their connections. From this, the finger roots (joining to the palm), the finger tips and distance of the finger root to palm centre are calculated.

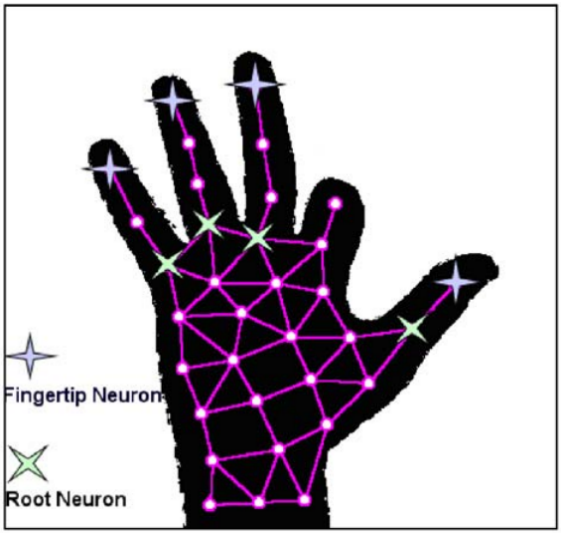


Figure 2.8: A neural network used to group data [51]

For training, the Gaussian distribution for each feature (tip, root and distance) of each class is calculated. For recognition, the same SGONG process is applied to incoming data and the finger tip coordinates, root coordinates and distance from the palm are calculated, see figure 2.9.

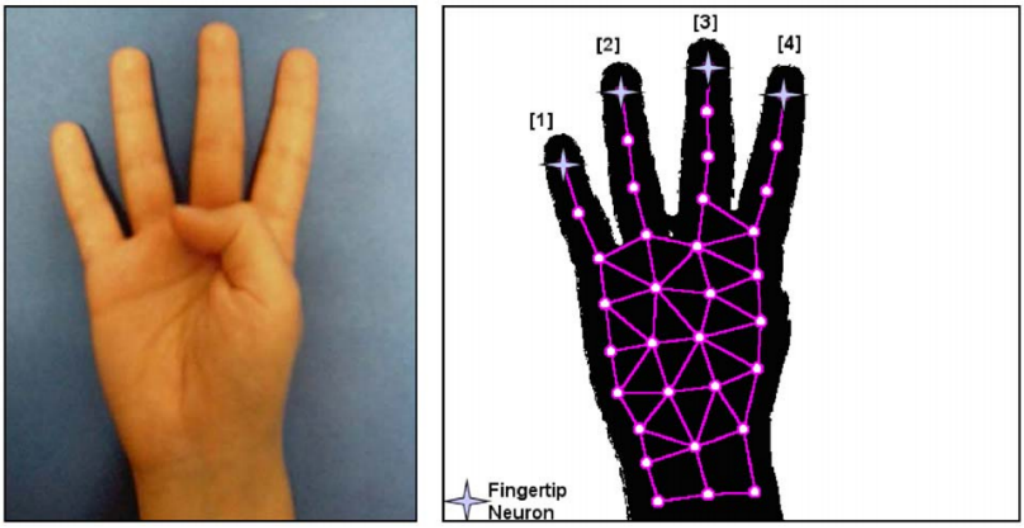


Figure 2.9: An incoming gesture and it's virtual representation [51]

For each detected finger in the image, the likelihood of each trained finger class is calculated by summing the likelihood of the finger tips, root and distance of each class, see figure 2.10. The detected fingers are identified as the finger class with the highest likelihood.

Finger	Features' values		Feature likelihood/class				
			Little	Ring	Middle	Index	Thumb
1.	TC	23.18	0.0143	0.0131	0	0	0
	RC	33.69	0.0329	0.0022	0	0	0
	Distance	−48.9	0.0550	0	0	0	0
	Sum		0.1022	0.0153	0	0	0
2.	TC	4.94	0	0.0336	0.0192	0	0
	RC	10.15	0	0.0402	0.0032	0	0
	Distance	−17.3	0	0.0370	0.0016	0	0
	Sum		0	0.1108	0.0240	0	0
3.	TC	−13.2	0	0	0.0226	0.0208	0
	RC	−16.6	0	0	0.0171	0.0099	0
	Distance	24	0	0	0.0163	0.0084	0
	Sum		0	0	0.056	0.0391	0
4.	TC	−30.3	0	0	0	0.0245	0
	RC	−39.9	0	0	0	0.0168	0
	Distance	61.65	0	0	0	0.0115	0.0301
	Sum		0	0	0	0.0528	0.0301

Figure 2.10: Finger identification of gesture to classify [51]

For the final gesture classification, all combinations of the number of raised fingers detected in the image are considered. For each combination, the likelihood of each raised finger is summed. The gesture is classified as the combination with the highest likelihood.

Little	Ring	Middle	Index	Thumb	Sum
x	x	x	x	–	0.3218
x	x	x	–	x	0.2991
x	x	–	x	x	0.2822
x	–	x	x	x	0.1954
–	x	x	x	x	0.1085

Figure 2.11: Gesture finger combinations are calculated likelihood for the gesture to classify [51]

2.2.2 Visual Model based

In 2007, Holt et al. used 2 cameras in stereo position [36] to 'record' gestures in 3D [57]. The 3D coordinates of the left and right hands relative to the head in each frame were extracted and used as input to a DTW. Classification was achieved with 1 Nearest Neighbour (1-NN).

Kinect SDK Dynamic Time Warping (DTW) Gesture Recognition [114] is open source software that uses the Kinect's depth data with KinectNui (external API) [14] to retrieve the 2D coordinates of the joint positions. It is similar to Holt's work in that it uses DTW with the 1-NN classifier.

The Kinect is again used by Rayes et al. with a different external library, OpenNI [25], to retrieve 3D joint position coordinates, which are then made scale and translation invariant [85]. The paper does not state the classification algorithm, but it is most likely a 1-NN or k-NN. Classification accuracy increased from 60% to 68% when using a DTW that associated weights with each joint depending on it's participation in a particular gesture.

Celebi etc al. also uses the Kinect with OpenNI, a weighted DTW and again the classification algorithm is not divulged [117]. The weights are still based on joint relevancy in gestures, but calculated in a different way. The 3D joint positions are made scale invariant by normalising the person size with the distance between the person's left and right shoulders. The 3D joint positions are made translation invariant, accounting for when the person's shoulder centre is not in the centre of the depth image, by subtracting the shoulder centre from all joint positions. Classification accuracy increased from 60% to 96.7% suggesting that this weighted algorithm is significantly better than that of Rayes et al.

Hall wrote the tutorial 'Gesture Recognition with Kinect Using Hidden Markov Models' [64]. OpenNI is used alongside the Kinect to give 3D coordinate positions of the hands. Hall does not concern himself with normalising the coordinates and uses the data as is. Similarly to Yamato et al and Lee and Kim, the coordinates of each frame are assigned to output symbols using k-means clustering. The sequence of symbols is then used to train and test a DHMM.

2.2.3 Sensor Based

In 2009, Betters and Todoroff attached a 3 axis accelerometer and 2 axis gyroscope to a musicians ankle, so that the musician could change the sound of the instrument they were playing or

trigger events on gesture command [41]. The acceleration and angle velocity are scaled before being used in DTW and an unspecified classification algorithm. The DTW uses a multi-grid technique, where multiple DTW grids are active at any one time, slightly shifted and hypothesising start points. The grid with the shorest path length is assigned as a gesture, removed from the data stream and the process starts again. This allows DTW to be used in real time.

In 2010, Jian recognised typical gestures used in a hotel reception scenario (such as nodding) with 97.5% accuracy at it's best [74]. This was achieved through the use of a 3-axis accelerometers, 3 axis gyroscope and magnetometer on each of the 7 upper body joints that output acceleration, angular velocity and magnetic field readings. This data is processed to produce quaternions (see 3.2.4.2) representing the rotational orientation of the joints. The quaternions are used as input to the DTW and classification performed by 1-NN. The concept of windowing allows incoming gestures to only be compared with gestures that are closer in length, reducing computation time.

Murakami and Taguchi recognised Japanese sign language from data received from a data glove worn by the user in 1991 [95]. At each time step, the bend angle of the fingers, the relative 3D positional coordinates of the hands and the rotation of the hands represented as Euler angles are normalised and used as input to a neural network. For static gesture recognition, each output node corresponds to a character and so the node with the highest value is the assigned class. The recognition rate was 77% for testing data that the neural network hadn't learnt from and 98% with testing data that it had learnt from. For a neural network to have the ability of accepting time-sequential data and recognising dynamic gestures, it must be made recurrent [68] as in 2.12. This means that an intermediate layer in the network is allowed to feedback on itself to a copy that stores all previous values and provides a history of data.

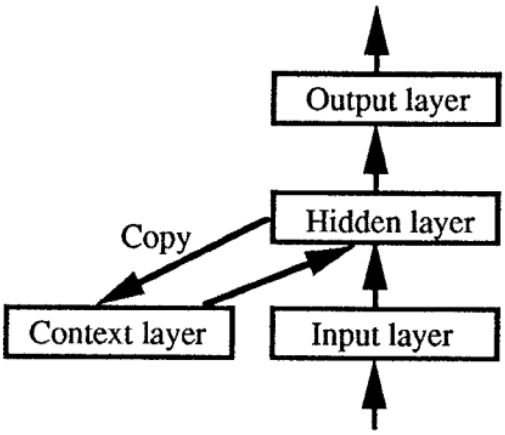


Figure 2.12: Recurrent neural network where 'history' is stored in the Context layer [95]

This paper achieved 80% accuracy which improved to 96% with augmentation and filtering of the input data.

Using a Wii controller, Schlomer et al. recognised shapes such as a square, circle, roll and 'Z' [123]. The Wii has a built in accelerometer and infrared detector which produces the movement, speed and tilt of the controller, as well as it's direction in relation to the Wii console. Schlomer et al. intercept the Bluetooth message constructed by the Wii remote that has been reverse engineered by the open source community. The data provided is transformed into

discrete symbols using k-means clusterings and used to train a DHMM. For a single participant, the highest recogonition rate was 93.4%.

Chapter 3

Design

The following sections looks at some aspects of the approaches seen in the literary review in more details and explains the decisions made in choosing the input device, gesture representations and machine learning algorithms for this project.

3.1 Input Device and APIs

From the literary review, the visual based approach seems the most popular. Whether the appearance based approach (with standard video cameras) or the model based approach (with depth cameras) is used, the majority of applications process the data in order to acquire coordinate positions of joints. The current version of the Kinect SDK (v1.6) provides 3D coordinates for joints via skeletal tracking, as well as other joint information, without having to use an external API. The joints tracked are highlighted in 3.1.

The Kinect SDK is fully supported by Microsoft (unlike the open source reverse engineering of the Wii), contains coding examples and performs the image processing behind the scenes to provide easy access to joint information for the programmer. Therefore the Kinect has been chosen as the method of input for this project.

Any .NET language can be used to access the Kinect SDK, but most of the sample code it written in `C#`. Therefore the project will be written in `C#` using Microsoft Visual `C#`, the suggested IDE.

3.2 Feature Selection: Gesture Representations

For machine learning to work well, 'good' features of the gesture should be selected [65]. 'Good' is characterised as features that highly correlate for each gesture class but differ between gesture classes. This focuses the machine learning algorithm on important aspects of the gesture, leading to intelligent learning and accurate predictions.

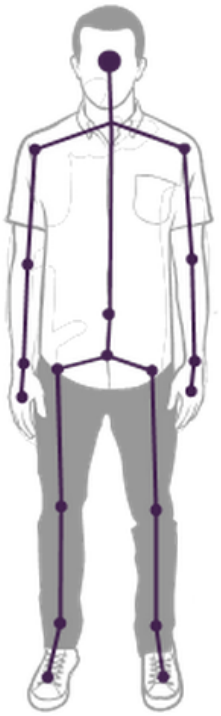


Figure 3.1: The skeleton joints tracked by Kinect [91]

3.2.1 Requirements of the representation

The features chosen to represent a gesture needs to be translation invariant, scale invariant [85] [117] and orientation variant [74].

- **Translation invariance**: The same gesture performed in two different locations should be classified the same.
- **Scale invariance**: The same gesture performed by two different people (i.e. an adult and a child) should be classified the same.
- **Orientation variance**: The same gesture performed in two different directions should be classified differently.

It should be noted that that the more features that are selected, the more data the machine learning algorithm has to cope with and in general, the longer it will take to train and classify.

3.2.2 Coordinates

It is evident from the literary review that coordinates, solely or part of the features selected, are a popular representation for gestures. Coordinates specify the position of a point in x-dimensional space in relation to an origin with x equally spaced, different directional axes. The Kinect provides the 3-dimensional coordinate positions of joints in 3D space, see figure 3.2.

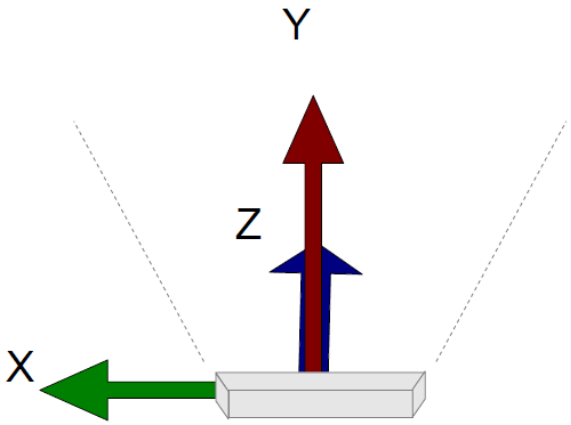


Figure 3.2: The coordinate system as seen by Kinect

The location of the gesture affects the coordinates of the user and therefore coordinates do not provide translation invariance. The size of the user can be thought of as affecting the location of joints in comparison to a user of a different size and hence does not provide scale invariance. Fundamentally, coordinates cannot represent orientation. Some concept of orientation can be found if we consider all the joints in the body. A gesture performed facing slightly to the left would have the left arm in a location behind the right arm. The same gesture performed facing slightly to the right would have the right arm in a location behind the left arm. Therefore the gestures would be recognised differently. However, if we consider only the user's head, recognising a nodding gesture would be unsuccessful because no movement is witnessed: the location of the user's head stays the same.

3.2.2.1 Coordinate Transformations

By translating the gesture to a common area, some of the above problems can be alleviated. The gestures can be translated/shifted by picking a single coordinate (i.e. the hip centre) to map to the origin and mapping the other joints in relation to this. This makes the gesture translation invariant and preserves the illusion of orientation variance.

To overcome scale invariance, each joint position could be scaled in relation to some measurement of the user such as their height or the distance between their shoulders [117].

3.2.3 Angles

An angle is the space between two lines joined at a vertex. In the context of gesture recognition, angles can be found at i.e. the elbow, between the upper and low arm (figure 3.3).

Angles are both translation and scale invariant because they are not innately positional. They can not provide a representation of orientation because angles are only defined in 2D space.

To acquire these angles, the coordinates output by Kinect would have to be post processed.

Figure 3.3: The angle formed between joints

3.2.4 Orientations

Orientation is one's position in relation to a specific place or object [26]. It is expressed by the rotation and/or translation that would be needed to move from the reference object to the new position [63][23]. Kinect provides the orientation of bones in a hierarchical (relative to the direction of the parent bone) or absolute rotation (relative to root parent bone, the absolute orientation of the user) [89]. This is shown in figure 3.4. The two types of rotation are available mainly to support graphics programming, but testing will have to be done to discover which, if any, give the best representation of a gesture. The hierarchical representation is used in [74].

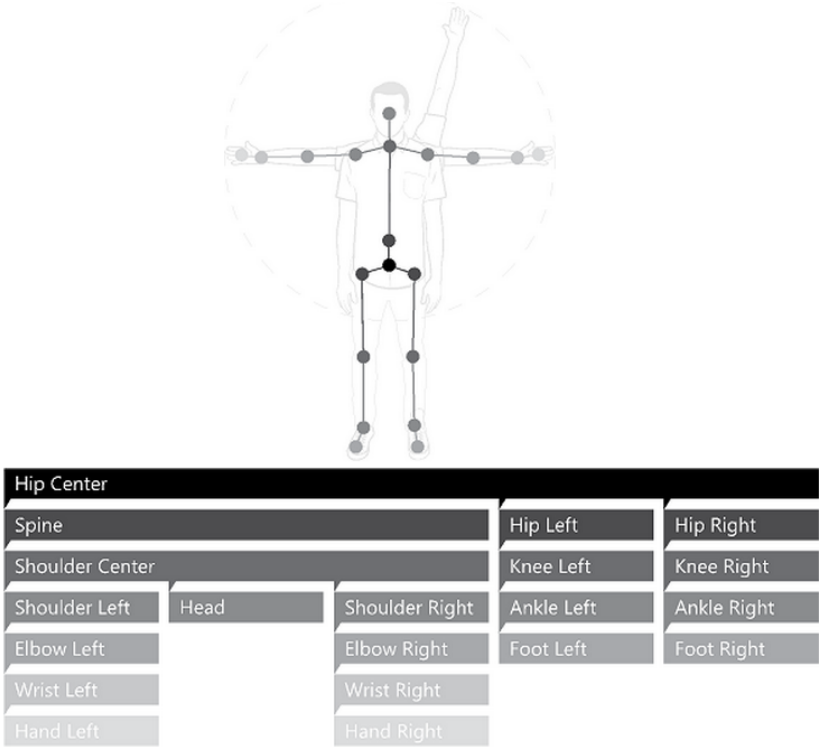


Figure 3.4: The Kinect arranges bones in a hierarchical structure [89]

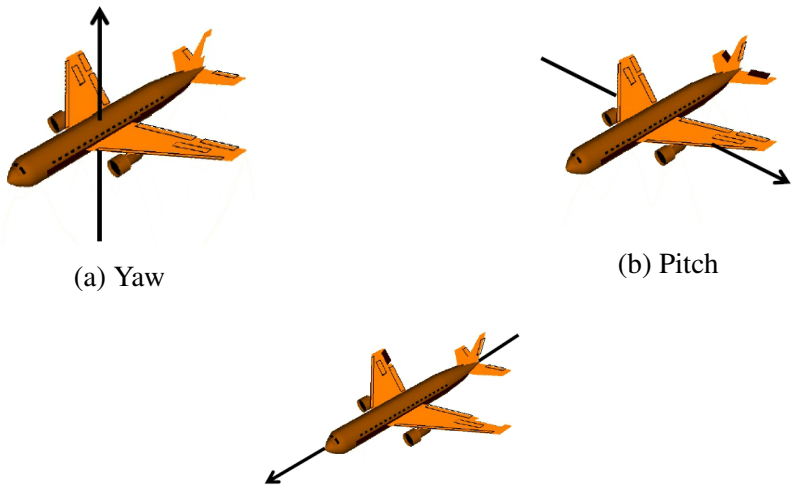
As well as providing orientation variance, bone orientations are not positional and so provide translation and scale invariance.

The different ways to represent the rotations of bone orientations will now be discussed.

3.2.4.1 Euler Angles

Euler angles describe orientations in 3D space and consist of 3 sequential rotations (described by 3 angles) around the 3 different axes [22]. The rotations are embedded meaning that the rotations occur around the rotated objects local coordinate system (known as the reference/rotational frame) as opposed to the world coordinate system.

The rotations are Yaw (around the Y-axis), Pitch (around the X-axis) and Roll (around the Z-axis), applied in that order. See figure 3.5.



(a) Yaw

(b) Pitch

(c) Roll

Figure 3.5: Possible Orientations of an Object in 3D Space [97]

A major problem of Euler angles is that they suffer from Gimbal Lock [7] [84]. If one of the Euler angles is 90°, the local axis becomes aligned ('locked') to one of the world axis where the rotations originally began. Rotating around the local axis will be equivalent to rotating around one of the original axes and so a degree of rotation is lost. In the worst case scenario, all three axes become aligned losing two of the three degrees of rotation. This is explained diagrammatically in the appendix A.1.

This concern is more prevalent in graphics processing but could be problematic when calculating Euler angles from the coordinates given by the Kinect.

3.2.4.2 Quaternions

Quaternions have many uses and can be applied to vectors to achieve translation, reflection, scale and rotation as well as calculating parallel and perpendicular components to a plane [15]. The way in which the components of quaternions are calculated and used in each of these

transformations differs and not all material available is clear as to which representation is being referred to. So here follows an explanation of quaternions used to represent rotation and hence providing an orientation in 3D space.

The following information has been collated from [15][67] [74][35][103][24][66] and [54].

Quaternions are a 4 dimensional number system that extend the complex numbers. Therefore a quaternion has 4 components: 1 in the real dimension and 3 in the imaginary dimensions i, j and k.

$$q = W + X_i + Y_j + Z_k$$

shorted to

$$q = W + X + Y + Z$$

where q is an arbitrary quaternion. The value of the components encode the vector to rotate around, the angle of rotation and a scalar to chance the size of the vector being rotated. A normalised vector represents a pure rotation with no scaling, which is the type of quaternions output by the Kinect.

$$q = W^2 + X^2 + Y^2 + Z^2 = 1$$

The value of the components of a normalised quaternion are calculated by

$$W = \cos(\theta/2)$$
$$X = \sin(\theta/2) \times x$$
$$Y = \sin(\theta/2) \times y$$
$$Z = \sin(\theta/2) \times z$$

where θ is the angle of rotation and (x,y,z) is the angle to rotate around.

Quaternions have two equal representations. Rotating an object clockwise around the y-axis $60^\circ$and rotating an object anti-clockwise 300°result in the same orientation for the object.

$$W + X + Y + Z = -W - X - Y - Z$$

To apply a quaternion to a vector, one must represent the vector (x,y,z) as a quaternion, by zeroing the real number.

$$0 + x + y + z$$

The conjugate of the rotation quaternion (q^{-1}) must be calculated. This represents the same rotation in the opposite direction by reversing the rotation vector.

$$q^{-1} = W - X - Y - Z$$

To formula to apply a quaternion to a vector is

$$v' = q * v * q^{-1}$$

where v' is the resulting vector represented as a quaternion.

The definition of quaternion multiplication is

$$\begin{aligned}
q'_W &= (q1_W \times q2_W) - (q1_X \times q2_X) - (q1_Y \times q2_Y) - (q1_Z \times q2_Z) \\
q'_X &= (q1_W \times q2_X) + (q1_X \times q2_W) + (q1_Y \times q2_Z) - (q1_Z \times q2_Y) \\
q'_Y &= (q1_W \times q2_Y) - (q1_X \times q2_Z) + (q1_Y \times q2_W) + (q1_Z \times q2_X) \\
q'_Z &= (q1_W \times q2_Z) + (q1_X \times q2_Y) - (q1_Y \times q2_X) + (q1_Z \times q2_W)
\end{aligned}$$

where q' is the resulting quaternion, $q1$ is an input quaternion and $q2$ is another input quaternion.

The advantage of rotating vectors using quaternions is that the rotation is applied in 4D space, avoiding any problems such as Gimbal lock (see 3.2.4.1) and the resulting vector can easily be transformed back to 3D space. The reason that the angle *theta* is halved in the equations is because quaternion multiplication involves applying two quaternions, the original and it's conjugate. See figure 3.6.

Quaternions are directly output by the Kinect in both hierarchical and absolute form. The Kinect also outputs the matrix form of quaternions with 16 elements. This was disregarded due to the large number of elements to describe the same information which would increase the time taken by the machine learning algorithm.

3.2.4.3 Summary

Coordinates can provide translation and scale invariance if they are post processed. However, they can only provide an illusion of orientation. Angles cannot provide any concept of orientation and have therefore been disregarded. Euler angles can provide all three requirements but pre processing of coordinates (or quaternions) would have to be applied because the Kinect does not supply this information. Quaternions also provide all three requirements and their only disadvantage is that they are not easy to read, debug and material on the topic is of a lower quality than more popular representations. The difference between the use of three features of the coordinates/Euler angles and the four features of quaternions is insignificant compared to the pre and post processing that would be needed for use these former representations. The table below summaries this information.

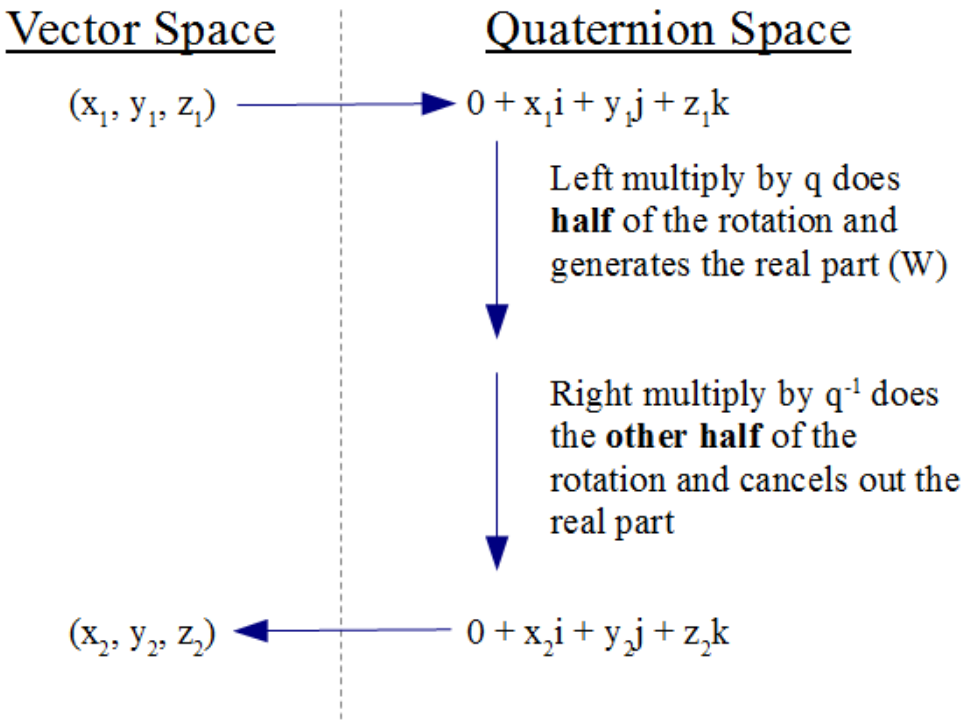


Figure 3.6: Applying quaternion rotations to a vector

	Coordinates	**Euler Angles**	**Quaternions**
Readability	High	Medium	Low
Number of features	3	3	4
Provided by Kinect?	Yes	No	Yes
Usable without post processing?	No	Yes	Yes
Translation Invariant (after processing)?	Yes	Yes	Yes
Scale Invariant (after processing)?	Yes	Yes	Yes
Orientation Variant (after processing)?	Partially	Yes	Yes

Quaternions have been chosen as the best gesture representation. Due to the difficulty in reading, understand and debugging quaternions, coordinates will also be used in the project as an alternative representation.

3.3 Machine Learning Algorithm

Machine learning must be applied to a history of known gestures in order to predict future gestures. For the historical data, this project will record both the gesture performance and the name of the gesture, therefore supervised learning [88] machine learning algorithms (which utilise this information) will be considered.

From the literary review, it can be seen that certain machine learning algorithms are more suited to the different types of gestures than others. It was evident that static and dynamic

gestures should, at first, be considered separately, as dynamic gesture recognition is more complex.

3.3.1 Requirements of the machine learning algorithm

Aside from classification, it would be useful if the machine algorithm was able to

- distinguish between gestures and non-gestures: Applications using gesture recognition are more usable if gestures can be performed at any time without having to manually trigger 'recognise' commands.
- recognise gestures in real time: An application should be able to tell the difference between a purposeful gesture and an irrelevant movement of the body.
- accept continuous data: The gesture representation of quaternions and coordinates are both continuous data.

The rest of this section discusses the appropriateness of some popular machine learning algorithms.

3.3.2 Static Gestures

For most of the machine learning algorithms in the literary review, dynamic gesture recognition was an umbrella for static gesture recognition. The only algorithm solely suited to static gesture recognition was the neural network. This is a rather complicated model to begin the project with when simpler algorithms for static gesture recognition exist. The sections following discusses some simple, quick to implement machine learning algorithms suited solely to static gesture recognition. The intention is to implement a simple static gesture recogniser with quaternions to ensure they are the correct gesture representation to use before moving onto dynamic gesture recognition (and static gesture recognition as a subset) as soon as possible.

Nearest Neighbours
The nearest neighbour algorithm is considered one of the simplest machine learning algorithms [53]. To classify an incoming gesture, it must be compared to all other seen gestures and the closet matches selected. The majority gesture of the closest gestures selected is assigned to the incoming gesture. Nearest neighbour does not provide a means of recognising non-gestures because it does not produce a probability of how likely the assigned class is to be correct. Real time gesture recognition could be problematic if there are a large number of known gestures to compare against incoming gestures. However, it is well suited to continuous data, depending on the algorithm implemented to calculate the distance (closeness) between gestures.

Logistic Regression
Although logistic regression didn't feature in the literary review, it has been applied to static gesture recognition before [82]. The input to logistic regression can be binary, discrete or continuous and the output is the probability of one of two classes. Therefore it can be used to classify between two classes and a numerical threshold can be applied to the data to identify

non gestures. As with any other linear model, it works by trying to partition the data with a straight line such that the number of classification errors is minimal. By substituting the linear model into a logistic function (sigmoid function/logistic curve), a graph with values between 0 and 1 is produced. This means the value from the graph can be used directly as a probability [43]. For problems with more than two classes of gestures, the logistic regression model can be generalised to a multinomial logistic regression (softmax logistic regression) [19][18][20]. To arrive at the multinomial model, a 'pivot' is chosen from one of the K number of gesture classes (i.e. the last one, K). $K-1$ binary logistic regressions are made, each classifying K against one of the other gestures. Using the logistic function formula and weights for each model, along with the fact that all of the K probabilities must sum to 1, formulas can be constructed to give the probability of each of the K gesture classes. These formulas each contain a sum over all the models, but unless there was a significant amount of gesture classes then this model is suitable to real time gesture recognition.

Static Gesture Summary
Despite the fact that logistic regression provides all the requirements of a machine learning algorithm, the nearest neighbour neighbour classifier is being chosen to implement static gesture recognition. It is quick and easy to implement and will surface any unknown problems associated with quaternions that could affect dynamic gesture recognition. The nearest neighbour classifier is also used in many of the Dynamic Time Warping applications in the literary review, so it may be useful for future implementation choices.

3.3.3 Dynamic Gesture Recognition

The two most popular machine learning algorithms from the literary review are Hidden Markov Models (HMMs) and Dynamic Time Warping (DTW) which are discussed below.

Hidden Markov Models (HMMs)
The following information is collated mainly from [80] and related video lectures, [105][81][50][64] and others from the literary review.

The joint distribution of any variable from a set of random variables can be calculated using conditional probabilities. This is the chain rule.

$$P(\cap_{k=1}^{n}, A_k) = \sqcap_{k=1}^{n} P(A_k | \cap_{j=1}^{k-1}, A_j)$$

For example:

$$P(A_4, A_3, A_2, A_1) = P(A_4|A_3, A_2, A_1) \times P(A_3|A_2, A_1) \times P(A_2|A_1) \times P(A_1)$$

The Markov property is the assumption that the probability of any variable in the chain depends only on the previous variable and not on the sequence of states that precede it. Applying this to the above:

$$P(\cap_{k=1}^{n}, A_k) = \sqcap_{k=1}^{n} P(A_k|A_{k-1})$$

For example:

$$P(A_4, A_3, A_2, A_1) = P(A_4|A_3) \times P(A_3|A_2) \times P(A_2|A_1) \times P(A_1)$$

This is known as a Markov chain. A Markov chain is a model of a mathematical system, where the variables mentioned are discrete and represent states. Any state can transition to another state with some probability, possibly zero. A state sequence is a set of executed transitions where each transition made is a step in time.

A Hidden Markov Model is a Markov chain where the states are not directly observable (hidden); instead an observation (output), that is dependant on the state, is visible (see figure 3.7). In other words, each state emits an observation (from a set of observations available to all states) with some probability. This means that at any time, one does not know the current state of the system, but has a probabilistic insight into state of the system based on the observations seen.

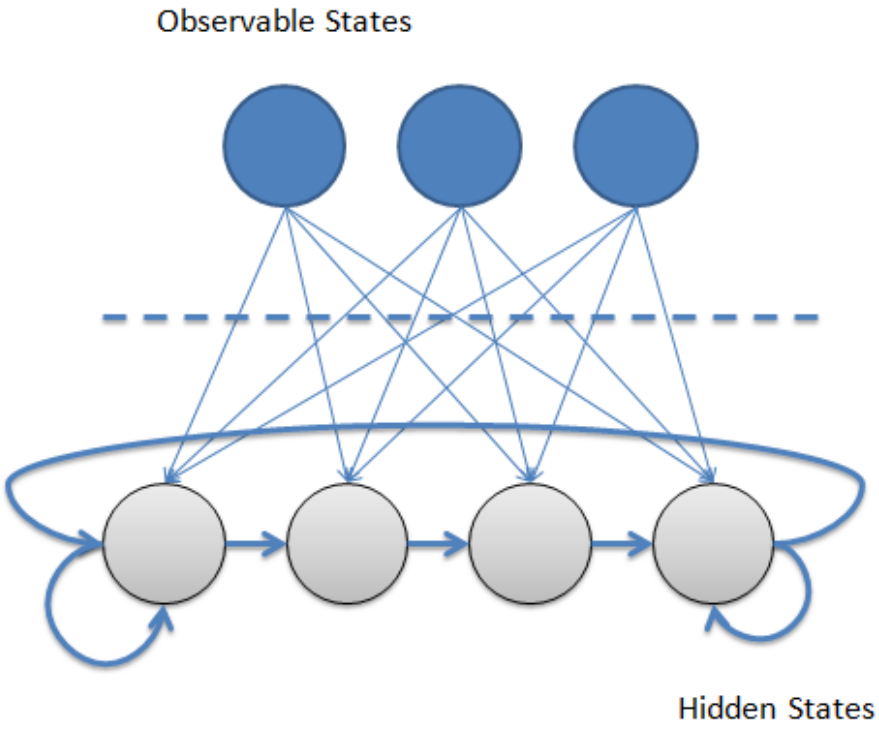


Figure 3.7: A Hidden Markov Model (HMM) [12]

HMMs are traditionally defined by: $\lambda = (N, M, A, B, \pi)$ where

- N is the number of states for the model.
- M is the number of observations (observation symbols) per state. This is also known as the alphabet size.

- *A* is the NxN state transition matrix, the probability distribution of transitions between states.
- *B* is the NxM emission matrix, the probability distribution of the observations from each state.
- π is the prior probability matrix (vector), the initial probability distribution of the states at time 0.

As a concrete example, imagine a casino where one can witness the outcome of a die (1-6) but not the moment when the dealer chooses which die to roll. The die chosen may be either fair or loaded (biased towards the number 6) which represents the state. The results of rolls are the observations. There is a probability that the dealer swaps between a fair and loaded dice (transition probabilities) and the loaded dice has a higher probability of rolling a 6 than the fair dice (emission probabilities). There is also a probability distribution across the first die to be chosen (prior probabilities). With gestures, the observations are the quaternions/coordinates of the joints. The states are not as obvious as in the casino example. The states are an abstraction, as are the transition, emission and prior probabilities.

There are two types of HMMs depending on the distribution of the transition probabilities. An ergodic HMM (figure 3.8) is one in which all states are reachable from all other states.

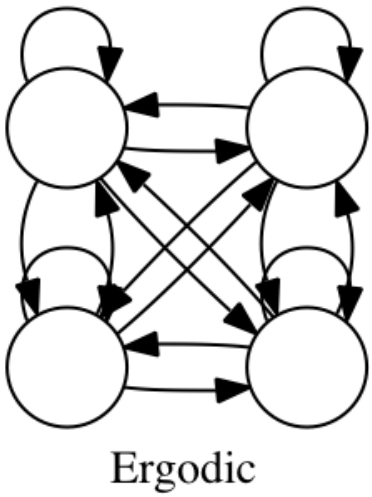


Figure 3.8: An Ergodic HMM [44]

A forward HMM (figure 3.9) only allows transitions from one state to following (forward) states.

The latter is more suitable for gesture recognition because it eliminates the possibility of jumping between distinct states and restricts the model of gestures to an adjacent sequence of states, as is what constitutes a gesture in nature.

There are well known problems to be solved when using HMMs:

- **1: Evaluation**: Given a model and an observation sequence, compute the probability that the observation sequence was produced by this model.

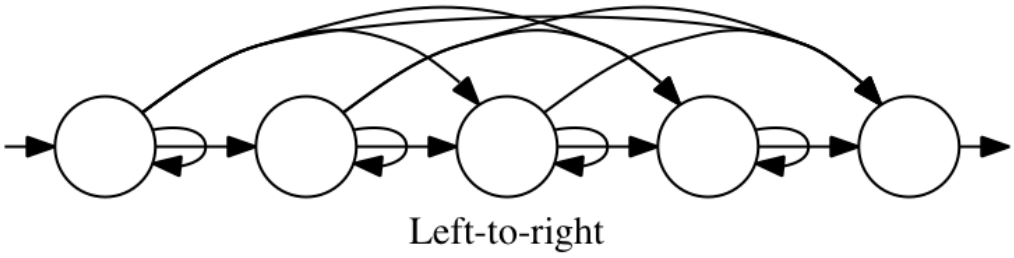


Figure 3.9: A Forward (or Left-to-Right) HMM [44]

- **2: Decoding**: Given a model and an observation sequence, infer the most likely sequence of states that produced the given observation sequence.
- **3: Learning**: Given either an observation sequence or a set of observation sequences, compute the optimal model parameters, as to maximise the probability of each observation sequence originating from the model (problem 2).

To solve these problems, the following algorithms are used:

- **Viterbi algorithm**: solves problem 1 and 2. Given an observation sequence, it calculates the most likely path/sequence of states through the model and it's probability.
- **Forward algorithm**: solves problem 1. It is very similar, but subtly different to the Viterbi algorithm. Given an observation sequence, it infers the most likely state at each time step given the previous sequence of states.
- **Baum-Welch algorithm**: solves problem 3. Given an observation sequence (or set of observation sequences), it finds the most likely parameters (transition, emission and prior probabilities) for a model that produced the observations sequence(s).
- **Forward-Backward algorithm**: used as part of the Baum-Welch algorithm. Given an observation sequence, it calculates the posterior probabilities of individual states (also known as the posterior marginals) at a certain time given the entire observation sequence (as opposed to just the previous observations in forward algorithm).

Choosing between DHMMs and CHMMs Continuous Hidden Markov Models (CHMMs) are ideal because the chosen gesture representation of quaternions and coordinates are continuous data. However, the algorithms are more complex in CHMMs than in Discrete Hidden Markov Models (DHMMs) because instead of the emission probabilities being discrete integers, it is now a continuous probability distribution [47]. To avoid using CHMMs, it is possible to use DHMMs with continuous data discussed in the following section, 3.3.3

Using DHMMs with Continuous Data To use continuous data as input to a DHMM, the data must be transformed to discrete data. This can be achieved by k-means clustering [73][81][64][123][72] as seen in the literary review. For the training process, given an observation sequence or set of observation sequences, the similar observations are grouped together using k-means clustering[110]. The groups centroids are assigned an integer value, together forming the set of DHMM output symbols. Then each continuous observation sequence is transformed into a

discrete observation sequence; each observation is assigned the integer value of the centroid it is closest to, measured by some calculation of 'distance'. This process can be seen in figure 2.4.

Details for how this was achieved for both quaternions and coordinates can be found in 4.

Dynamic Time Warping

The following information is collated from [74][1] [76][41][117] and others in the literary review.

DTW is primarily used to compare time-sequential sequences of different lengths (time). This is useful for gesture recognition because variations will naturally occur in different people. To do this, the sequences are 'warped' with respect to each other, making them the same length (see figure 3.10)

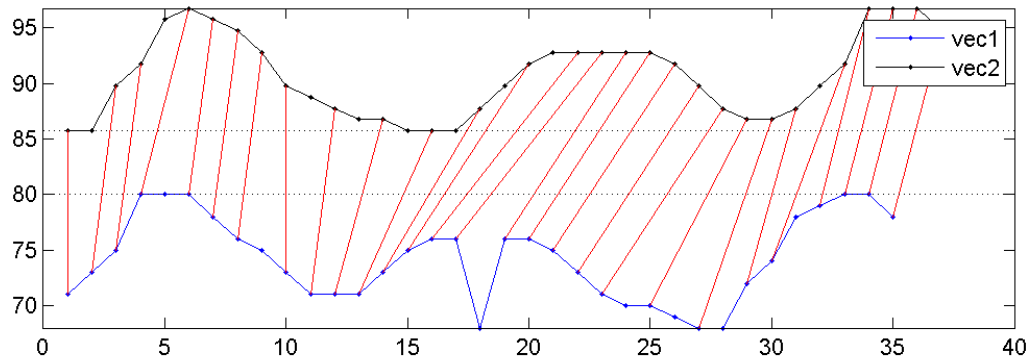


Figure 3.10: Dynamic Time Warping (DTW) between two sequences vec1 and vec2 [1]

To find the best mapping of observations (i.e. the optimal path with the smallest distance) a form of dynamic programming is used. As the sequences are time-sequential, some restrictions have to be imposed:

- **Boundary condition**: The start and end observations are aligned.
- **Monotonicity condition**: The observation mappings are aligning in order of time. i.e. the observations in the observation sequence should not become out of order when mapping.
- **Step size condition**: No observations are to be omitted from the mapping. Strictly speaking, the step size could be increased if one wants to allow outliers to be excluded.

To summaries the latter two conditions, the mapping path will be monotonically increasing in both it's arguments. Further still, given any element of one observation sequence, it should be possible to find at least one corresponding element in the other observation sequence and visa versa.

DTW in itself isn't a classification algorithm. One must use an algorithm such as the nearest neighbour algorithm (see section 3.3.2). In NN, the comparison/distance measure between the incoming gesture and all other gestures is calculated using DTW.

DTW has the ability to identify gesture from non-gesture due to the boundary condition. For an incoming gesture, if the distance between end points, the overall distance between the gestures being compared or the summed distance of all gesture comparisons is too large, that gesture could be classified as a non-gesture. DTW works with continuous data but the potential number of gesture comparisons make it is very computationally expensive and therefore probably not suited to real time gesture recognition.

Dynamic Gestures Summary
Both HMMs and DTW provide the potential to recognise gestures from non gestures and accept continuous data. HMMs are more suited to real time gesture recognition because they are trained offline and the classification algorithm is likely to be quicker than the distance calculations of DTW across all gestures as the number of gestures and gesture lengths increase. Up until the 1980's DTW was the primary method for speech recognition [42] but due to their determinism and lack of power to model stochastic signal [55][75] were replaced by HMMs. [78] states that HMMs outperform DTW in terms of speed (and accuracy) but [107] suggests that DTW has better accuracy, though the results are problem dependant. [76] writes that DTW have the advantage of being able to more accurately compare sequences of different lengths, but simply normalising the sequences to the same length makes little significant difference. [77] compares HMMs and DTWs but without the ability to access the paper, the results cannot be relayed here. Due to the lack of a definitive decision on which is the most appropriate algorithm for gesture recognition and the unsuitability to real time gesture recognition, HMMs have been chosen as the machine learning algorithm for this project, despite the fact that DTW is easier to implement.

3.3.4 Software and User Interface Requirements

The end goal of the project is to map recognised gestures to game inputs (i.e. with an emulator) as a proof of concept for gesture recognition. However, to make the acquisition of gestures, execution of machine learning algorithms and gesture recognition easier, an intermediate piece of software is being created.

The functional requirements of the software need the ability to:

- Add gestures
- Load gestures
- Choose between gesture representations
- Edit the machine learning algorithm parameters
- Train the machine learning algorithm
- Load a trained model
- Recognise newly added gestures in real time
- Recognise loaded gestures

The non-functional requirements of the software are:

- To be easily extendible for other types of gesture representation
- To be easily extendible for other types of machine learning algorithms

The user interface chosen for the software is a wizard style. This application satisfies “Is this the right user interface” [92] in that the tasks are fundamentally sequential and therefore cannot be satisfied with something lighter weight like a dialog box. It will simplify the user experience by guiding the user through the necessary steps to reach gesture recognition.

Chapter 4

Implementation

The main focus of this section is the particularly interesting or troublesome aspects of attempting to integrate quaternions with discrete hidden Markov Models.

4.1 Capturing Gestures: Gesture Representation

When the Kinect is actively tracking a skeleton, it fires a skeleton frame whenever new data is ready. This is approximately 30 times a second. In default mode, 20 joints are tracked with the hip centre as the root. In seated mode, 10 upper body joints are tracked with the shoulder centre at the root. See figure 4.1.

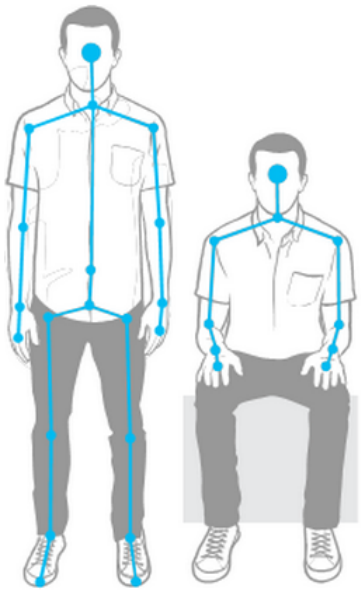


Figure 4.1: The skeleton joints tracked by Kinect in: default mode on the left and seated mode on the right [91]

For specific details of how the Kinect outputs quaternions see 3.2.4.2 and 3.2.4 and for coordinates see 3.2.2.

The process of capturing the gesture is described using coordinates, but is identical for quaternions. For a frame, each joint position is retrieved and stored collectively in a double array as an "*Observation*" (figure 4.2).

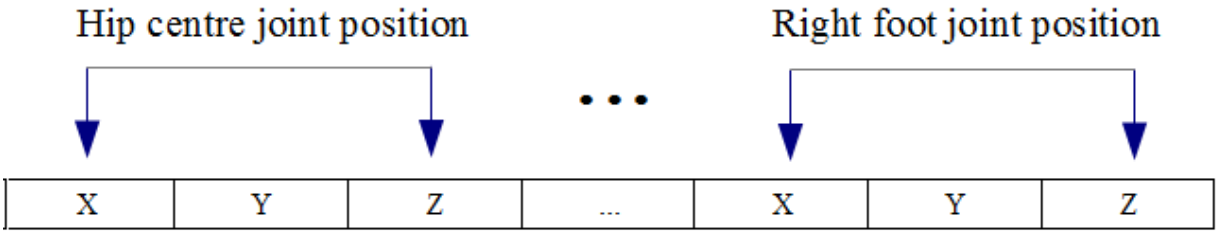


Figure 4.2: A single *Observation* consists of coordinates of all the joint positions on the user's body

This is the entirety of data used in static gesture recognition because it is the state of the body at a single moment in time. For dynamic gesture recognition, a number of observations are collected from a series of frames in an "*ObservationSequence*" (figure 4.3). A continuous gesture in this project is sequence of 60 observations.

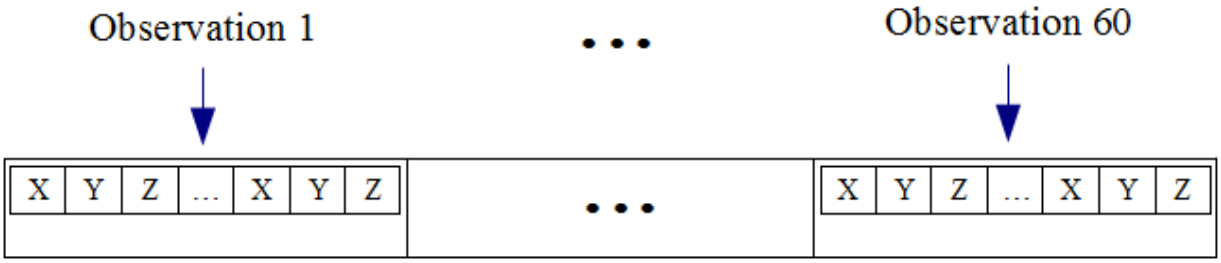


Figure 4.3: An *ObservationSequence* consists of 60 *Observation*s

The use of *Observation*s, *ObservationSequence*s and the double array make the code easily extendible to other representations.

When using coordinates, before being stored in any of the data structures, they are scaled to the origin in the way described in 3.2.2.1 to provide translation invariance. It was felt that scale invariance wasn't a primary concern and left for future work.

4.2 Static Gesture Recognition

As previously stated, the static gesture recognition was a preamble to dynamic gesture recognition and so a standalone program was created using quaternions only. Upon pressing a button to capture the pose, 10 examples of the pose (10 *Observation*s) were extracted from the 30 frames fired during the one second recording. The NN algorithm was used, specifically the 1-NN, which was being executed on the frame at 4 times a second against recorded poses. This meant the poses were being classified automatically but this is not the ideal solution to

real time gesture recognition because the frame could have captured the start or end of the pose, that would not be considered a pose by the user. The NN algorithm required an implementation of 'distance' between gestures. For coordinates, the Euclidean distance over all the joints coordinates could have been used. According to [74], the entire Euclidean distance over quaternions could not be used. The *Observation* must first be split into it's quaternions for each bone orientation. As each quaternion has two equal representations, the distance between the quaternion is defined as the minimum of:

- $dist(qc, qr)$
- $dist(qc*, qr)$
- $dist(qc, qr*)$
- $dist(qc*, qr*)$

where qc is the quaternion from the *Observation* to classify, qr is a quaternion from one of the recorded *Observation*s, $qc*$ is qc's equal representation, $qr*$ is qr's equal representation and $dist$ a function that calculates the Euclidean distance between the quaternions. After the minimum distance had been calculated for each quaternion in the *Observation* to classify, they were summed to give the total distance between Quaternions. It was later discovered that Euclidean distance is not appropriate to use when working with quaternions that represent rotations [21].This is discussed in more detail in 4.3.1. Despite this, simple gestures were recognised during user testing - see chapter 6.

4.3 Dynamic Gesture Recognition

Upon starting to implement dynamic gestures, the main piece of software was created and the standalone static gesture recogniser left behind (see 4.2).

4.3.1 Architecture of the Gesture Recogniser

The gesture recogniser is the core class of the software. It manages all the tasks concerning transformation of data, as well as training and testing the machine learning algorithm. It stores:

- **Continuous training data**: Multiple observations sequences separated into gesture classes.
- **Discrete training data**: The discrete representations of the continuous observation sequences, again separated into gesture classes.
- **A k-means clustering model**: Used to transform the continuous observation sequences into discrete observation sequences.
- **Hidden Markov Models**: One for each gesture class, used collectively to classify incoming observation sequences (a new gesture).

Training the Gesture Recogniser

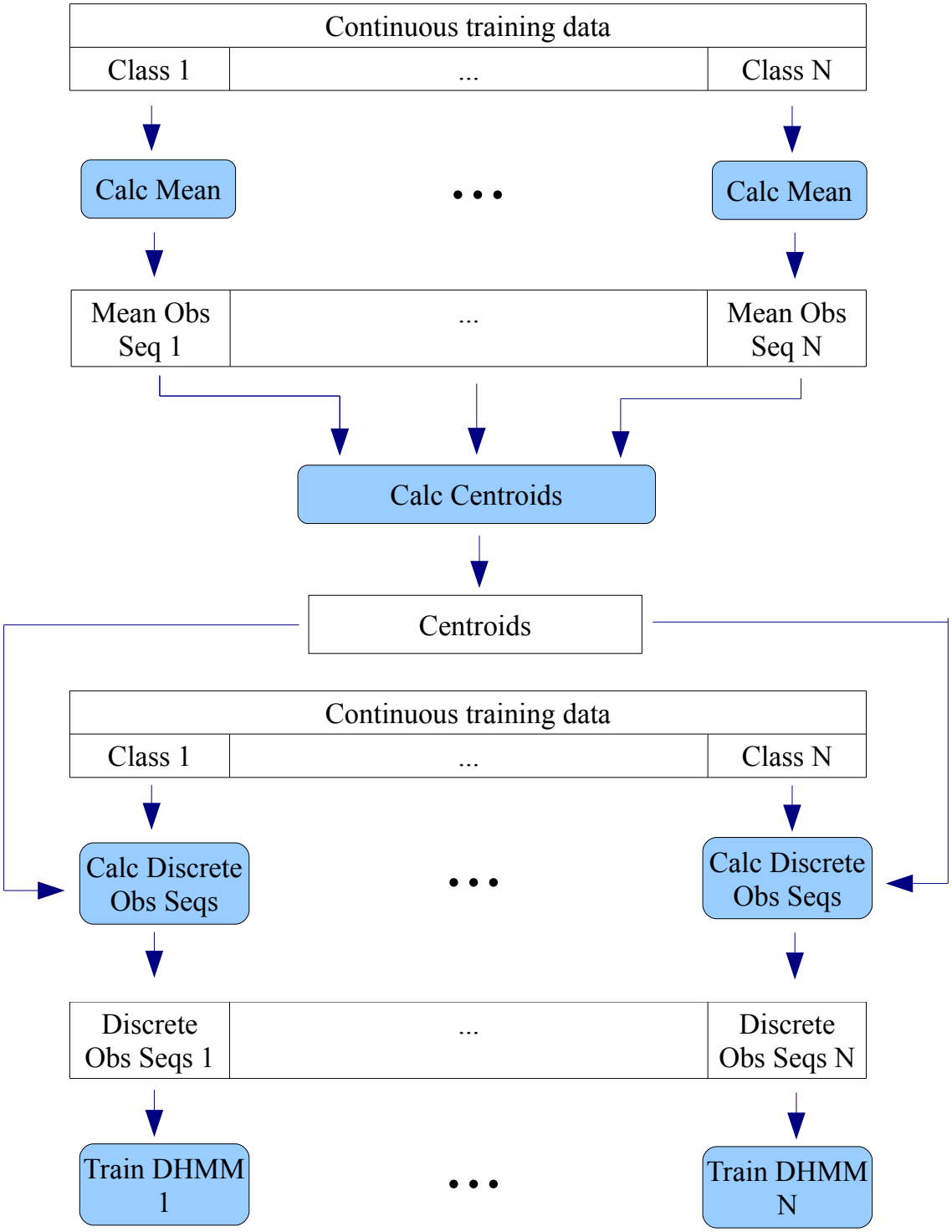


Figure 4.4: The process of training the gesture recogniser

Class 1 to Class N are the continuous observations sequences, separated into their various gesture classes. Calc Mean calculates the mean observation sequence, a representative of the set

of observation sequences, to lessen the input for the next stage of the training. This was to help maintain suitability to real time gesture recognition. Calc Centroids groups the similar observations in the mean observation sequence and labels the group centres as DHMM output symbols using *k*-means clustering. See section 3.3.3 for details.Calc Discrete Obs Seqs transforms the observations sequences into discrete observation sequences using the centroids found for each class in the previous stage. Again, see section 3.3.3. Train DHMM trains each gesture class's DHMM with the discrete observation sequences calculated in the last step using the Baum Welch learning algorithm.

Calculating Centroids As part of transforming continuous data to discrete data, *k*-means clustering was used for vector quantisation (see 3.3.3). It is a commonly written algorithm so source code from [110] was used. It is a simple, iterative algorithm:

```
kmeans(mean observation sequence, K)
{
    centroids = every K observations from the mean observation sequence
    while the centroids are still moving
    {
        for each observation in the mean observation sequence
        {
           assign the observation to it's nearest centroid (by
           comparing observations)
        }
        for each centroid
        {
             recalculate it's position (by averaging the
             observations assigned to it)
        }
    }
}
```

It can be seen that there are two parts to calculating centroid: comparing observations (initial discussion in section 4.2) and averaging observations. The reason for using source code instead of a library call was because the algorithm requires a different implementation for calculating centroids for quaternions and coordinates.

Comparing Individual Observations This is used during *k*-means clustering to assign observations to centroids (see section 4.3.1).

Coordinates When the observations to be compared consist of a group of joint position Cartesian coordinates, the distance can be measured with Euclidean distance. This can either be calculated by summing the distance between each joint in the observation

$$distance(O1, O2) = \sum_{i=1}^{20} dist(O1_i - O2_i)$$

where $O1$ is the first observation, $O2$ is the second observation, i it the joint within the observation, *distance* is the total distance between onservations and *dist* is the Euclidean distance measure between joints.

$$dist(J1,J2) = \sqrt{(J1_X - J2_X)^2 + (J1_Y - J2_Y)^2 + (J1_Z - J2_Z)^2}$$

where $J1$ and $J2$ are joints and X, Y and Z are coordinates of each joint.

Due to using Euclidean distance, the distances across all joint positions can be calculated in one pass

$$distance(O1,O2) = \sqrt{\sum_{i=1}^{20}(O1_{i_X} - O2_{i_X})^2 + (O1_{i_Y} - O2_{i_Y})^2 + (O1_{i_Z} - O2_{i_Z})^2}$$

Quaternions As seen in section static gestures bit, Euclidean distance cannot be used when comparing quaternions that represent rotations. The material on this topic is similar, yet conflicting, but a common response is that the distance should be a measure of the angle between the quaternions [17][13]. Calculating the angle was the source of conflicting opinions. [21] says to use the value of the dot product and [6] says to use the absolute value of the dot product.

$$\arccos(q1 \cdot q2)$$

and

$$\arccos(|q1 \cdot q2|)$$

respectively where $q1$ is the first quaternion, $q2$ is the second quaternion and $\cdot$ is the dot product.

However, [9] says one cannot treat quaternion angles in the same way as vector angles and the result produced from the formula above would not represent the angle between them. [31] suggests a similar formula:

$$2 \times \arccos(|q1 \cdot q2|)$$

[32] suggests

$$2 \times \arccos(|(q1/q2) \cdot 1|)$$

or the norm of the difference, if only an approximation is needed.

$$2(1 - |q1 \cdot q2|)$$

No information to confirm any of these could be found in academic papers or the literature.

[2] and [16] both suggested converting the quaternions to an alternative representation, axis/angle, comparing and then converting back if necessary. Given that this was the only agreed upon solution, this was implemented. Axis/angle is exactly what it states: the angle of rotation around an arbitrary vector. Quaternions are converted to axis/angle using

$$angle = 2 \times \arccos(q_W)$$
$$x = q_X / \sqrt{(1 - (q_W)^2)}$$
$$y = q_Y / \sqrt{(1 - (q_W)^2)}$$
$$z = q_Z / \sqrt{(1 - (q_W)^2)}$$

where q is the quaternion being converted, $angle$ is the resulting angle of rotation in the axis/angle representation and (x, y, z) is the resulting vector to rotate around in the axis/angle representation. This formula was implemented and the axis/angles compared by summing the angle between the vectors and the difference between the original angles:

$$\arccos(v1 \cdot v2) + |a1 - a2|$$

This is an inaccurate measure and it remains incorrectly implemented in the project. It is wrong for two reasons. Firstly, the fact that quaternions have two equal representations, and therefore axis/angle also have two equal representations, was not taken into account. Secondly, even if this was taken into account through multiple comparisons like in 4.2, the distance measure would still be incorrect. Consider two quaternions representing the same orientation. The first quaternion is a rotation of 120 around the Y-axis, appropriately encoded in quaternion form. The second is a rotation of 240 around the negative Y-axis. Converting to axis/angle, the same values are obtained. Comparing the equal representations should give distance measure of zero. However, based on the constructed formula, the difference between the angles is 120°and the angle between the vectors is 180°. This totals to a difference of 300°, very far from the 0°it should be. The proposed solution is to get rid of the axis/angle representation completely. Instead, apply each quaternion to an arbitrary vector (i.e. the Y-axis) and then calculate the angle between them using the dot product. In this way, the two equal representations are eliminated by producing the same orientation when applied to the vector.

Averaging Observations This is used during k-means clustering to recalculate the centroid positions (see section 4.3.1). It is also used in computing the mean observation sequence.

Coordinates The usual mean average can be applied when working with coordinates and, like computing distances (see section 4.3.1), the set does not have to be split into the coordinates of the positions of individual joints.

Quaternions The usual mean average cannot be applied to quaternions to obtain an average quaternion. The first alternative implementation tried was from the Unity community [5], which alters the usual mean average to begin averaging with the identity matrix [103]. It states that this method only works if the quaternions are "relatively close to each other" which is imprecise. After testing, it was evident that this method was incorrect. Further investigation found that the usual mean average cannot be applied in any form to quaternions that represent rotation. According to [128], if the usual mean average is applied to rotation quaternions "the result is not a general rotation quaternion: moreover, there is no sensible geometric interpretation of such a procedure". This author goes on to explain that the effect of multiple quaternion rotations can be obtained by multiplying the quaternions together. The order in which they are multiplied affects the final rotation. The fact that rotational multiplication is not commutative is the reason that quaternions cannot be averaged. Therefore, quaternions can be transformed into logarithmic space where the multiplication becomes addition that is commutative. The newly represented quaternions can be averaged using the usual mean average before being transformed back to quaternion space. [4] expresses the same concept as does [3] [93] [54], here the latter implement it using matrices. For the real time application of the project, the number of conversions between quaternions and matrices would be large and time consuming. Less computational power for averaging would be used by representing the gesture as a matrix to begin with, but as was already discussed in section 3.2.4.2 matrices have been disregarded due to their 16 elements decreasing the speed of training the machine learning algorithm. The incorrect method remains in the project and implementing one of the proposed solutions is left as further work.

4.3.2 Training the HMMs

Coding a HMM would have been a huge task so the source code for a DHMM from [50] was used. Not simply making a call to a library function enabled parameters to be changed. Initially, the prior probabilities meant that a gesture would always have to begin in the same state. The Baum Welch learning algorithm was unable to make any changes to the prior probabilities because the single probability of '1' provided no flexibility. Therefore, as suggested by [105] ("random or uniform initial estimates of priors and transmission parameters is adequate for giving useful re-estimates of these parameters in almost all cases."), the prior probability distribution was set uniformly. The source code already provided a uniform distribution over the transition probabilities, but as discussed in section 3.3.3 this should be refined to a forward HMM for a gesture recognition application. Also, for any state, the set of allowable transitions were limited to the following two states instead of all the following states to more closely mimic a natural gesture. [105] also said "...for the emission parameters, experience has shown that good initial estimates are helpful in the discrete symbol case, and are essential in the continuous distribution case." The emission probabilities were left as a uniform distribution because a more useful initial estimate was unknown. The source code also involves scaling [105] and logarithms [86] as optimisations and as a way to avoid numerical inconsistencies when working with the extremely small conditional probabilities (for example when dividing by the total probabilities in the Baum Welch algorithm, rounding errors are avoided).

4.3.3 Recognising Gestures

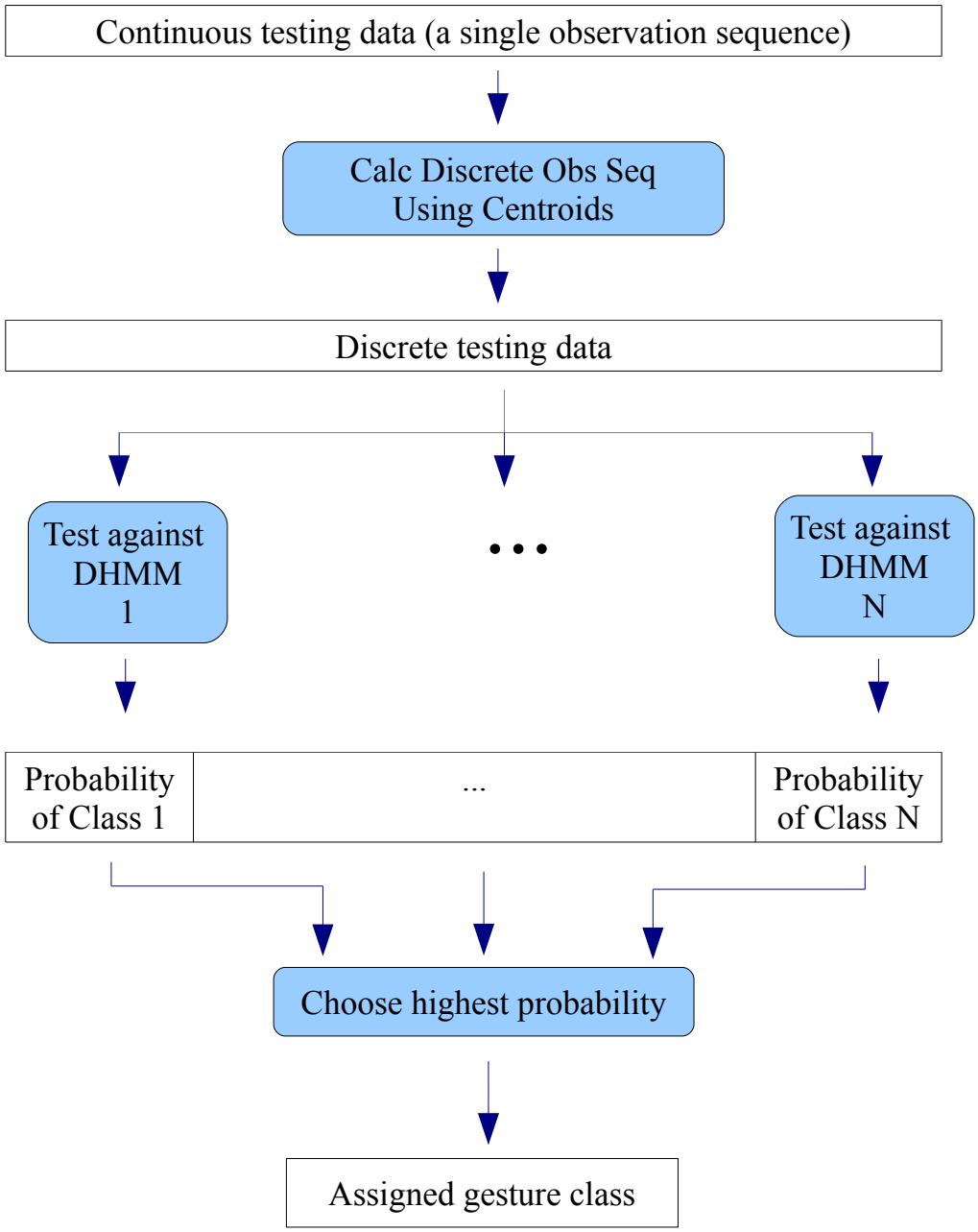


Figure 4.5: The process of classifying a gesture

When a gesture to be recognised is passed to the gesture recogniser, the continuous observation sequence is first converted to a discrete observation sequence using the k-means clustering centroids already trained. Each observation is assigned the label of the centroid to which it is closet. Now the observation sequence is assessed against each trained DHMM (for each gesture class) using the forward algorithm to see how likely it is that he gesture originated from that model. The observation sequence is assigned the gesture of the model that produced the highest probability.

Recognising gestures from non-gestures

As HMMs produce a probability alongside the assigned gesture, it has the potential to recognise gestures from non gestures. If the probability is not high enough then it isn't a gesture. A simple numeric threshold from [64] was implemented by summing the probability of each of the gestures the HMM trained on, multiplying the sum by 2 and divided by the number of training gestures. Testing proved that the threshold was too high for new gestures to be classified. [105] wrote that "A simple threshold usually does not work well". The reason for this problem stems from the fact that the probabilistic value is not important, only that it is higher than the other values [80]. Real time gesture recognition could not be implemented as a result of not being able to distinguish gestures from non gestures. The software created requires the user to press a button to capture the gesture they want recognised. A solution to non-gestures can be found in section along with real time gesture recognition in section 7.1.1.

Chapter 5

Results

Below show screen shots and short explanations of the piece of software created to aid with the gesture recognition process.

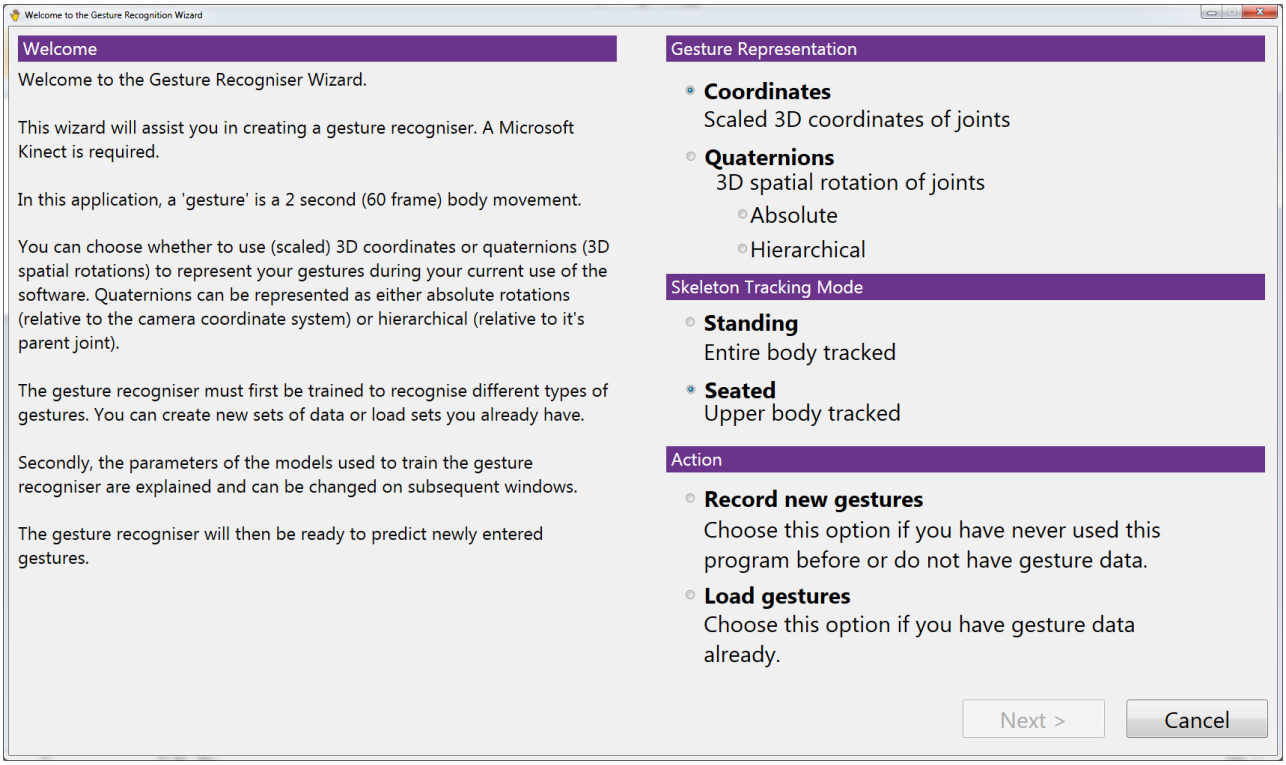


Figure 5.1: Start window

The wizard starts (figure 5.1) with options of choosing between different gesture representations, skeletal tracking modes and whether the user wants to add or load gestures.

The user can add (figure 5.2) new gestures, supplying labels to the data and switching between the current gesture they are recording. The gestures can be discarded if they did not record correctly and can be saved to files.

The user can also choose to load data (figure 5.3) from a file.

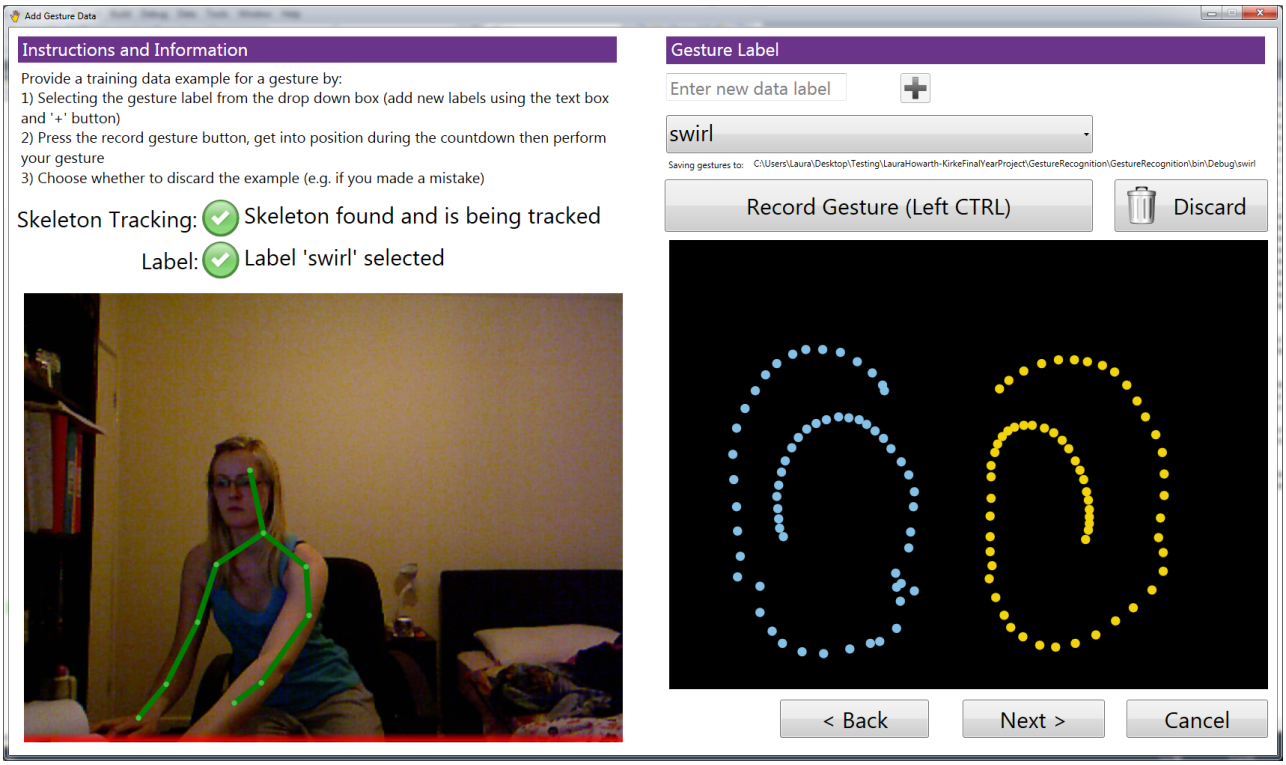


Figure 5.2: Add gestures window

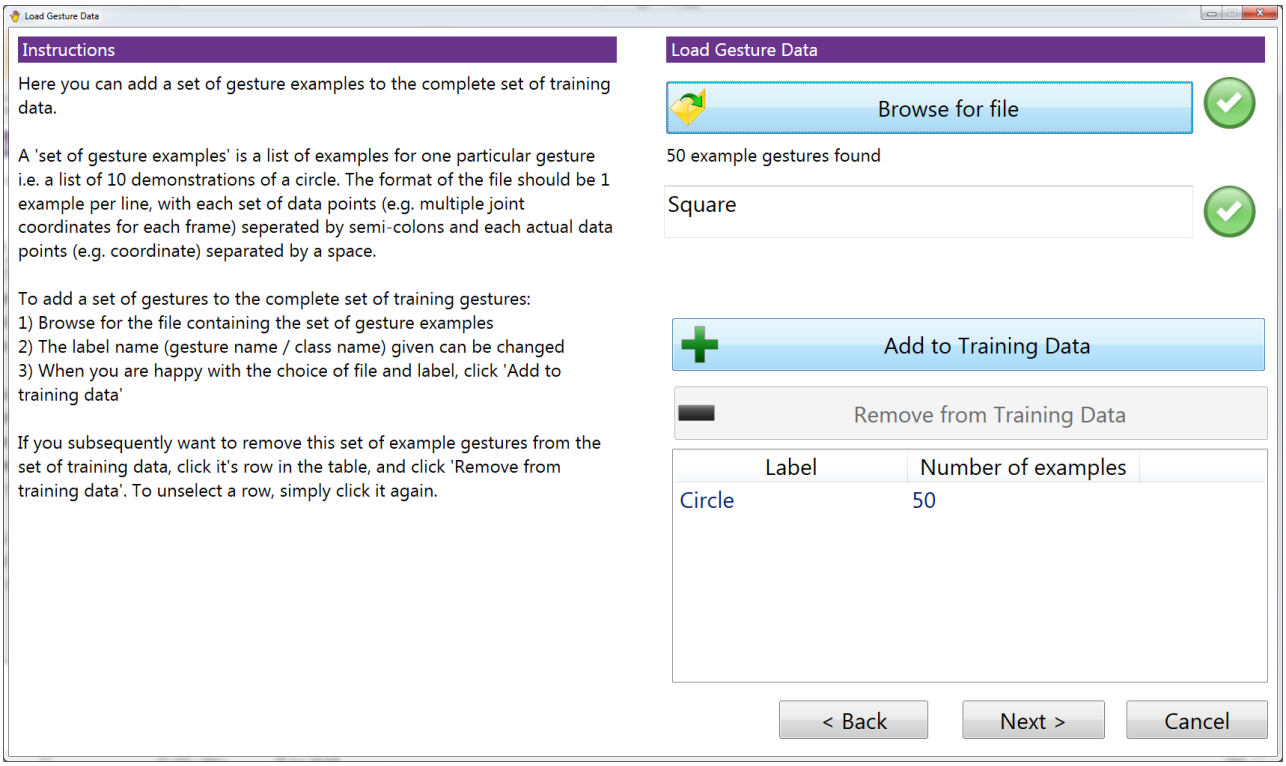


Figure 5.3: Load gestures window

The user can change the number of HMM states and symbols, as well as the amount of iterations as a stopping criteria for the machine learning algorithms (figure 5.4).

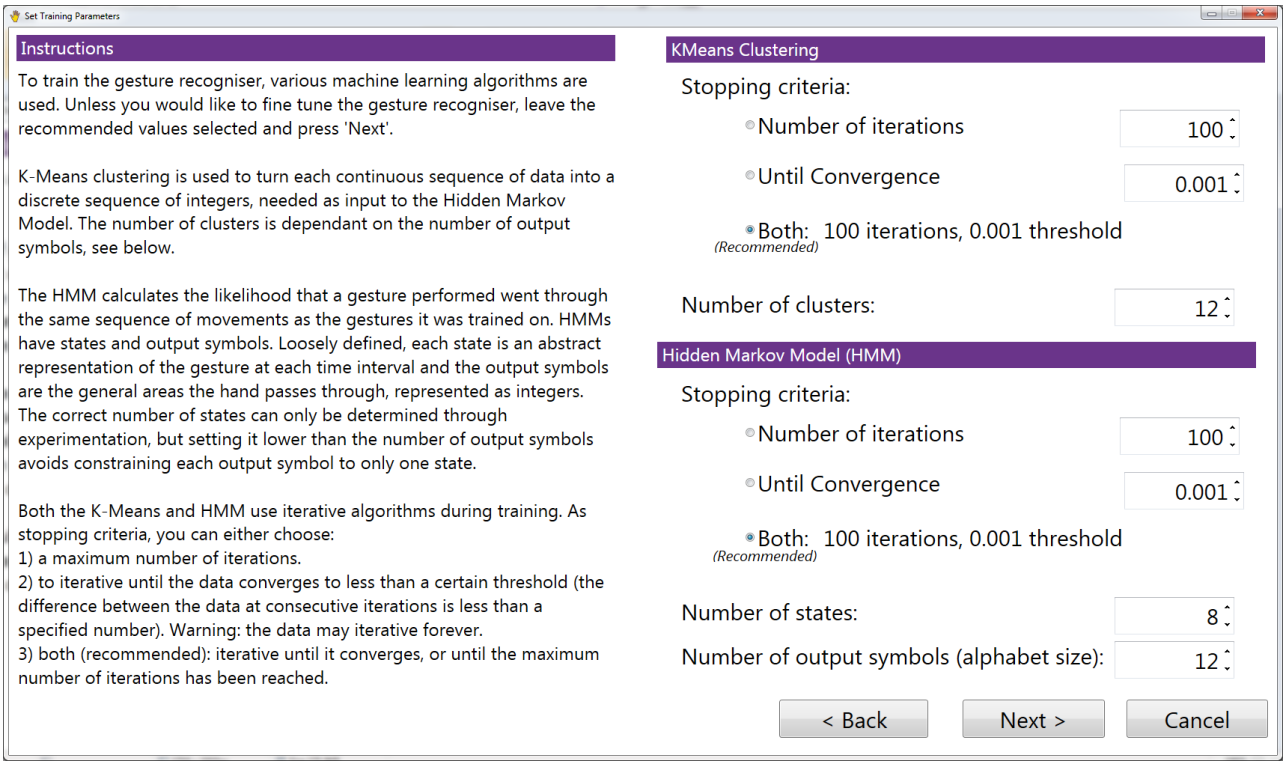


Figure 5.4: Gesture recogniser parameters window

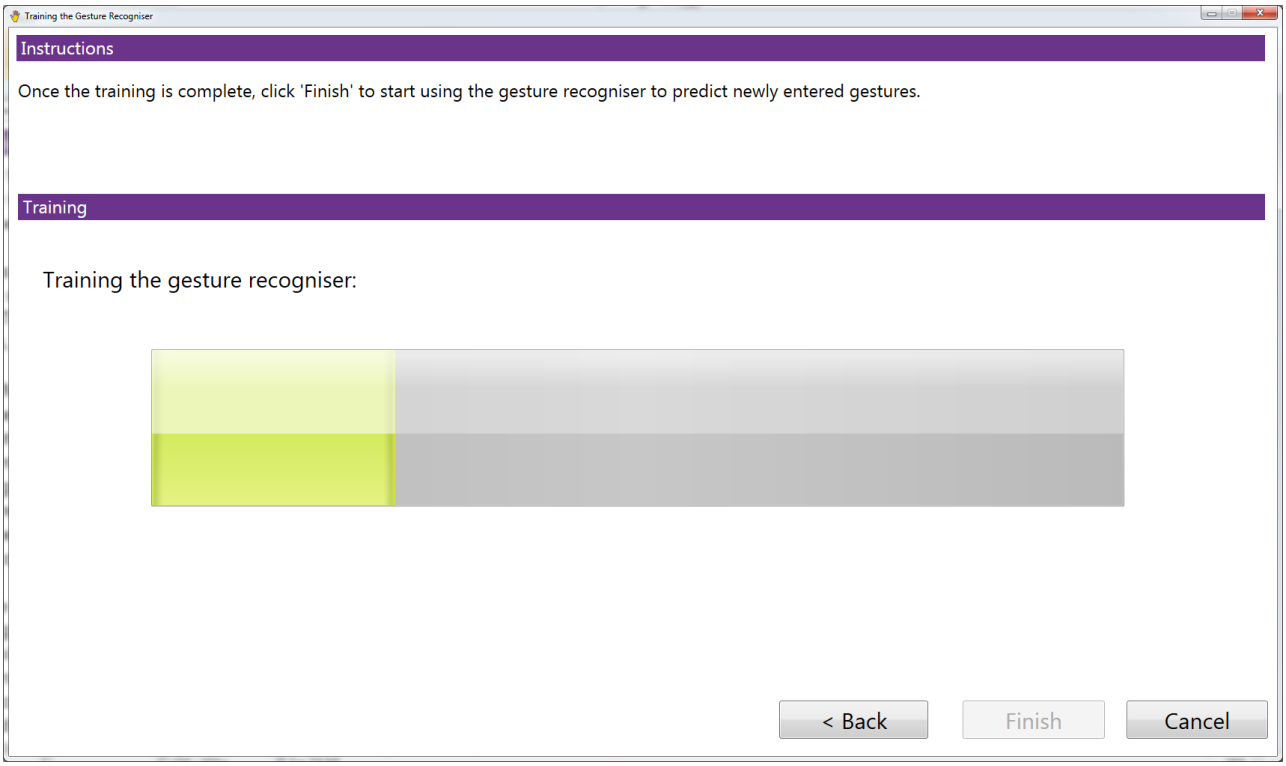


Figure 5.5: Gesture training progress window

The user is updating on the progress the gesture recogniser has made on training the model (figure 5.5)

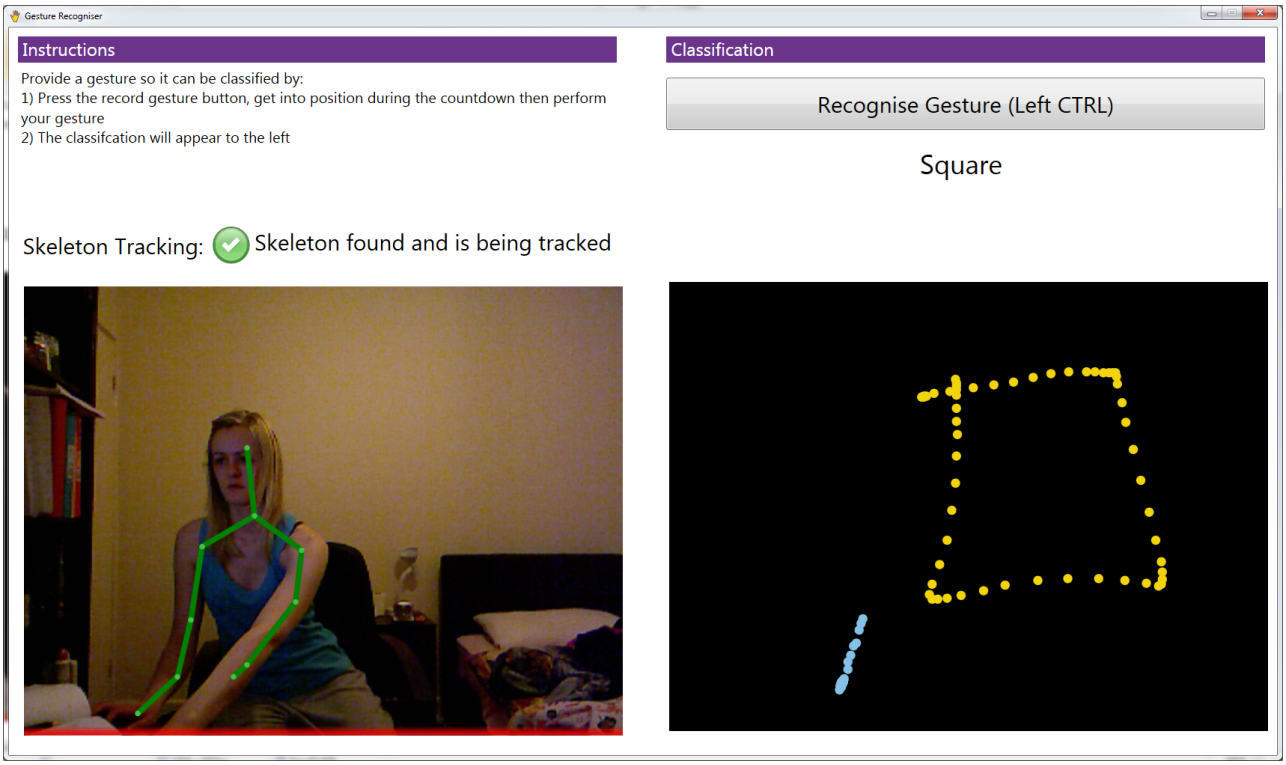


Figure 5.6: Gesture recognition window

Finally the user can input new gestures and classify them against the trained model.

Chapter 6

Testing and Evaluation

The testing focuses on gesture recognition rather than the piece of software made to support the project aims.

6.1 Gesture Recognition

To test the accuracy of gesture recognition, 1 adult male and 1 adult female recorded 20 samples of different gestures.

Firstly, to emphasize the important of the number of states and symbols in the DHMM, 3 similar gestures, that are more likely to be misclassified, were recorded: a circle, square and triangle. The gestures were performed in the same location, with the right hand, starting from approximately shoulder height. The participants were told to try and maintain the size and speed of the gesture. These gestures are shown diagrammatically in A.2.

All gesture data for each class was collated, and a random split took to select training data and testing data. This was performed 10 times for each set of states and symbols and averaged to produce more reliable results.

The first set of states and symbols considering was when they are both equal. To achieve a good rate of classifications, setting them equal is not a good idea. The reason being is that we do not know if the incoming gesture contains any of the gestures we are comparing it against. Setting the number of symbols equal to the number of states will automatically constrain the observations to always pass through a state that belongs to the target gesture, however distantly related to the gesture it is. However, it was thought that this could be useful in determining the approximate amount of states needed to model the data set.

Figure 6.1 shows the total accuracy for coordinates when using an equal number of HMM states and symbols.

The overall accuracy was also tested for absolute quaternions 6.2. This has confirmed suspicions that the comparison and averaging of quaternions is, in fact, incorrect. Logically, quaternions should out performed coordinates due to the fact that they provide all the requirements of a gesture representation. Due to this, no more tests were conducted involving quaternions.

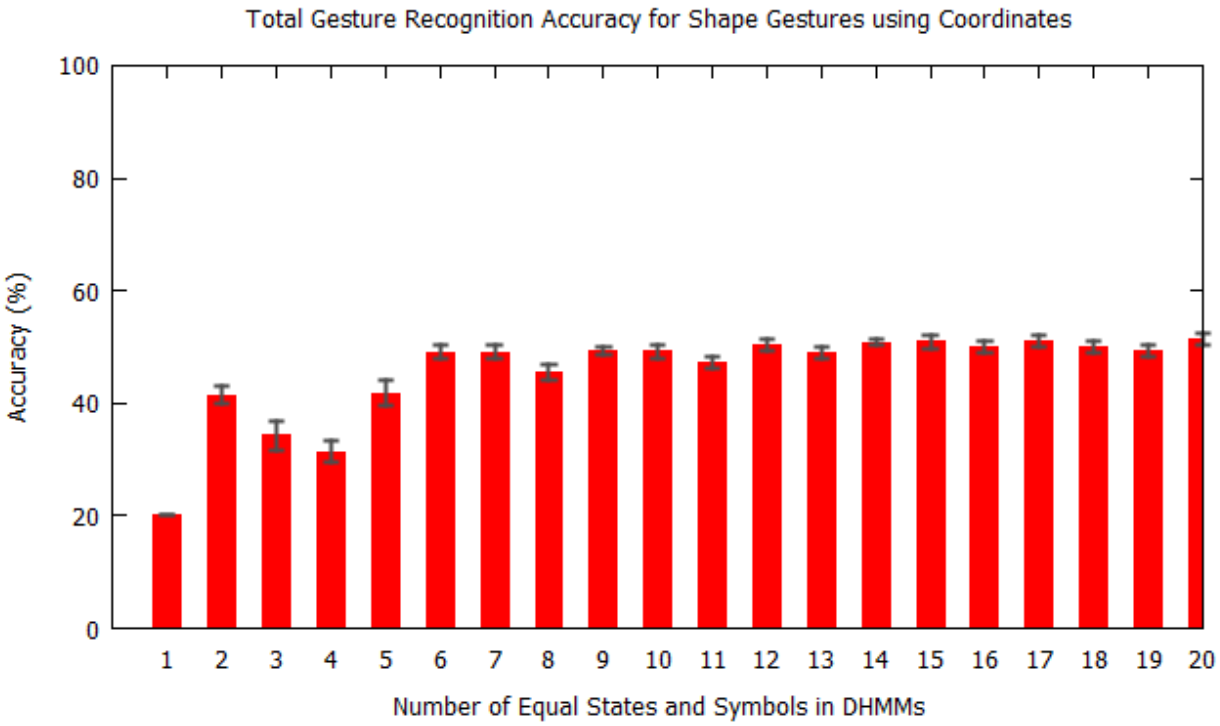


Figure 6.1: Equal States and Symbols for Coordinates

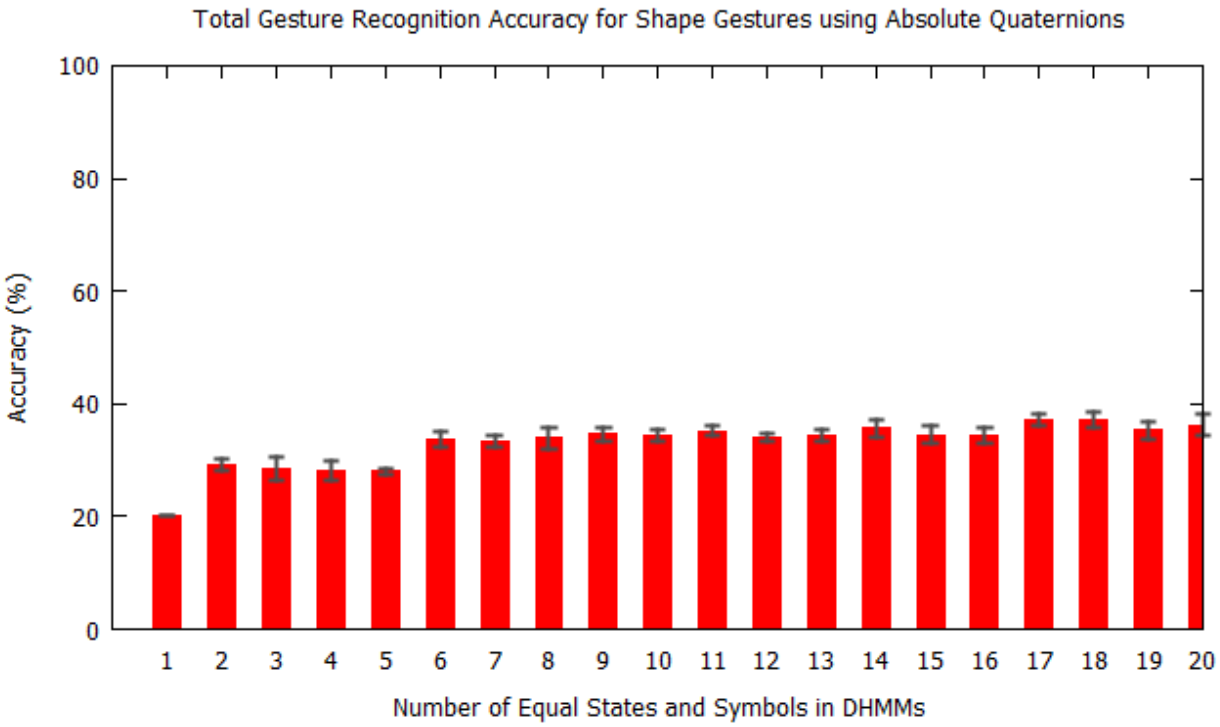


Figure 6.2: Equal States and Symbols for Absolute Quaternions

It can be seen that the accuracy for coordinates in 6.1 is slightly increased at 14 with a lower error. The symbols were increased to see if it would improve the accuracy.

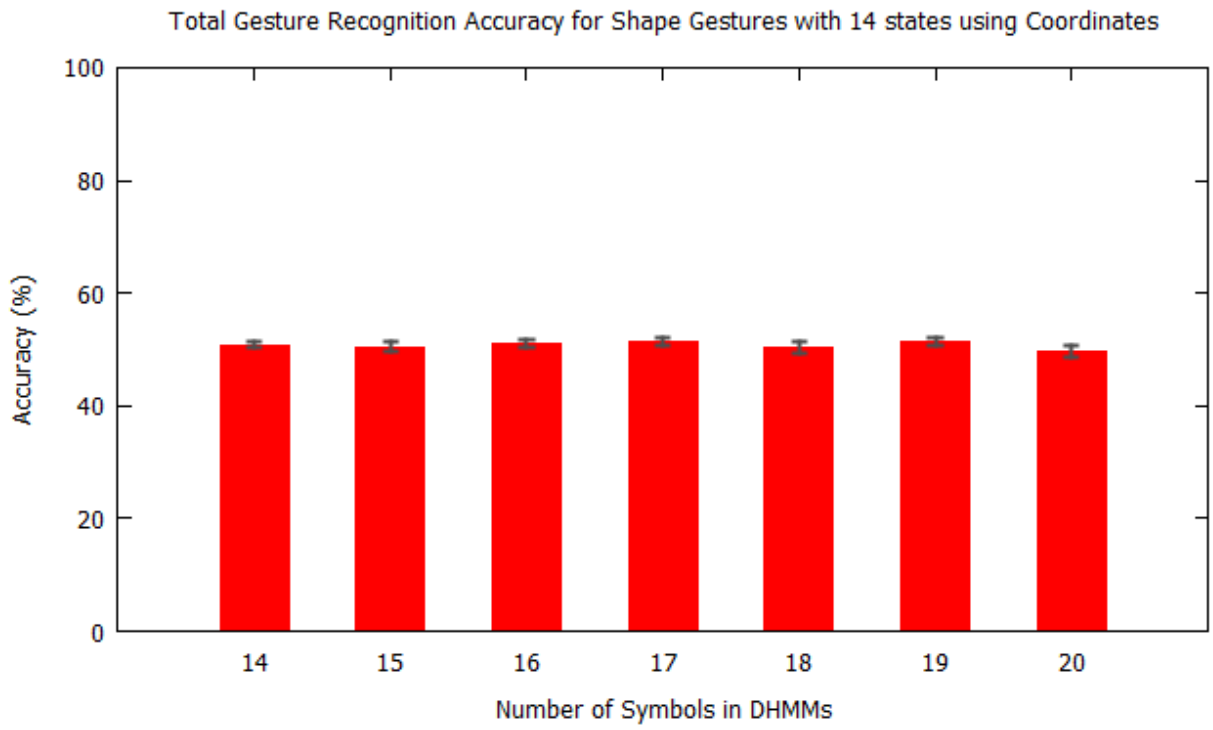


Figure 6.3: 14 states and increasing symbols

As can be seen from 6.3, increasing the number of symbols made little difference. It is believed that more training data from a wider range of people is needed to confirm this concept, which is widely enforced in the literature.

The requirements of a gesture representation also need to be tested. Firstly, as no algorithm was implemented to account for the scale of user's, this will not be tested. For translation invariance, 20 gestures of the action of throwing a ball forwards (like a basketball) were recorded in four different areas of the camera space, shown in 6.4. The same gesture of throwing the ball forwards was recording again but facing approximately 40°to the left and again about 40°to the right to allow testing for the illusion of orientation.

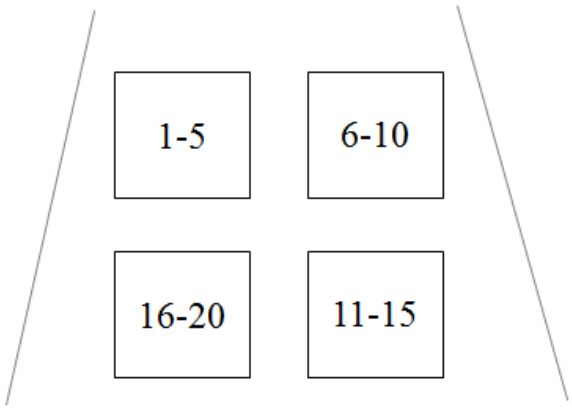


Figure 6.4: The number of gestures recorded in different areas of the play space

Due to time constraints, these tests were unable to be performed, but the tests are outlined here For translation invariance, collate the left, right and forward data. In iterations, choose the testing data in a leave one out fashion; choose one area of the room as testing data and the other three as training data. It is expected that the classification should be relatively high because coordinates are being scaled to compensate for their positional aspects 3.2.2.1 and despite the fact that the same gesture is being performed, the other gesture classes are facing in different directions which make them a different gesture all together

For orientation testing, gestures from the different areas can be mixed and a normal test performed to see if the recognises the different orientations.

Overall, it is felt that the testing for this project is unsatisfactory, especially for this type of application. Time constraints and lack of access to testers were the main difficulty.

6.2 Software and User Interface

Unit tests were performed on the piece of software to test the core functionality of adding and recording gestures, changing algorithm parameters, training and recognising gestures. Unit tests were also performed on data input, such that the user could not record the gesture if they were already recording, the parameters (such as the maximum number of iterations of the machine learning algorithm) boundaries were maintained and the same gesture label not being used for two different gesture classes. The minimum and maximum distance of the user from the Kinect was enforced and the user was not allowed to progress through the wizard until relevant information had been gathered.

Chapter 7

Conclusions

Gesture recognition is becoming increasingly popular in every day life since it's re-introduction in the gaming industry. Due to advances in hardware, devices such as the Kinect are being utilised in unconventional industries such as surgical operations.

Various approaches to gesture recognition have been described from it's roots in the literature through to modern day. This informed the choice of input device, gesture representation and machine learning algorithm. Quaternions were found to be the most appropriate data to use for representing gestures. Unfortunately, problems emerged with comparing (section 4.3.1) and averaging (section 4.3.1) quaternions during implementation. As the errors remain in the application, testing to confirm their suitability over coordinates could not be performed. As mentioned, the testing is believed to be inadequate. However, the aim of the project was to find the most suitable approach to gesture recognition and it is felt that this was achieved through research. Until the problems associated with quaternions are fixed, much of the testing would have revolved around coordinates, which has been seen to be covered extensively already in the literary review.

Both static and dynamic gesture recognition were achieved through the discovery that certain machine learning algorithms are more suited to each type of gesture than others. Improvements to the work include implementing either logistic regression or a neural network for static gesture recognition because they are more appropriate than NN; they were omitted in favour of proceeding to dynamic gesture recognition. Testing showed the important of choosing an appropriate number of states and symbols for the DHMMs, depending on the problem and data. It was also discovered that certain machine learning algorithms are more suited to discrete or continuous data than others. It was decided to use DHMMs even though the chosen gesture representation (quaternions/coordinates) was continuous because they are simpler and the same technique of vector quantisation was seen in the literary review. However, all the problems relating to quaternions arose from this area, so if the project were to be repeated, CHMMs would be used.

A polished piece of software was created to aid in the gesture recognition process, from adding gestures, training the model with the ability to change parameters to recognising gestures. Most importantly, the code and interface were created in such a way that future gesture representations and machine learning algorithms can easily be incorporated. Two of the initial functional requirements were not fulfilled due to time constraints and prioritisation, but both of which can

easily be implemented. First is loading a trained model and the other is loaded gestures to be recognised (though the back end for this was written but not connected to the interface).

It is believed that the level of testing is unsatisfactory for this project. This is mainly due to the lack of range of testers providing training data. The accuracy of gesture recognition was not as high as was expected

7.1 Further Work

Before taking the application further, the problems with averaging and comparing quaternions should be fixed, possibly with the solutions outlined in section 4.3.1).

One of the initial aims of the project was to recognise gestures in real time. This was not achieved and hence neither was mapping the gestures to game inputs as a proof of concept. This was partly due to the fact that gestures could not be distinguished from non-gestures. The solutions to these problems are proposed as future work.

7.1.1 Recognising gestures from non-gestures

Lee and Kim [81] suggest creating a threshold HMM to be used in conjunction with all dedicated HMMs for each gesture class. The threshold model is a collation of all the states from all the dedicated HMMs in an ergodic structure. In this sense, it is a weak model for all trained gestures as the likelihood is smaller than that of the dedicated models (which have a more accurate forward probability distribution). For a gesture to be classified, the likelihood resulting from the threshold model can be used as a numeric threshold for the highest likelihood produced from the dedicated HMMs. Pseudo code for this process is shown below.

```
recogniseGesture(ObservationSequence gesture)
{
    threshold = hmms[thresholdHMM].evaluate(gesture);

    foreach gesture class, c
    {
        likelihood = hmms[c].evaluate(gesture);
        if (likelihood > highestLikelihood)
        {
            highestLikelihood = likelihood
            predicatedClass = c;
        }
    }

    if (likelihood > threshold)
       return predictedClass
    else
       return No gesture found
}
```

The disadvantage of the threshold model is that it can have a large number of states that make training and recognising time consuming and memory intensive, but a suggested solution is to merge similar states.

7.1.2 Real-time gesture recognition

By implementing the threshold model discussed in 7.1.1, real time gesture recognition can also be solved. From a continuous stream of input data, patterns have to be matched with all possible segments of the stream. This is called "gesture spotting" and is regarded in the literature as a difficult problem, mainly because the transition between gestures could be wrongly assumed to be meaningful. Gestures can be spotted by finding their start and end points in the input stream via a gesture spotting network (GSN). At each time step, a window of previous observations are selected from the input stream. The probability of this observation sequence is calculated by all the dedicated HMMs and the threshold model using the Viterbi algorithm. When the probability from one of the dedicated models rises above that of the threshold model, a candidate end point (CEP) has been found.

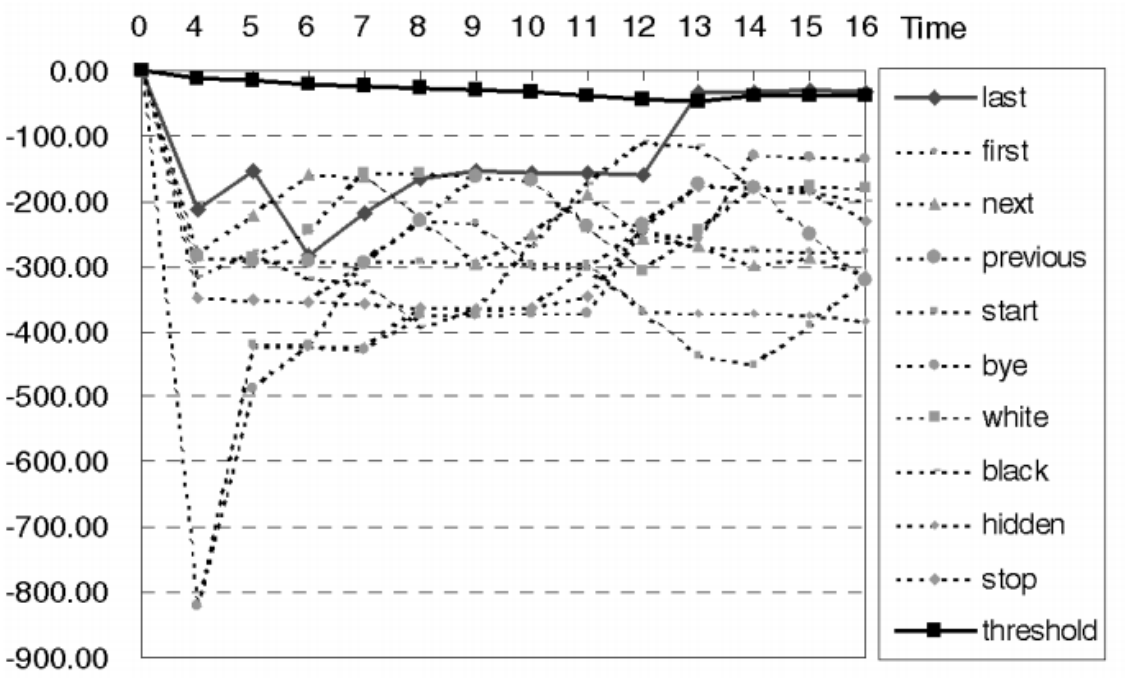


Figure 7.1: A CEP is identified at time 13 [81]

From here, the start point can be found by backtracking the Viterbi algorithm. In general, the gesture will have several CEPs and gestures may be nested. Therefore, the gesture from the above process should only be reported once the last CEP of the current gesture has been found and determined not to be nested within another gesture. After a given amount of time either:

- All dedicated HMM values will fall below the threshold model, in which case the current gesture can be reported.
- Another CEP from another gesture will been found.

- If the start point of the found CEP precedes the CEP of original gesture, then the original gesture is part of a bigger gesture.
- Otherwise, the original gesture is a separate gesture to the one belonging to the latest CEP.

This is summarised in figure 7.2.

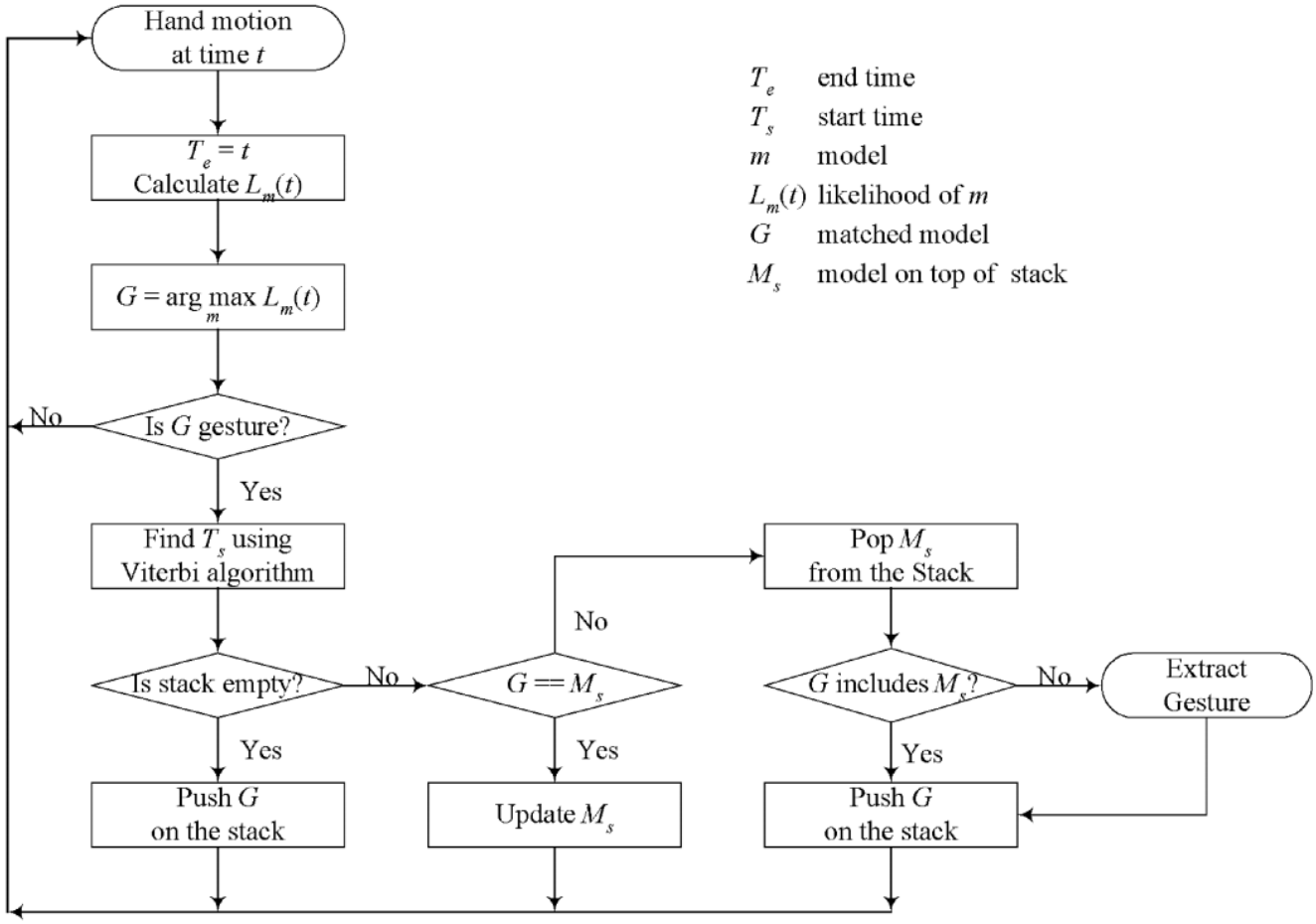


Figure 7.2: The process of gesture spotting through end point detection [81]

Other solutions to real time gesture recognition are expressed in [61][102][115] but none that also integrate non-gesture recognition.

7.2 Summary

Hopefully this paper has made a positive contribution to using quaternions as an alternative representation for gestures, by outlining their advantages, problems and pitfalls. It is hoped that the the project will be useful as a learning tool and that it will be taken further with the suggested solutions.

References

[1] 8-4 dynamic time warping. http://mirlab.org/jang/books/dcpr/dpDtw.asp?title=8-4%20Dynamic%20Time%20Warping, 30/04/2013.

[2] Angle between two quaternions. http://www.ogre3d.org/forums/viewtopic.php?t=44704, 30/04/2013.

[3] average of multiple quaternions? http://stackoverflow.com/questions/12374087/average-of-multiple-quaternions, 30/04/2013.

[4] Averaging quaternions. http://math.stackexchange.com/questions/61146/averaging-quaternions, 30/04/2013.

[5] Averaging quaternions and vectors. http://wiki.unity3d.com/index.php/Averaging_Quaternions_and_Vectors, 30/04/2013.

[6] Comparing quaternion rotations. http://www.gamedev.net/topic/542263-comparing-quaternion-rotations/, 30/04/2013.

[7] Computer graphics. realism and performace. http://itee.uq.edu.au/~comp3201/OGL3_7.pdf, 30/04/2013.

[8] data glove. http://www.webopedia.com/TERM/D/data_glove.html, 30/04/2013.

[9] Dot (inner) product of 2 quaternions. http://www.gamedev.net/topic/140433-dot-inner-product-of-2-quaternions/, 30/04/2013.

[10] Gesture recognition. http://www.webopedia.com/TERM/G/gesture_recognition.html, 30/04/2013.

[11] Gesturetek. http://www.gesturetek.com/, 30/04/2013.

[12] Hmm. http://abitofalchemy.blogspot.co.uk/2012/11/wavelet-based-statistical-signal.html, 30/04/2013.

[13] How do i compare quaternions? http://answers.unity3d.com/questions/288338/how-do-i-compare-quaternions.html, 30/04/2013.

[14] Kinectnui. http://kinectnui.codeplex.com/, 30/04/2013.

[15] Math - quaternions. http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/, 30/04/2013.

[16] Maths - quaternion to axisangle. http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToAngle/index.htm, 30/04/2013.

[17] Module geometry tools. https://kforge.ros.org/Sushi/www/pr2_python/pr2_python.geometry_tools-module.html, 30/04/2013.

[18] Multinomial logistic regression. http://www.strath.ac.uk/aer/materials/5furtherquantitativeresearchdesignandanalysis/unit6/multinomiallogisticregression/, 30/04/2013.

[19] Multinomial logistic regression models. http://sites.stat.psu.edu/~jls/stat544/lectures/lec19.pdf, 30/04/2013.

[20] The multinomial logit model. http://data.princeton.edu/wws509/notes/c6s2.html, 30/04/2013.

[21] Nearest neighbours using quaternions. http://stackoverflow.com/questions/4917400/nearest-neighbours-using-quaternions, 30/04/2013.

[22] Ogre euler angle class. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Euler+Angle+Class, 30/04/2013.

[23] Opengl rotation and translation done correctly. http://stackoverflow.com/questions/14084071/opengl-rotation-and-translation-done-correctly, 30/04/2013.

[24] Opengl:tutorials:using quaternions to represent rotation. http://content.gpwiki.org/index.php/OpenGL:Tutorials:Using_Quaternions_to_represent_rotation, 30/04/2013.

[25] Openni. http://www.openni.org/, 30/04/2013.

[26] Orientation. http://dictionary.reference.com/browse/orientation?s=t, 30/04/2013.

[27] Pointgrab. http://www.pointgrab.com/, 30/04/2013.

[28] The quadcopter get its orientation from sensors. http://theboredengineers.com/2012/09/the-quadcopter-get-its-orientation-from-sensors/, 30/04/2013.

[29] The quadcopter get its orientation from sensors. Whatarethedifferencesbetweenagyroscope,accelerometerandmagnetometer?, 30/04/2013.

[30] Reveal powerpoint presentation. http://reveal.rs.af.cm/, 30/04/2013.

[31] Rotation difference between two quaternions? http://www.gamedev.net/topic/423462-rotation-difference-between-two-quaternions/, 30/04/2013.

[32] Small note on quaternion distance metrics. http://fgiesen.wordpress.com/2013/01/07/small-note-on-quaternion-distance-metrics/, 30/04/2013.

[33] Socially assistive robotics: An nsf expedition in computing. http://www.robotshelpingkids.org/, 30/04/2013.

[34] Softkinetic. http://www.softkinetic.com/, 30/04/2013.

[35] Using quaternion to perform 3d rotations. http://www.cprogramming.com/tutorial/3d/quaternions.html, 30/04/2013.

[36] What is a stereo camera? http://www.vmresource.com/camera/cameras-general.htm, 30/04/2013.

[37] Gy. Dorko B. Caputo1 and H. Niemann. Combining color and shape information for appearance-based object recognition using ultrametric spin glass-markov random fields. *Proceeding SVM '02 Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines*, (pp. 97-111), 2002.

[38] Lucas Bang. Computer sign language recognition. http://www.youtube.com/watch?v=hWA8wWbrBn8, 30/04/2013.

[39] Allyssa Barnes. Sign language recognition. http://www.youtube.com/watch?v=ftuQvh5cjpU, 30/04/2013.

[40] Barry Bazzell. Kinect gesture recognition technology helping surgeons. http://articles.businessinsider.com/2011-03-21/tech/29971945_1_kinect-sunnybrook-imaging, 30/04/2013.

[41] F. Bettens and T. Todoroff. Real-time dtw-based gesture recognition external object for max/msp and puredata. *Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference*, July 2009.

[42] Prof: J. Bilmes. Review of dynamic time warping. http://melodi.ee.washington.edu/~bilmes/ee516/lecs/lec9_scribe.pdf, 30/04/2013.

[43] Gavin Brown. The comp61011 guide to machine learning. University of Manchester lecture notes, 30/04/2013.

[44] Tim Carstens. What is a hidden markov model. http://intoverflow.wordpress.com/2008/05/27/what-is-a-hidden-markov-model/, 30/04/2013.

[45] Nick Collins. Sign language program converts hand movements into text. http://www.telegraph.co.uk/science/science-news/9134827/Sign-language-program-converts-hand-movements-into-text.html, 30/04/2013.

[46] Microsoft Corporation. Human interface guidelines kinect for windows. (v1.5.0), 2012.

[47] C. Couvreur. Hidden markov models and their mixtures. 1996.

[48] P. Marsh O'Shaughnessy D. Morris, P. Collett. *Gestures, Their Origins and Distribution. London: Cape*. Jonathan Cape, 1979.

[49] Science Daily. Future surgeons may use robotic nurse, 'gesture recognition'. http://www.sciencedaily.com/releases/2011/02/110203152548.htm, 30/04/2013.

[50] Cesar de Souza. Hidden markov models in C#. http://www.codeproject.com/Articles/69647/Hidden-Markov-Models-in-C, 30/04/2013.

[51] N. Papamarkos E. Stergiopoulou. Hand gesture recognition using a neural network shape tting technique. *Engineering Applications of Artificial Intelligence*, (Volume 22 Issue 8, pp. 1141-1158), Dec 2009.

[52] The Engineer. Gesture-recognition system could reduce surgical delays. http://www.theengineer.co.uk/medical-and-healthcare/news/

gesture-recognition-system-could-reduce-surgical-delays/1015164.article, 30/04/2013.

[53] Seyda Ertekin. k-nn. http://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring lecture-notes/MIT15_097S12_lec06.pdf, 30/04/2013.

[54] J. Crassidis F. Markley, Y. Cheng and Y. Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, (Vol. 30, No. 4, pp. 1193-1197), 2007.

[55] Chunsheng Fang. From dynamic time warping (dtw) to hidden markov model (hmm). Mar 2009.

[56] J. Fasola. Robot exercise instructor: A socially assistive robot system to monitor and encourage physical exercise for the elderly. *RO-MAN, 2010 IEEE*, (pp. 416-421), Sept 2010.

[57] M. Reinders G. Holt and E. Hendriks. Multi-dimensional dynamic time warping for gesture recognition. *Thirteenth annual conference of the Advanced School for Computing and Imaging*, June 2007.

[58] Nick Gillian. Gesture recognition toolkit. http://www.nickgillian.com/software/grt, 30/04/2013.

[59] Christopher Grant. Kinect hacks: American sign language recognition. http://www.joystiq.com/2010/12/20/kinect-hacks-american-sign-language-recognition/, 30/04/2013.

[60] J. Guo. Hand gesture recognition and interaction with 3d stereo camera. Nov 2011.

[61] P. Lukowiczb G. Trstera H. Junkera, O. Amfta. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, (Vol 41, pp. 2010-2024), June 2008.

[62] U. Bellugi H. Poizner and V. Lutes-Driscoll. Perception of american sign language in the dynamic point light displays. *MEDLINE*, (7: pp.g430-440), Aprrl 1981.

[63] Gidget Haddah. Euclidean symmetrics in mathematics. http://carlossicoli.free.fr/H/Haddad_G.-Euclidean_Symmetries_in_Mathematics-White_Word_Publications(2012).pdf, 30/04/2013.

[64] Jonathan C. Hall. How to do gesture recognition with kinect using hidden markov models (hmms). http://www.creativedistraction.com/demos/gesture-recognition-kinect-with-hidden-markov-models-hmms/, 30/04/2013.

[65] M. Hall. Correlation-based feature selection for machine learning. April 1999.

[66] Berthold K.P. Horn. Some notes on unit quaternions and rotation. http://people.csail.mit.edu/bkph/articles/Quaternions.pdf, 30/04/2013.

[67] Luis Ibanez. Tutorial on quaternions part i, 30/04/2013.

[68] N. Ibraheem and Rafiqul Khan. Vision based gesture recognition using neural networks

approaches: A review. *International Journal of human Computer Interaction (IJHCI)*, (Volume (3), Issue (1)), 2012.

[69] IGN. Eyetoy: Play review. http://uk.ign.com/games/eyetoy-play/ps2-570957, 30/04/2013.

[70] IGN. Kinect preview. http://uk.ign.com/games/kinect/xbox-360-14357198, 30/04/2013.

[71] IGN. Playstation move preview. http://uk.ign.com/games/playstation-move-159404/ps3-14318622, 30/04/2013.

[72] InTecj. Hidden markov models, theory and applications. http://pure.rhul.ac.uk/portal/files/1446635/Hidden_Markov_Models_Theory_and_Applications.pdf, 30/04/2013.

[73] J. Ohya J. Yamato and K. Ishii. Recognizing human action in time-sequential images. *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (pp. 379-385), Jun 1992.

[74] H.C. Jian. Gesture recognition using windowed dynamic time warping. (pp. 11-14), 2010.

[75] Mohammed Waleed Kadous. Dynamic time warping. http://www.cse.unsw.edu.au/~waleed/phd/html/node38.html, 30/04/2013.

[76] Eamonn Keogh. Everything you know about dynamic time warping is wrong. https://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CDIQFjAA&url=http%3A%2F%2Fwww.cs.ucr.edu%2F~eamonn%2FDTW_myths.ppt&ei=CopgUc3LLIqX1AWG1IGABg&usg=AFQjCNEwzo3rHc_QEZzRJIcV2XXR9OblXg&sig2=rkU9RKoxp_FbsyEZfO9iww&bvm=bv.44770516,d.d2k, 30/04/2013.

[77] JA Kogan and D. Margoliash. Automated recognition of bird song elements from continuous recordings using dynamic time warping and hidden markov models: a comparative study. Apr 1998.

[78] Daniel Kohlsdorf. Building a gesture recognizer - part 2 hidden markov models. http://daniel-at-world.blogspot.co.uk/2013/01/building-gesture-recognizer-part-2.html, 30/04/2013.

[79] Frederic Lardinois. Ukrainian students develop gloves that translate sign language into speech. http://techcrunch.com/2012/07/09/enable-talk-imagine-cup/, 30/04/2013.

[80] Christopher Lee. Hidden markov model intro. http://vimeo.com/7175217, 30/04/2013.

[81] H. Lee and J. Kim. An hmm-based threshold model approach for gesture recognition. *IEEE Transactions Pattern Analysis and Machine Intelligence*, (pp. 961-973), Oct 1999.

[82] S. Liwicki and M. Everingham. Automatic recognition of fingerspelled words in british sign language.

[83] S. Lung. Sign language recognition with kinect. Sep 2011.

[84] Brendon Lyons.

[85] G. Dominguez M. Rayes and S. Escalera. Feature weighting in dynamic time warping for gesture recognition in depth data. *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on Computer Vision*, (pp. 1182-1188), Nov 2011.

[86] T. Mann. Numerically stable hidden markov model. Feb 2006.

[87] Richard Marks. Eyetoy, innovation and beyond. http://blog.us.playstation.com/2010/11/03/eyetoy-innovation-and-beyond/, 30/04/2013.

[88] MathWorks. Supervised learning (machine learning) workflow and algorithms. http://www.mathworks.co.uk/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html, 30/04/2013.

[89] Microsoft. Joint orientation. http://msdn.microsoft.com/en-us/library/hh973073.aspx, 30/04/2013.

[90] Microsoft. Kinect for windows sensor components and specifications. http://msdn.microsoft.com/en-us/library/jj131033.aspx, 30/04/2013.

[91] Microsoft. Tracking modes (seated and default). http://msdn.microsoft.com/en-us/library/hh973077.aspx, 30/04/2013.

[92] Microsoft. Wizards. http://msdn.microsoft.com/en-us/library/windows/desktop/aa511260.aspx, 30/04/2013.

[93] M. MOAKHER. Means and averaging in the group of rotations. *SIAM Journal on Matrix Analysis and Applications*, (Volume 24 Issue 1, pp. 1-17), 2002.

[94] Nima Moshtagh. Minimum volume enclosing ellipsoid. http://www.mathworks.com/matlabcentral/fileexchange/9542, 30/04/2013.

[95] K. Murakami and H. Taguchi. Gesture recognition using recurrent neual networks. *Proceeding CHI '91 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (pp. 237-242), 1991.

[96] G. R. S. Murthy and R. S. Jadon. A review of vision based hand gestures recognition. *International Journal of Information Technology and Knowledge Management*, (Volume 2, No. 2, pp. 405-410), July-Dec 2009.

[97] NASA. Aircraft pitch motion. http://www.grc.nasa.gov/WWW/K-12/airplane/pitch.html, 30/04/2013.

[98] G. Nejat and M. Ficocelli. Can i be of assistance? the intelligence behind an assistive robot. *IEEE International Conference on Robotics and Automation Pasadena*, (2008), May 2008.

[99] BBC News. Sign language 'turned into text' by aberdeen scientists. http://www.bbc.co.uk/news/uk-scotland-north-east-orkney-shetland-17297489, 30/04/2013.

[100] N. Aggarwal P. Garg and S. Sofat. Vision based hand gesture recognition. *World Academy of Science, Engineering and Technolog*, (49 2009), 2009.

[101] N. Aggarwal P. Garg and S. Sofat. Vision based hand gesture recognition. *World Academy of Science, Engineering and Technology*, (25 2009), 2009.

[102] H. Junker M. Stager G. Troster A. Atrash P. Lukowicz, J. Ward and T. Starner. Recognizing workshop activity using body worn microphones and accelerometers. *Pervasive Computing*, (pp. 37-39), 2004.

[103] Cory Petkovsek. Ogre quaternion and rotation primer. `http://www.ogre3d.org/tikiwiki/Quaternion+and+Rotation+Primer`, 30/04/2013.

[104] K. Wong R. Nag and F. Fallside. Script recognition using hidden markov models. *ICASSP 86*, 1986.

[105] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, (Nol 77, No 2, pp. 257-287), 1989.

[106] Marc Ramsey. Faq: What's the difference between a gyro and an accelerometer? do i need both? `http://diydrones.com/profiles/blogs/faq-whats-the-difference`, 30/04/2013.

[107] Kumar Ravinder. Comparison of hmm and dtw for isolated word recognition system of punjabi language. `http://link.springer.com/chapter/10.1007%2F978-3-642-16687-7_35`, 30/04/2013.

[108] Math Open Reference. Eccentricity an ellipse. `http://www.mathopenref.com/ellipseeccentricity.html`, 30/04/2013.

[109] Math Open Reference. Major / minor axis of an ellipse. `http://www.mathopenref.com/ellipseaxes.html`, 30/04/2013.

[110] Ceasar Roberto. K-means clustering. `http://crsouza.blogspot.co.uk/2010/10/k-means-clustering.html`, 30/04/2013.

[111] Steven Rodriguez. The twenty wii launch games. `http://www.nintendoworldreport.com/news/12402`, 30/04/2013.

[112] Chris Roper. Playstation eye hands-on. `http://uk.ign.com/articles/2007/09/25/playstation-eye-hands-on/`, 30/04/2013.

[113] Margaret Rouse. Definition gesture recognition. `http://whatis.techtarget.com/definition/gesture-recognition`, 30/04/2013.

[114] 'rymix'. Kinect sdk dynamic time warping (dtw) gesture recognition. `http://kinectdtw.codeplex.com/`, 30/04/2013.

[115] A. Kosmala S. Eickeler and G. Rigoll. Hidden markov model based continuous online gesture recognition. *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference*, (Vol 2, pp. 1206-1208), Aug 1998.

[116] Samsung. Experience the wonder of samsung's new smart tvs. `http://www.samsung.com/us/2012-smart-tv/`, 30/04/2013.

[117] T. Temiz S.Celebi, A.Aydin and T.Arici. Gesture recognition using skeleton data with weighted dynamic time warping.

[118] Andy Schmeder. Hacking a wii remote for full 3d absolute position and orientation sensing. http://cnmat.berkeley.edu/node/7649, 30/04/2013.

[119] Siemens. Gesture recognition in the or. http://www.siemens.com/innovation/apps/pof_microsite/_pof-fall-2011/_html_en/gesture-recognition-in-the-or.html, 30/04/2013.

[120] Tom Simonite. Augmented reality meets gesture recognition. http://www.technologyreview.com/news/425431/augmented-reality-meets-gesture-recognition/, 30/04/2013.

[121] T. Starner and A. Pentland. Visual recognition of american sign language using hidden markov models. *International Workshop on Automatic Face and Gesture Recognition*, (pp. 189-194), 1995.

[122] K. Symeonidis. Hand gesture recognition using neural networks. August 2000.

[123] N. Henze T. Schlomer, B. Poppinga and S. Boll. Gesture recognition with a wii controller. *Proceedings of the 2nd international conference on Tangible and embedded interaction*, 2008.

[124] Firstpost Technology. Ces 2013: Coming soon smart tvs with gesture recognition. http://www.firstpost.com/tech/ces-2013-coming-soon-smart-tvs-with-gesture-recognition-581653.html, 30/04/2013.

[125] Roberto Ulloa. Edge orientation histograms in global and local features. http://roberto.blogs.cultureplex.ca/2012/01/26/edge-orientation-histograms-in-global-and-local-features/, 30/04/2013.

[126] M. Umeda. Recognition of multi-font printed chinese characters. *Proc 6th ICPR*, (pp. 793-796), 1982.

[127] Andrea Vedaldi. Histogram of oriented gradients (hog) features. http://www.vlfeat.org/api/hog.html, 30/04/2013.

[128] L. Vicci. Averages of rotations and orientations in 3-space. Aug 2001.

[129] Steve Wright. Gesture recognition. http://blog.iinet.net.au/gesture-recognition/, 30/04/2013.

[130] Xun Zhang. Sony playstation move - v2.0. http://wiki.etc.cmu.edu/unity3d/index.php/Sony_PlayStation_Move_-_v2.0, 30/04/2013.

[131] X. Zheng and S. Koenig. A project on gesture recognition with neural networks for introduction to articial intelligence classes.

Appendix A

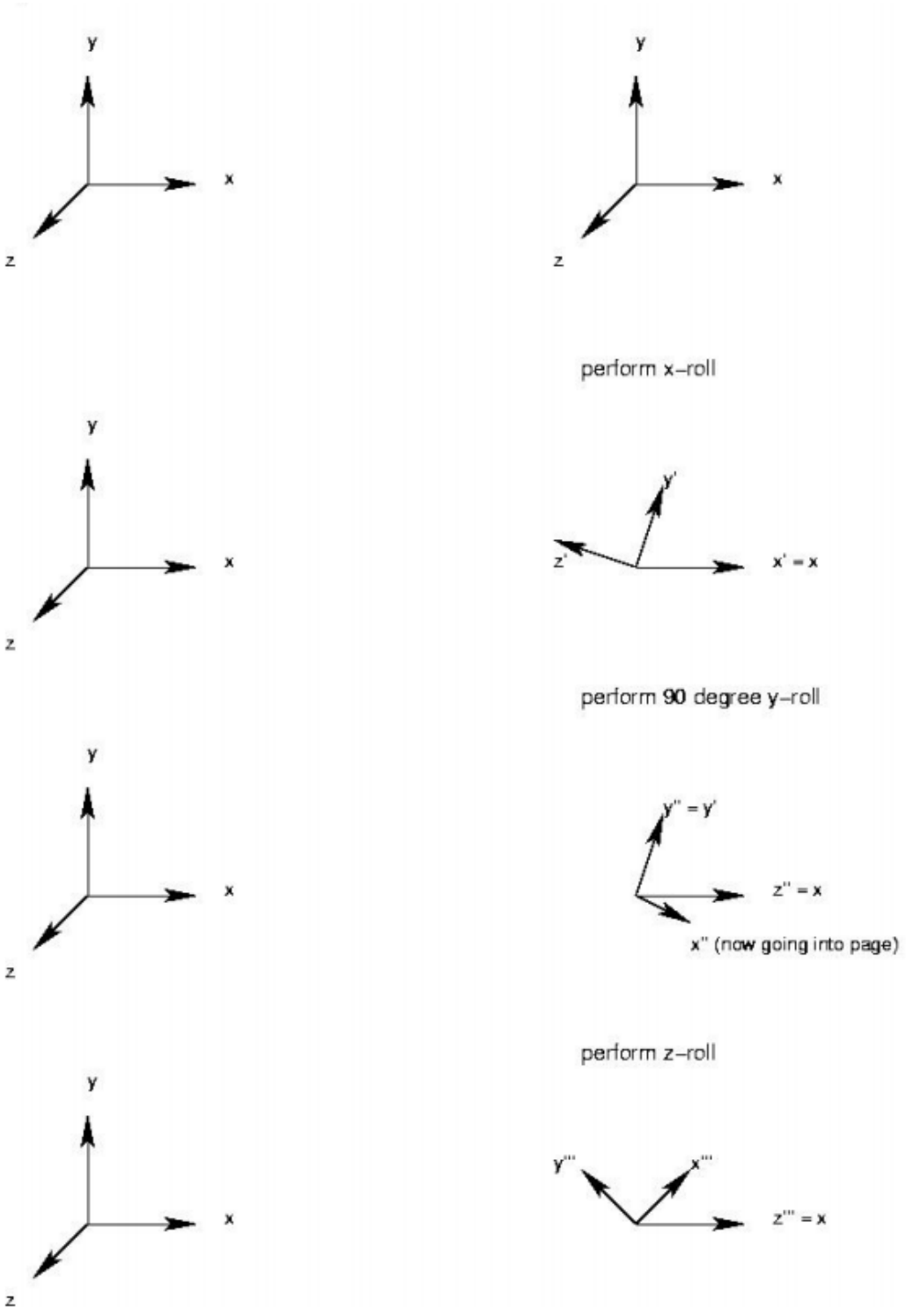


Figure A.1: The steps leading to Gimbal lock. The left hand side is the world coordinate system and the right hand side is the local coordinate system [7]

Scenario: Hand Writing / Drawing

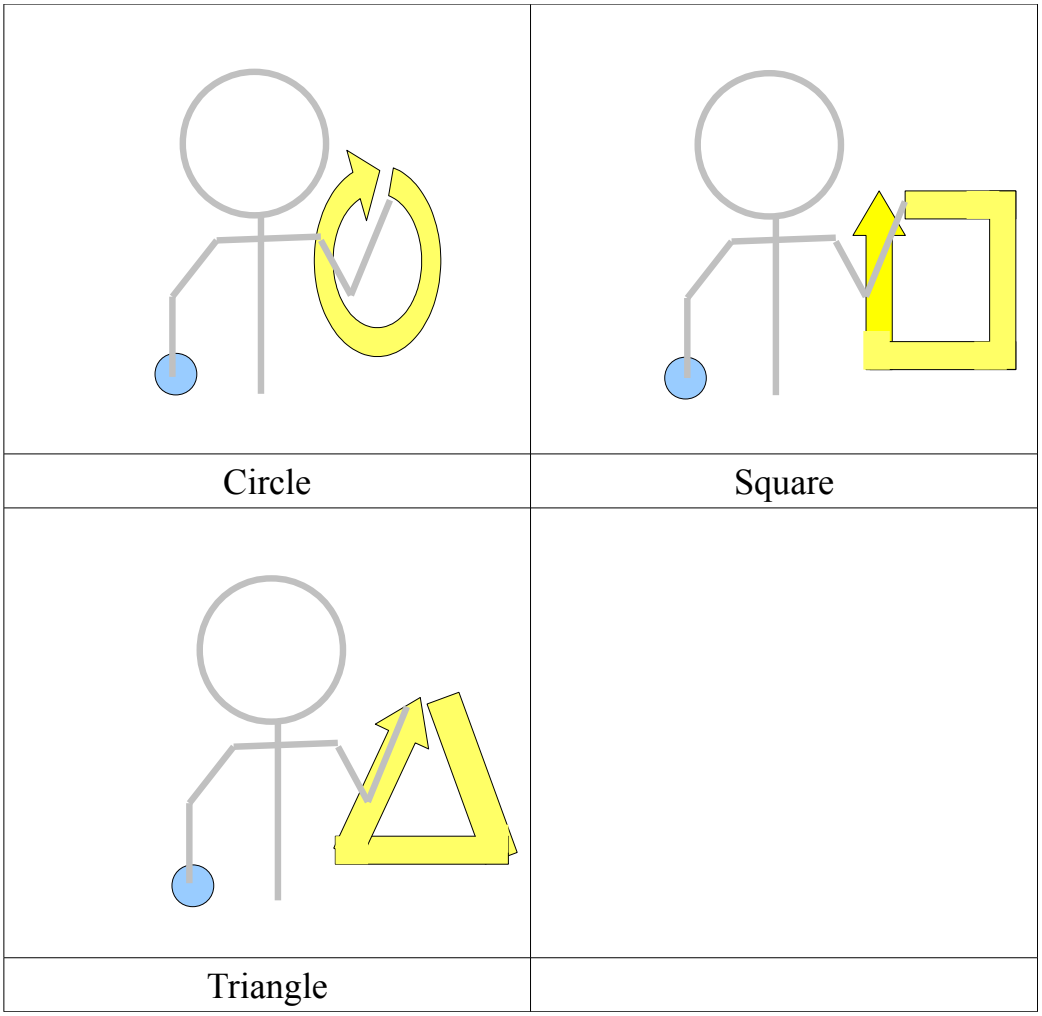


Figure A.2: Similarly shaped gestures used for testing the importance of the number of states and symbols in HMMs