An Evaluation of Depth Camera-Based Hand Pose Recognition for Virtual Reality Systems

by

Andrew William Clark

Submitted to the School of Mathematics, Statistics and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

at the

UNIVERSITY OF KWAZULU-NATAL

December 2018

Author .....
School of Mathematics, Statistics and Computer Science
December 14, 2018

Certified by .....
Prof Deshen Moodley
Associate Professor
Thesis Supervisor

Certified by .....
Mr Anban Pillay
Lecturer
Thesis Supervisor

An Evaluation of Depth Camera-Based Hand Pose Recognition for Virtual Reality Systems

by

Andrew William Clark

Submitted to the School of Mathematics, Statistics and Computer Science
on December 14, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science (Computer Science)

Abstract

Camera-based hand gesture recognition for interaction in virtual reality systems promises to provide a more immersive and less distracting means of input than the usual hand-held controllers. It is unknown if a camera would effectively distinguish hand poses made in a virtual reality environment, due to lack of research in this area. This research explores and measures the effectiveness of static hand pose input with a depth camera, specifically the Leap Motion controller, for user interaction in virtual reality applications. A pose set was derived by analyzing existing gesture taxonomies and Leap Motion controller-based virtual reality applications, and a dataset of these poses was constructed using data captured by twenty-five participants. Experiments on the dataset utilizing three popular machine learning classifiers were not able to classify the poses with a high enough accuracy, primarily due to occlusion issues affecting the input data. Therefore, a significantly smaller subset was empirically derived using a novel algorithm, which utilized a confusion matrix from the machine learning experiments as well as a table of Hamming Distances between poses. This improved the recognition accuracy to above 99%, making this set more suitable for real-world use. It is concluded that while camera-based pose recognition can be reliable on a small set of poses, finger occlusion hinders the use of larger sets. Thus, alternative approaches, such as multiple input cameras, should be explored as a potential solution to the occlusion problem.

Thesis Supervisor: Prof Deshen Moodley
Title: Associate Professor

Thesis Supervisor: Mr Anban Pillay
Title: Lecturer

Acknowledgments

I would like to thank the UKZN/CSIR Meraka Centre for Artificial Intelligence Research for their financial assistance during my Master's years. Furthermore, I would like to extend my gratitude towards my supervisors Anban Pillay and Deshen Moodley for their endless patience and insight into even the most esoteric topics within Computer Science.

Preface

The research contained in this dissertation was completed by the candidate while based in the Discipline of Computer Science, School of Mathematics, Statistics and Computer Science of the College of Agriculture, Engineering and Science, University of KwaZulu-Natal, Westville, South Africa. The research was financially supported by the Center for Artificial Intelligence Research (CAIR).

The contents of this work have not been submitted in any form to another university and, except where the work of others is acknowledged in the text, the results reported are due to investigations by the candidate.

Declaration: Plagiarism

I, Andrew Clark, declare that:

(i) the research reported in this dissertation, except where otherwise indicated or acknowledged, is my original work;

(ii) this dissertation has not been submitted in full or in part for any degree or examination to any other university;

(iii) this dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons;

(iv) this dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) their words have been re-written but the general information attributed to them has been referenced;

b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced;

(v) where I have used material for which publications followed, I have indicated in detail my role in the work;

(vi) this dissertation is primarily a collection of material, prepared by myself, published as journal articles or presented as a poster and oral presentations at conferences. In some cases, additional material has been included;

(vii) this dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Virtual Reality (VR) systems simulate a three-dimensional environment to make users feel as though they are physically immersed in the generated environment. Such an effect is achieved through a VR head-mounted display, which blocks vision of the real world and displays a stereoscopic view of the simulated world. A means to interact with the environment is also provided, using hand-held controllers, cameras, or gloves. The immersion and fluidity of VR interaction over the traditional mouse and keyboard has given rise to VR being used in fields including entertainment, education, rehabilitation, training simulations, and remote operation.

A hand gesture is any movement of the hand to convey an idea or carry a meaning. Gestures can be either static or dynamic [7]. Static gestures involve the hand not changing shape or position over the duration of the gesture, while dynamic gestures involve some form of hand, finger or arm movement. A static hand gesture is referred to as a hand pose. Thus, hand pose recognition is the process of taking a hand pose as input, and outputting the type of pose.

1.1.1 Devices

Depth cameras have proven to be effective input devices for vision-based hand pose recognition, as they make the task of separating foreground from background considerably easier than RGB cameras. Depth cameras that capture infrared are especially useful as they are independent of both skin colour and visual lighting conditions. Of these cameras, the Leap Motion Controller (LMC) and Microsoft Kinect have been popularly used in vision-based hand pose recognition research.
This section provides a detailed description of the primary devices mentioned in this thesis, namely the Oculus Rift DK2 display, Leap Motion Controller, and Microsoft Kinect. A comparison between the Kinect and Leap Motion Controller is also provided.

1.1.1.1 Oculus Rift DK2

The Oculus Rift is a Virtual Reality (VR) Head-Mounted Display (HMD) that renders graphics stereoscopically for an immersive user experience. The specific display used in this research is the Oculus Rift DK2 (Development Kit 2), an older beta version of the consumer version. The Oculus Rift CV1 (Consumer Version 1) has recently become available to consumers, but the use of this newer HMD will not affect the pose recognition measurements made in this research. The DK2 was used in this research to render the VR environment to users.
The Oculus Rift has one display and lens for each eye to achieve a wide field of vision. Each of the two displays render slightly different images to create an illusion of three-dimensional space for the wearer. The Oculus Rift comes with an infrared camera to track the HMDs position, while the HMD itself tracks its own orientation. The orientation and positional updates get sent to the VR rendering application so that it may adjust its camera accordingly, giving users the illusion that they are able to physically look around in the environment.

Figure 1-1: The Oculus Rift DK2 and positional-tracking camera.[1]

1.1.1.2 Leap Motion Controller

The Leap Motion Controller (LMC) is a lightweight and affordable commercial stereoscopic infrared camera that specializes in tracking a user's hands with sub-millimeter accuracy [16]. The device consists of three infrared LEDs and two infrared cameras, to create an interaction space of eight cubic feet [32]. The two cameras give the LMC a stereoscopic view of the user's hands, allowing it to create a depth map. The infrared LEDs and cameras allow the LMC to work under any lighting conditions and skin colour. Thanks to its lightweight nature, the device can also be mounted onto the Oculus Rift, as seen in figure 1-4. The device and the features it captures can be seen in figures 1-2 and 1-3.

The raw image data from the LMC is first processed by the LMC driver software in order to provide intuitive data to the programmer. This allows the programmer to avoid any image processing steps entirely. The processed data is accessed through the Leap Motion API, which provides data in the form of Frames. Each frame represents the data processed by the driver in a single instance of time. Frames contain a list of Hand objects that the controller is able to identify. Each Hand stores data about one particular hand, such as the palm position, palm normal, and a list of Finger objects.

[1]Image source: `https://s3.amazonaws.com/static.oculus.com/website/2014/03/camera_dk2.jpg`

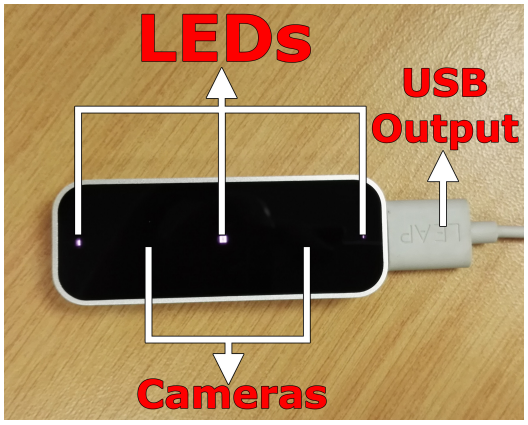


Figure 1-2: The Leap Motion Controller.

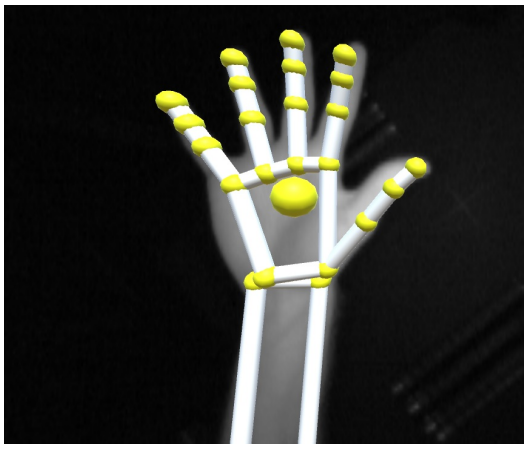


Figure 1-3: Yellow marks represent the features captured by the LMC.

Each Finger object stores data about one finger, such as the positions of each joint and the tip of the finger. These features were used instead of depth images for pose classification. The API also provides limited data correlating to pose recognition. This data includes:

- **Pinch Strength:** A value from 0 to 1 indicating how close the hand is to a pinching pose.
- **Grab Strength:** A value from 0 to 1 indicating how close the hand is to a grabbing pose.
- **Circle Gesture:** Whether a circular motion was made by a finger.
- **Swipe Gesture:** Whether the hand moved in a straight line with fingers extended.
- **Screen Tap Gesture:** Whether a finger tapped or poked forward.
- **Key Tap Gesture:** Whether a finger tapped downwards.

1.1.1.3 Microsoft Kinect

The Microsoft Kinect contains an infrared depth camera, infrared emitter, RGB camera and multiple microphones to detect voice commands as well as the posture of the

Figure 1-4: The LMC mounted onto the Oculus DK2.[2]

human body [42]. The Kinect was initially designed as an additional peripheral to the Xbox 360 gaming console. A newer version of the Kinect has been released, known as the Kinect for Xbox One, which has improved features such as a higher resolution, and is often sold with the newer Xbox One gaming console. The device is usually available at retail gaming or electronics stores. In research, the data captured by the Kinect is extracted through its API.

The Kinect API [41] processes the data received by the Kinect device into frames of data. A notable type of frame is the BodyFrame. This frame contains a collection of all calculated joint positions for all human bodies seen by the Kinect, as seen in Figure 1-6. In terms of hands, the API provides the positions of the tip of the hand, the thumb, the wrist, and the hand itself. The API does not provide positions of individual fingertips.

1.1.1.4 Comparison of the LMC and Kinect

Both the LMC and Kinect are widely available in electronics stores, with the LMC being cheaper than the Kinect. The LMC is also much smaller and lighter than the Kinect, allowing it to be mobile. The bulkier Kinect is expected to be in a fixed position, while the LMC can still operate effectively while mounted on a VR head-mounted display. As discussed in Chapter 1, the API of the Kinect provides limited data pertaining to the hand, while the LMC's API provides detailed hand data

[2]Image source:
`http://riftybusiness.com/wp-content/uploads/2014/08/Leap-Motion-VR.jpg`

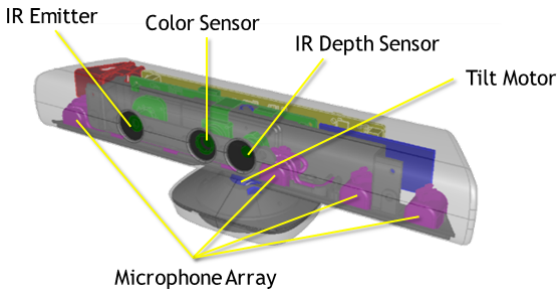


Figure 1-5: The Microsoft Kinect.[3]

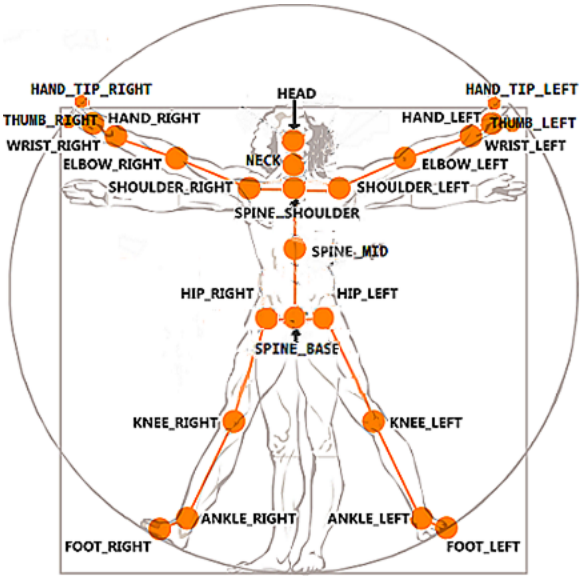


Figure 1-6: The joint features captured by the Kinect.[4]

only. The raw depth image data collected by either device is accessible through their respective APIs, so hand pose recognition via image processing can be performed if necessary. Libraries such as the Nimble SDK used in [39], are available for researchers using the Kinect to detect hand features. The Kinect has a much larger interaction space than the LMC, with a maximum viewing range of approximately 6m, while the LMC has a range of approximately 0.6m. The field of view of the LMC is 150° by 120°, while the Kinect's field of view is approximately 57° horizontally by 43° vertically. The LMC has been shown to have sub-millimeter accuracy [16], while the Kinect is not able to achieve such an accuracy [64].

[3]Image source: `https://i-msdn.sec.s-msft.com/dynimg/IC584396.png`
[4]Image source: `https://i-msdn.sec.s-msft.com/dynimg/IC741371.png`

Table 1.1: Comparison between the LMC and Microsoft Kinect.

	LMC	**Kinect**
Specialization	Hand and finger tracking	Full body tracking
Availability	Retail electronics stores	Retail electronics and games stores
Price	$70 standalone. $75 with mount [33]	$150 Developers' Kinect for Xbox One [40]
Mobility	Small, light, mobile, VR mount	Larger, heavier and stationary
Interaction Space	Shorter range, wider angle.	Much longer range, narrower viewing angle
Versatility	Only hand tracking using infrared.	Body tracking, RGB camera & microphones
Data from API	Detailed hand features only	Body features, minimal hand features

Table 1.1 compares the various aspects of the LMC against the Kinect. While the Kinect is a popular depth camera in vision-based pose recognition, the LMC is a more suitable candidate for pose recognition in a virtual reality environment. The LMC specializes in capturing hand and finger data, while the Kinect is used for full-body tracking, and does not support individual finger tracking by default. Furthermore, the LMC software processes the images fed to it by the camera to provide positional and orientation feature data to the programmer, as opposed to the raw image data from the Kinect. The low weight and small size of the LMC allow it to be mounted to the front of an HMD, allowing the user to always have their hands tracked provided their hands are in their field of view. In contrast, the Kinect has to be left in a stationary position, and if the user were to turn their back to it, their hands would become occluded by their body. Having the LMC attached to the front of an HMD allows the user to have a significantly higher degree of mobility when interacting in virtual reality when compared to the Kinect. The LMC has thus been deemed more suitable for the purposes of this research than the Kinect, due to its compatibility with VR via mounting and its specialization in hand pose tracking.

Many researchers have used the LMC in the field of hand pose recognition, but of these very few have applied it to VR scenarios. Those that used the LMC in VR did not create a system to classify a very wide array of poses using machine learning algorithms. The performance measurements outside of VR made by other researchers may not necessarily be the same as performance measurements in VR. Most non-VR research involving the LMC has the device placed flat on a table, such that the camera

will be able to see the user's palm. However, when mounted, the LMC will see the back of the user's hand most of the time. Being able to see the palm might reduce the occurrence of finger occlusion, as a bent finger will still be visible to a camera viewing the palm, but be occluded to a camera viewing the back of the hand. Furthermore, an LMC mounted onto a VR head-mounted display will no longer be stationary, possibly causing further recognition issues. Furthermore, the time it takes for a pose to be recognized is an important factor for real-time 3D gestural interaction [29], and this latency is often not recorded. For these reasons, it is reasonable to assume that evaluating the performance of the LMC outside of VR may yield different results to performance tests inside VR. Thus, there is a need to make performance measurements for the LMC in VR.

A wide range of poses is important to measure the effectiveness of a pose recognition system. Most of the researchers that used the LMC in VR used the few recognizable poses built into the Leap Motion software to achieve the aim of using the camera within a particular context. It is thus an important step in this research to construct a set of poses that represent as many hand poses that can be made in VR as possible. Other researchers have introduced data sets of poses, but without the context of VR included. Certain pose types have become a common trend amongst LMC-based VR applications, and a pose set emphasizing these poses is required for this research. Thus, a set of hand poses to be tested will be defined in the course of this research.

1.2 Problem Statement, Aim and Objectives

By exploring an intuitive means of interaction in VR, immersion in VR applications can be improved, thus allowing for more meaningful and engaging applications. One such means of interaction is by recognizing a user's hand poses, and translating it to an action on the environment. This provides a natural means of interaction over the keyboard and mouse as humans predominantly use their hands to interact with their environments. Current virtual reality displays for the PC, such as the HTC Vive and

Oculus Rift, utilize hand-held controllers for gestural VR interaction. These devices accurately track positional hand data, but not individual finger data. They partly rely on button presses for VR interaction, rather than purely relying on natural and intuitive hand poses. Future VR systems are expected to evolve towards natural hand gestures as the primary input, thus there is a need to explore different methods of capturing more complete hand data. One such method is a camera or vision-based method, where image recognition techniques and classification methods are combined to recognize a hand pose. Vision-based methods are able to track the posture of individual fingers while not encumbering users' hands, thus possibly replacing cumbersome hand-held controllers. There is a lack of research in the field of camera-based pose recognition in VR, especially with measurements of camera effectiveness. Thus, there is a need to explore the effectiveness of camera-based hand pose recognition for VR environments. It is not known if a depth-based camera would effectively distinguish hand poses made in a VR environment.
Previous work was done on a set of four distinct poses [10]. It is also not known how effectively a depth-based camera could distinguish a larger and more comprehensive set of poses.

1.2.1 Aim

The aim of this research is to evaluate the effectiveness of using a depth camera for pose recognition as a means of interaction in virtual reality.

1.2.2 Objectives

In order to achieve the aim, this research has been broken into sequential objectives as follows:

- Review gesture taxonomies and select a taxonomy to use for pose set creation.
- Create a benchmark pose dataset through use of the LMC in VR. The pose types are chosen using the selected taxonomy and VR applications.

- Evaluate the performance of three machine learning classifiers for pose recognition on the benchmark pose dataset, using features constructed from the LMC API.
- From the findings of the experiments, determine a subset of poses from the dataset that yields a very high degree of accuracy.

1.3 Contributions

This research explores an immersive and natural means of interaction for VR. Several contributions are made to the fields of hand pose recognition and virtual reality. Very little research exists that measures the performance of camera-based pose recognition systems in VR. This research provides results measuring the performance of various machine learning classifiers on poses made in VR and captured via camera. Furthermore, a benchmark set of poses is derived from an existing gesture taxonomy and other existing VR applications. Previously, no dataset of hand poses for VR existed. Researchers can use this benchmark set of poses for comparison, or base their own pose sets on this one. A novel algorithm is also provided to reduce a pose set to achieve a certain accuracy goal. A reliable pose set with over 99% accuracy was formed using the novel algorithm, and could be practically used in current VR applications where high accuracy is necessary. The same algorithm with different parameters could be used to form other reliable pose sets.

1.4 Thesis Breakdown

The structure of the thesis is as follows: Chapter 2 highlights the state of the art in pose recognition for virtual reality, followed by Chapter 3, where the creation of the pose dataset is described. Chapter 4 describes the pose recognition experiments and methods of the research. Chapter 5 displays the results obtained from the experiments, and describes a means by which the pose set is reduced to form a new reliable pose set for a significantly improved recognition rate. Findings of the experiments are

described in the discussion in Chapter 6, which is followed by the concluding Chapter 7.

Chapter 2

Literature Review

This chapter provides a comprehensive review of pose recognition in general and pose recognition for VR in particular. Section 2.1 gives an overview of the use of the Kinect and LMC in research, and compares the two devices. Section 2.2 provides a review of the pose types used in literature and current VR applications. Section 2.3 describes several different gesture taxonomies. Finally, hand pose recognition algorithms from other studies will be highlighted in Section 2.4.

2.1 Pose Recognition with the Kinect and LMC

A variety of input devices besides hand-held controllers have been used in hand pose recognition research. The more popular amongst these include gloves and depth cameras. Most research, however, has not been concerned with VR applications. Popular depth camera-based devices for hand gesture recognition in research include the Microsoft Kinect and the LMC, while glove-based research usually utilizes the CyberGlove [65, 68]. Glove-based devices, such as the CyberGlove, are cumbersome and not as easily available as cameras are, but do not suffer from occlusion as cameras do. Thus, this section only discusses the use of the LMC and Kinect depth cameras in research.

Researchers using the raw data from depth cameras often employ the use of RGB-D

segmentation to extract meaning from images [17, 22, 62]. These techniques may be useful for constructing one's own input features straight from the raw image data. However, there is very limited research measuring the effectiveness of the LMC and its API for VR. As such, this research is more concerned with using this already-available input library, namely the LMC API, and measuring pose recognition effectiveness from it.

Researchers have used the Kinect to manipulate virtual objects using the positions of the hands from the API [28], while others used the Nimble SDK to detect hand joint positions from the Kinect depth image to interact with virtual objects [39]. Various machine learning techniques were compared for gesture classification using the Kinect, where a highest average accuracy of 96.99% was achieved using a neural network [59]. Various researchers have used the Kinect for Sign Language recognition, achieving accuracies of over 80%. [2, 38, 58].

A significant amount of research utilizing the LMC for gesture recognition has been completed. For example, it has been extensively applied to the field of Sign Language recognition [9, 11, 38, 44, 49, 36, 63]. Other studies outside of VR have used the LMC for 3D virtual scene and object manipulation [15, 25], television remote control [61], 3D painting [56], and medical rehabilitation [1, 21, 55].

Due to its lightweight design, the LMC has been used in several VR research papers. Our previous research used the LMC mounted to the Oculus Rift for data visualization and interaction through hand poses [10]. Other research combining the LMC for gestural input and a VR head-mounted display for visual output include robotic arm remote operation [57], 3D model manipulation and visualization [51, 4, 26], 3D virtual navigation [27], and medical rehabilitation [5].

Extensive research has been done on evaluating the performance of depth cameras outside of VR, most notably in Sign Language recognition research, but not within VR. More specifically, very little research into gathering the performance metrics of a hand pose recognition system in VR has been done.

2.2 Hand Poses for VR

In order to develop a VR hand pose recognition system, there needs to be a set of hand poses to be recognized. Defining such pose sets allow other researchers to compare their hand pose recognition results on a common set of poses.

Publicly available datasets of hand poses do exist, such as the *Innsbruck Multi-View Hand Gesture* dataset [53] (captured by the Kinect), the *ChaLearn* dataset [18] (captured by the Kinect), and the dataset by Molina et al. [45] (captured by a time-of-flight camera). Datasets captured by an LMC have also been created, such as the *LeapMotion-Gesture3D Dataset* and *Handicraft-Gesture Dataset* both by Lu et al. [35], however these datasets consist of dynamic gestures. All of the above datasets do not contain gestures that were made in VR environments.

2.2.1 User Elicitation Studies

User elicitation studies involve having users define the hand poses they feel are most natural for a task. While the following studies were not performed in a virtual reality context, some of the poses made by users could provide insight into which poses might feel natural within virtual reality.

Hand poses users made when performing specific tasks on a computer were recorded [13]. The findings show that the "index finger" pose (i.e. only the index finger raised, all other fingers curled), is by far the most common hand shape for gesture interaction on a tabletop display. Other hand poses used include the spread hand, flat hand, grab/release motion, vertical hand, five-finger pinch, fist, 'L' shape, 'C' shape, and curved hand. Figure 2-1 depicts these poses.

User-defined gestures for interaction with objects on a tabletop display were gathered by [67]. The gestures used to interact with the tabletop display are not mid-air 3D hand poses, but are instead dynamic and make contact with a 2D tabletop display. Some findings of note is that users prefer to use a single hand over two when interfacing with the display, and that it may be more effective to provide the user with an

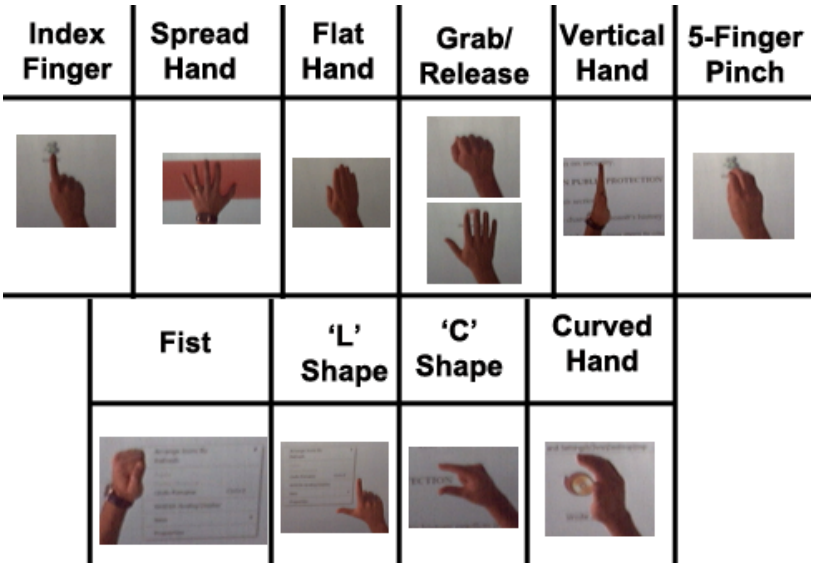


Figure 2-1: Common hand shapes extracted from the work of Epps et al [13].

on-screen widget for more complex commands instead of expecting them to perform a difficult or abstract gesture. Users in this study made extensive use of an index pointing pose to interact with elements on the screen, highlighting the importance of the index finger in gestural interaction.

Insights into user-elicited gestures for television interaction captured by the LMC is given in [61]. Of the gesture-based commands performed by the users, several received high agreement rates (i.e a high percentage of users all made the same gesture for that command). While most gestures made were dynamic, some did include static poses. The high agreement static poses were the open palm, closed palm, and thumbs-up poses.

In a recent elicitation study, gestures were captured via the LMC for musical interaction in a virtual reality environment [34]. Poses that were involved in gestures with high agreement include an index pointing pose, open hand pose, and index-thumb pinch. Many gestures used the same hand pose as one another, but with different motions.

Other smaller studies show that users prefer the five-finger pinch (finger purse) for in-air manipulation [23].

2.2.2 VR Hand Poses in Literature

The LMC was used for a VR Computer Aided Design application [4]. In this application, the two-fingered pinch pose (the thumb and index finger touch) was used to pick up a stationary component of a virtual mechanical device, and users could freely move this component around provided the pinch pose is held. Upon releasing the pinch pose, the component is once again frozen in space. Other actions such as resetting the positions of all components to their default locations were mapped to virtual user interface buttons that the user would touch with their hands.

Hand poses were also used to control movement in VR [27]. The user avatar's movement was controlled by the direction of the palm normal, and making a fist ceased movement.

2.2.3 Non-VR Hand Poses in Literature

The LMC and Kinect were used in a pose recognition system to control a 3D molecular graphics application [51]. The LMC was used for one-handed poses only, while the Kinect poses usually involved two hands. In the one-handed poses, a closed hand causes input to be ignored, while any other posture of the hand causes the system to respond. Translation along an axis is performed by moving the hand along the corresponding axis. Rotation is detected by pivoting the hand about the wrist. For example, in order to rotate about the x-axis relative to the monitor, the user would hold their hand with the palm facing the monitor and wave it forwards while keeping the wrist stationary. The mouse cursor can be moved around by extending the index finger and moving the hand around, while mouse clicks are performed by poking at the screen with the index finger. The mouse click is often used to select certain components in the 3D view.

The Kinect was also used in the manipulation system of [39]. An object is selected by making the hand intersect with that object. A grasping motion can then be made to make the object follow the hand around. Pointing with the index finger moves the

camera forward or backward through the world, depending on how far forward the hand is.

2.2.4 VR Hand Poses in Games and Simulations

2.2.4.1 LMC-based Games and Simulations

Many VR applications have been created specifically for the LMC. Most of these VR applications make use of the default information provided by the Leap Motion API, such as pinch and grab strength, to interact with the environment.

The World of Comenius is an educational VR simulation utilizing the LMC for gestural input [37]. The primary means of interaction is performed through intersecting the User Interface (UI) elements with the index finger. These UI elements could take the form of floating spheres showing new worlds to explore or classic windows containing buttons floating in the virtual space. Users may use a grasping motion then open their hands again to attach an object to their hands in order to move the object around. Doing the same grasp-then-release motion releases the object from the hand.

Blocks is an application developed by Leap Motion to demonstrate the effectiveness of their new Orion software [31]. A menu pops up when left hand's palm is turned upwards, and menu items are selected by intersecting them with the right hand's index finger. The grabbing gesture is used to pick up blocks, a two-handed pinch-and-drag gesture creates new blocks, and turning both palms upwards while moving the hands upwards toggles gravity. The thumbs-up pose is used to continue with the tutorial at the start.

Besides The World of Comenius and Blocks, multiple other VR applications have made use of button-based menu items, such as Geometric [52] and Virtual Strangers [20]. Many VR applications partially rely on menu buttons for interaction, however the buttons are usually placed diegetically, such as on a virtual terminal in the user's view. Interacting with the menu buttons is intuitively done through contact with the index finger, however it is often acceptable for any part of the hand to touch them. Therefore, a finger pointing pose, usually the index pointing pose, is an important

pose for VR interaction. Other important poses from the above applications that either form part of a dynamic gesture or are used as-is include: *Open Hand*, *Fist*, *Pinch*, and *Thumbs-Up*.

2.2.4.2 Other Games and Simulations

The Kinect was also used to control an avatar in the Second Life virtual world [47]. Users are able to control the camera by pretending to hold an imaginary window pane in their hands. This means that both hands are roughly the same distance in front of the user and are in a grasping pose as though they were holding the imaginary pane. Moving the pane to the left or right pans the camera left or right respectively, while pushing the pane forward or backwards zooms the camera in or out respectively. The camera is rotated clockwise by moving the left hand forward and the right hand back, which is the motion the user makes to rotate the imaginary pane.

2.3 Gesture Taxonomies

Gesture taxonomies group hand gestures with similar characteristics into categories. These taxonomies are thus useful in deriving a comprehensive set of hand poses as it allows researchers to take a sample of poses from each category to represent a much larger set of hand poses.

One way to categorize gestures is by the style of gesture [24]. For example, gestures can be classified as being either acts or symbols, where sign languages often employ symbolic gestures, while acts are context-sensitive [54].

Karam and Schraefel created a comprehensive and abstract taxonomy by which hand gestures could be separated into four major categories [24]:

- **Gesture Style:** This describes the gesture itself. This category is further separated into the following subcategories:
  - *Deictic:* Context-sensitive gestures used to identify an object with an ap-

plication domain, such as pointing to an object.

  - *Semaphores:* Gestures that could be either static or dynamic and have a meaning to be communicated to the application. Most gesture recognition research focuses on this gesture style.
  - *Gesticulation:* Gestures that are used naturally during conversations.
  - *Manipulation:* Gestures that have a strong relationship between how the hand moves and how a manipulated object moves, such as grabbing and moving an object.
  - *Sign language:* Gestures similar to semaphoric gestures, with the difference being that they are based on a spoken language.

- **Application Domain:** The application in which the gestures are applied, such as desktop use or virtual reality.
- **Enabling Technologies:** The technologies used in order to capture the gestural data from the user.
- **System Response:** The means by which the system responds to a gesture, for example visual, audio, or through commands to the CPU.

This taxonomy is broad, but lacks detail in separating hand shapes from one another. In more recent research, Vafaei argues that previous taxonomies, such as the one by Karam and Schraefel [24], are too broad, and do not capture specific dimensions, such as the physical form of the hand [60]. It is claimed that the older taxonomies are not related to gestural interaction with computers, and rather just gestures for human communication. Vafaei proposed a taxonomy by adjusting and combining dimensions used in the taxonomies of Wobbrock et al. [67] and Ruiz et al. [50]. The categories defined in the taxonomy include: Nature, Form, Binding, Temporal, Context, Dimensionality, Complexity, Body Part, Handedness, Hand Shape, and Range of Motion. A dimension of note is the Hand Shape dimension, which has assignable values such as *Flat, Open, Bent,* and *Curved*. At the time of his investigations, Vafaei states that Hand Shapes were not used as a dimension for taxonomies. A user elicitation study

was performed to determine the common hand shapes that users make in gestural interaction. Figure 2-2 lists the common hand shapes discovered.

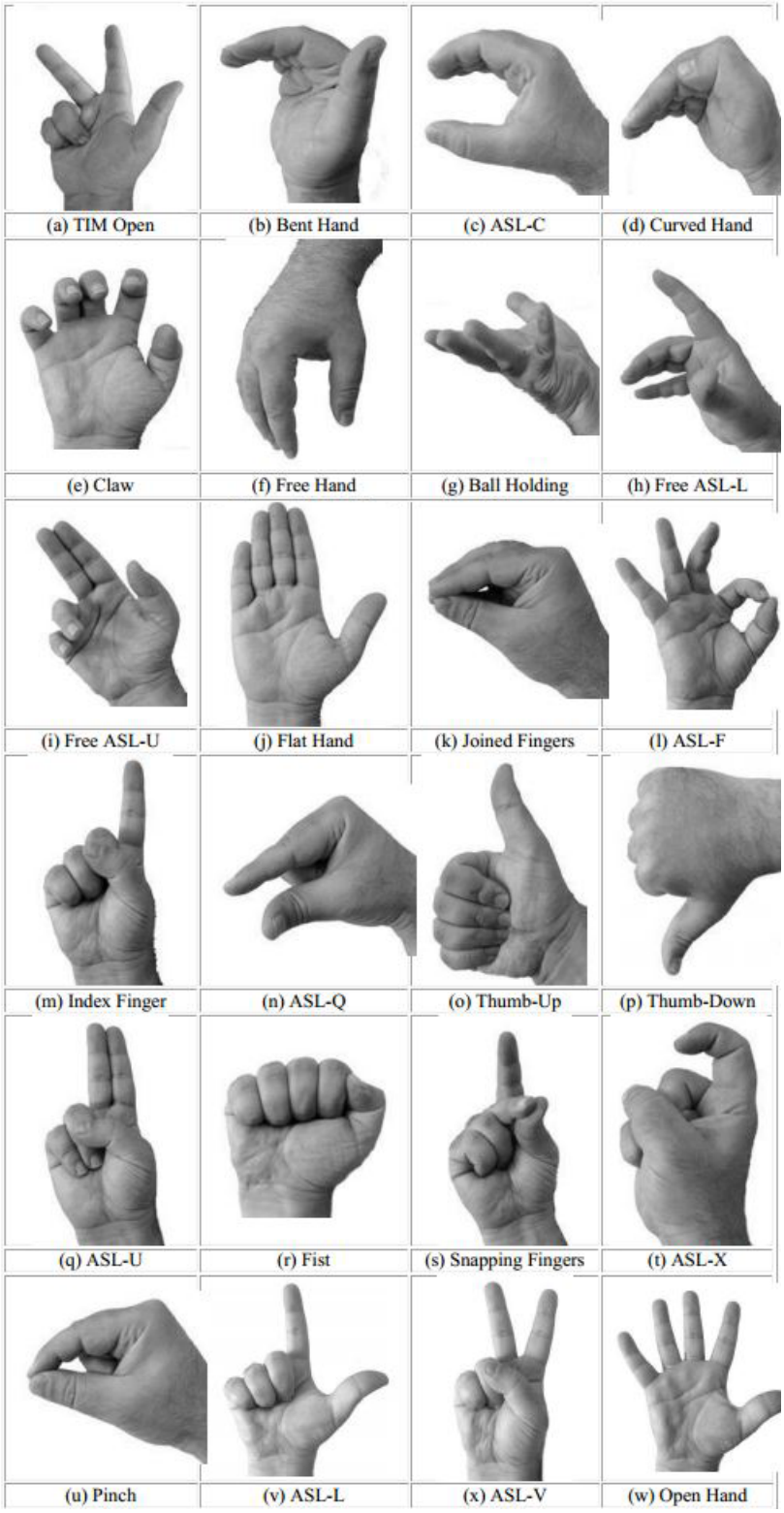


Figure 2-2: Common hand shapes extracted from the user elicitation study by Vafei [60].

While the hand shapes listed by Vafei provide insight into what could be contained

in a comprehensive list of poses, there are no further sub-categories of the Hand Shape dimension. This makes it difficult to create and verify the comprehensiveness of a set of hand poses, as ideally one would want to ensure that the set of hand poses covers all sub-categories of the Hand Shape dimension.

Mo devised a means to notate hand poses by proposing a notating language named GeLex [43]. In GeLex, each finger was described by a *Finger Pose* (Figure 2-3), and each relationship between two fingers was described by a *Finger Inter-relation* (Figure 2-4).

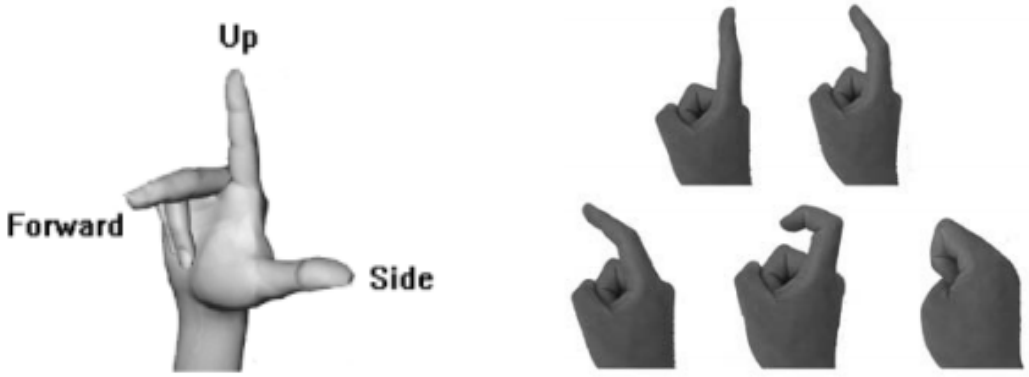


Figure 2-3: Finger poses defined in GeLex [43]. The poses illustrated in the right-hand figure, described from top-left to bottom-right: *Point*, *BendHalf*, *Bend*, *CloseHalf*, and *Close*.

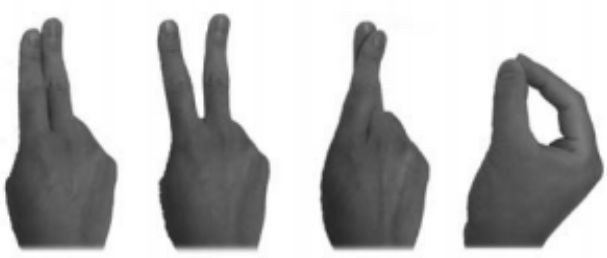


Figure 2-4: Finger Inter-relations defined in GeLex [43]. From left to right: *Group*, *Separate*, *Cross*, and *Touch*.

From these definitions of finger poses and inter-relations, an encoding technique was devised to describe a single hand pose using a series of integers. Each hand pose was described using five integers describing the pose of each finger, followed by four integers describing the relationship between the thumb and each of the other fingers,

followed by three integers describing the relationship between adjacent non-thumb fingers. Therefore, a single hand pose is described by a twelve-dimensional vector of integers.

GeLex separates hand poses very well into intuitive categories, and could provide a solid foundation for a set of representative gestures.

Since many recent studies in hand gesture recognition involve user-elicitation, Choi et al. set the focus of their study on developing a taxonomy that allows researchers to notate these gestures systematically [8]. Figure 2-5 depicts how they separated gestures into categories.

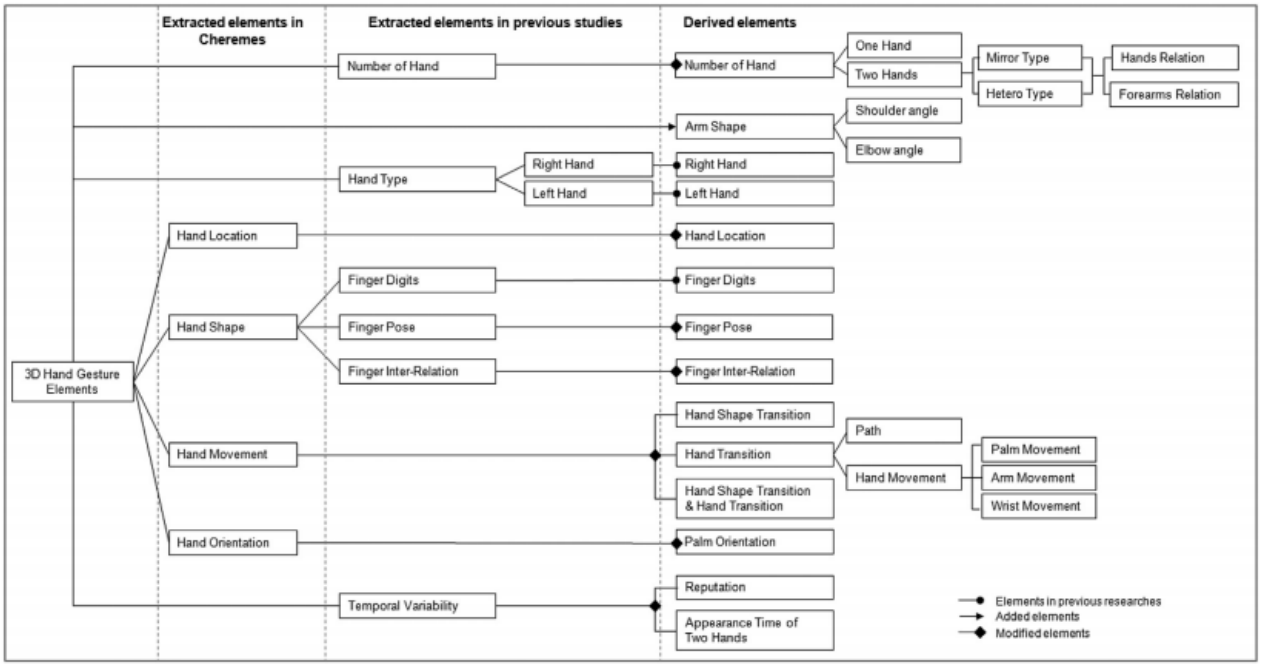


Figure 2-5: The taxonomy developed by Choi et al. [8]

The *Hand Shape* category is analogous to hand pose, and unlike most previous taxonomies, the *Hand Shape* category was further separated into sub-categories. Choi et al. based their categorization of *Hand Shape* on GeLex by Mo [43], and divided *Hand Shape* into *Finger Poses* and *Finger Inter-relations*. They simplified the *Finger Poses* and expanded on the *Finger Inter-relations* in GeLex. Figure 2-6 illustrates the *Finger Poses* and *Finger Inter-Relations* proposed by Choi et al.

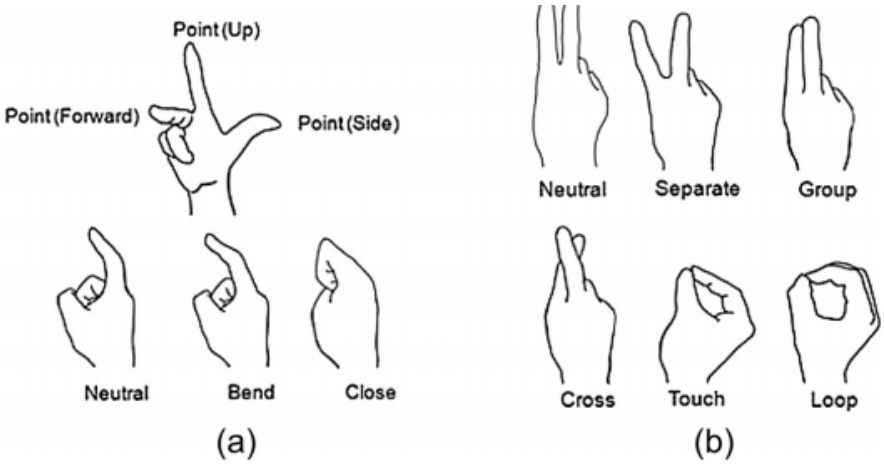


Figure 2-6: a) Finger Pose states. b) Finger Inter-relation states. Extracted from Choi et al. [8]

Another category of note in their taxonomy is the *Hand Orientation* category, which is further divided into *Palm Orientation* and *Fist Face Orientation*. *Palm Orientation* is the direction of the palm normal, while *Fist Face Orientation* is the direction the knuckles would point in if a fist were to be made. Both of these sub-categories could have an assignable state of forwards, backwards, left, right, up, or down.

Older taxonomies are very broad, and such general categorization is not applicable for the purposes of this research. In order to evaluate the performance of a camera-based system, it is obvious that changes in hand shape and orientation will have more of a direct impact on recognition performance than gesture meanings and styles. Of the taxonomies reviewed, Choi et al.'s taxonomy expands on GeLex and provides an in-depth means of separating gestures by hand shape and orientation. This will provide a strong basis to create a comprehensive pose set, as the evaluation of the set will simply involve ensuring that each sub-category of Choi et al.'s *Hand Shape* and *Hand Orientation* categories are represented in the comprehensive pose set. The construction of this pose set and a more in-depth view of Choi et al.'s notation method can be seen in Chapter 3.1.

2.4 Machine Learning Classifiers for Hand Pose Recognition

Multiple researchers have employed various techniques for recognizing hand poses utilizing different input devices and feature sets for the classification process. Widely used techniques include the Support Vector Machine, k-Nearest Neighbour algorithm, and Artificial Neural Networks.

Our previous research utilized the Leap Motion Controller mounted onto the Oculus Rift to capture hand poses for a virtual reality application [10]. The k-Nearest Neighbour algorithm was used as a classifier for the purposes of that research. A recognition rate of 82.5% was achieved on four distinct poses with a value of $k = 3$.

The LMC can provide inconsistent data at times, and thus the idea of combining the LMC and Microsoft Kinect to perform gesture recognition was investigated [38]. Data features extracted from the two camera-based devices was used to train a multi-class Support Vector Machine. The Support Vector Machine's recognition rate on ten hand poses was recorded using the data features extracted from only the LMC, then from the Kinect, and finally from both. The LMC features achieved a 80.86% accuracy, the Kinect achieved a 89.71% accuracy rate, and the combination of the features achieved an accuracy of 91.28%.

The LMC was used to detect the American Sign Language alphabetical hand poses [9]. The k-Nearest Neighbour algorithm achieved a 72.78% accuracy, while the the Support Vector Machine achieved an accuracy of 79.83%. Features used by both of these classifiers consist of the pinch and grab strength, both of which are provided by the Leap Motion API, and a set of derived features. These derived features include the sum of distances all fingers have moved averaged across capture frames, the sum of distance between fingers averaged across capture frames, the sum of triangular areas between fingers averaged across frames and the distance of each finger's furthest joint from the centre of the palm. Their k-Nearest Neighbour algorithm performed consistently for values of k below 150. Various kernel functions in the Support Vector Machine were tested, and the Gaussian radial basis function was found to be an

effective kernel for classification. Training and testing was done using 4-fold cross validation.

Two LMCs were used to detect the Arabic Sign Language's twenty-eight alphabetical signs using Linear Discriminant Analysis as the classifier [44]. The LMCs were placed at right angles to one-another such that each is able to see the hand pose from a different perspective. The features extracted from these cameras included data such as fingertip positions, palm positions, hand orientation, and palm sphere radius, all of which are directly provided by the Leap Motion API. By combining the data features of the two cameras, an average recognition rate of 97.7% is achieved.

The LMC was also used to classify alphabetical and numeric American Sign Language hand poses using a Multilayer Perceptron [36]. After finding optimal parameters for the Multilayer Perceptron, an average recognition accuracy of 90% was achieved.

Extensive research has been done on recognition classifiers using cameras other than the LMC. The Support Vector Machine algorithm was used to classify gesture data captured by the Kinect [14]. An 83.5% recognition accuracy was achieved on static number poses using an Artificial Neural Network with thirty hidden layers for classification [58]. The k-Nearest Neighbour, Naïve Bayes, Artificial Neural Network and Support Vector Machine techniques were compared with regard to pose recognition using the Kinect [59]. The k-Nearest Neighbour, Neural Network and Support Vector Machine all achieved recognition rates of over 80%, while the Naïve Bayes method of classification underperformed with an average rate of 46%.

Chapter 3

A Benchmark Pose Dataset for Virtual Reality

This Chapter describes creation of the pose dataset, named the benchmark pose set, as it provides researchers a common pose set to compare results. Section 3.1 describes the pose set and its creation. Section 3.2 covers the data gathering process, where a dataset is created from captured hand poses created by multiple participants.

3.1 A Static Pose Set for Virtual Reality

Choi et al.'s taxonomy [8] was used to construct a pose set by utilizing key elements from their hand gesture taxonomy. Figure 3-1 illustrates the elements they identified as being part of a 3D hand gesture.

Table 1
The 3D hand gesture taxonomy.

Group	Elements of a hand gesture	Sub-elements of a hand gesture	
Gesture type	One hand	Right hand Left hand	
	Two hands	Mirror type	Hands relation (HR) (separate, touch, cross, clasp)
		Hetero type	Forearms relation (FR) (separate, touch, cross)
Static gesture	Hand location	A physical location where hand gesture takes place	
	Hand shape	Finger pose (point, neutral, bend, close) Finger inter relations (neutral, separate, group, cross, touch, loop)	
	Hand orientation	Palm orientation (PO) (top, bottom, right, left, forward (away from body), backward (toward body)) Fist face orientation (FFO) (top, bottom, right, left, forward (away from body), backward (toward body))	
	Arm shape	Shoulder angle (SA) (SA = 0°, 0° < SA < 90°, SA = 90°, 90° < SA < 180°, SA = 180°) Elbow angle (EA) (EA = 180°, 180° < EA < 90°, 90°≤ EA ≤ 0°)	
Dynamic gesture	Hand shape transition	A hand shape at the start point is different from a hand shape at the end point	<Temporal variablity> Repetition (the number of repetitions)
	Hand transition	<Path> Linear movement, curvilinear movement (clockwise, counterclockwise) Hand location <Hand movement> Arm movement (SA movement, EA movement, SA and EA movements) Wrist movement (flexion, extension, clockwise rotation, counterclockwise rotation) Hand orientation movement (PO movement, FFO movement, PO and FFO movements) Hand location	Appearance time of two hands (simultaneous appearance, successive appearance)
	Hand shape transition and hand transition	[Hand shape transition] A hand shape at the start point is different from a hand shape at the end point [Hand transition] <Path> Linear movement, curvilinear movement (clockwise, counterclockwise) Hand location <Hand movement> Arm movement (SA movement, EA movement, SA and EA movements) Wrist movement (flexion, extension, clockwise rotation, counterclockwise rotation) Hand orientation movement (PO movement, FFO movement, PO and FFO movements) Hand location	

Figure 3-1: Elements of 3D hand gestures, extracted from the work of Choi et al. [8]

The elements identified in this paper are too broad for the context of this research, as we are only concerned with single hand poses placed in front of the viewer's face in VR. To this end, only the *Hand Shape* and *Hand Orientation* parameters will be considered for this pose set. The other parameters will apply to each pose as follows:

- The *Gesture type* parameter will be fixed at *One Hand*, which should be independent of left or right hand.
- The poses will be independent of *Hand Location* and *Arm Shape*.
- No *Dynamic gesture* data will be incorporated, as the pose set is made of static poses only.

Figure 3-2 illustrates how the *Hand Shape* and *Hand Orientation* parameters are notated.

Hand shape (HS)	Finger pose (FP)	Pointing (Up)	1
		Pointing (Forward)	2
		Pointing (Side)	3
		Neutral	4
		Bend	5
		Close	6
	Finger inter relation (FIR)	Neutral	1
		Separate	2
		Group	3
		Cross (F_i in front of F_j)	4
		Cross (F_j in front of F_i)	5
		Touch	6
		Loop	7
Hand orientation (HO)	Palm orientation (PO)	Top	1
		Bottom	2
		Right	3
		Left	4
		Forward	5
		Backward	6
	Fist face orientation (FFO)	Top	1
		Bottom	2
		Right	3
		Left	4
		Forward	5
		Backward	6

Figure 3-2: Notations for Hand Orientation and Hand Shape, extracted from Choi et al. [8]

A single hand shape (HS) is represented by five finger poses, then four finger inter-relations between the thumb and each of the fingers, then three inter-relations between adjacent non-thumb fingers. The format is as follows:

$$HS = f_1 f_2 f_3 f_4 f_5; f_{12} f_{13} f_{14} f_{15} - f_{23} f_{34} f_{45}$$

Where f_i represents the finger pose of finger i, and f_{ij} represents the finger inter-relation between fingers i and j. Finger 1 is the thumb, and finger 5 is the pinky. For example, a fist pose with the thumb pointing up would be represented by 16666;3222-333.

Hand orientation (HO) is represented in the following format:

$$HO = PO; FFO$$

Where PO and FFO are Palm Orientation and Fist-Face Orientation respectively. In the case where the palm faces forward and the fists point up, the Hand Orientation would be denoted as 5;1.

An ambiguity exists in the *finger inter relation* parameter of the hand shape.

Whenever a finger crosses in front of another, one could argue that if the hand orientation were inverted from facing forwards to facing backwards, the same finger would now instead be behind the other. This is disambiguated by defining the *Cross (F_i in front of F_j)* as finger i crossing over finger j on the palm side.

The left and right hands are mirror-images of one another, which would cause problems with the *Hand orientation* values of *Left* and *Right*. To illustrate this problem, consider a thumbs-up pose using the right hand. In this case, the *Palm orientation* is *Left* and the *Fist face orientation* is *Forward*. However, if we were to use these same orientations with the left hand, the thumb would now point down. Instead of using absolute values such as *Left* and *Right*, the problem can be resolved by renaming them to *Inwards* and *Outwards*, where *Inwards* is left for the right hand and right for the left hand, and *Outwards* is simply the opposite. Using this notation does mean that pointing the index finger *Inwards* means we're pointing left with the right hand, or right with the left hand, which could cause issues if the pointing direction is important. Such problems could be resolved during the VR application's execution, as the purpose of this solution is to make this pose set left and right hand independent.

3.1.1 Outline of the Benchmark Pose Set

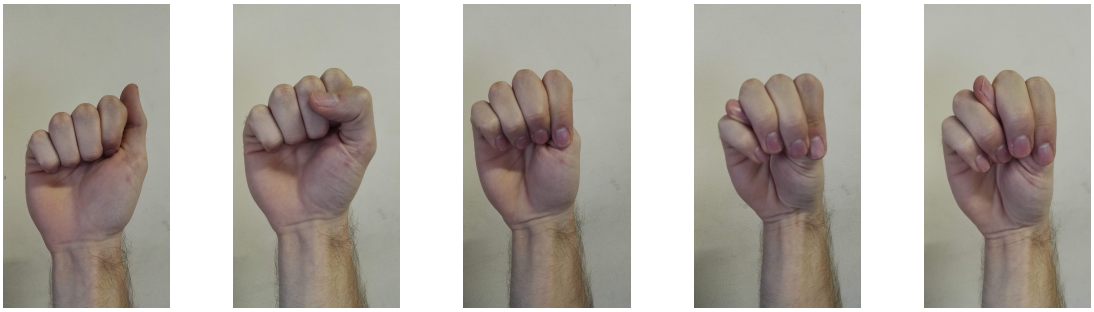
The pose set must contain all the poses common in LMC-based VR applications, as identified in Section 2.2.4.1. These poses are the *Open Hand*, *Point*, *Fist*, *Pinch*, and *Thumbs-Up* poses. Furthermore, the set will contain a wide variety of poses that have significant differences to each other, as well as poses that are similar to one another. This will make the set cover a broad spectrum of poses, yet also contain enough poses that share similarities to test the separating power of the LMC. Thus, various categories of poses will be proposed, each of which will contain poses with minor differences. The full list of poses can be seen in the appendix in Chapter 8. Note that some pose types will have the *ASL-* prefix, meaning that the pose described is an alphabetical letter in American Sign Language.

Hand Shape (HS)	**Finger Pose (FP)**	Pointing (Up)	1
		Pointing (Forward)	2
		Pointing (Side)	3
		Neutral	4
		Bend	5
		Close	6
	Finger Inter-Relation (FIR)	Neutral	1
		Separate	2
		Group	3
		Cross (F_i on palm-side of F_j)	4
		Cross (F_j on palm-side of F_i)	5
		Touch	6
		Loop	7
Hand Orientation (HO)	**Palm Orientation (PO)**	Top	1
		Bottom	2
		Outwards	3
		Inwards	4
		Forward	5
		Backward	6
	Fist-Face Orientation (FFO)	Top	1
		Bottom	2
		Outwards	3
		Inwards	4
		Forward	5
		Backward	6

Figure 3-3: A modified version of Choi et al.'s notation (Figure 3-2). The Left and Right values are replaced by Inwards and Outwards respectively, and the ambiguity in the *Cross* values has been resolved.

3.1.2 Fist Poses

These hand poses involve the non-thumb fingers being curled closed.

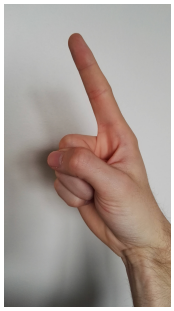


(a) ASL-A (b) Classic fist (c) Hidden thumb (d) ASL-M (e) ASL-N

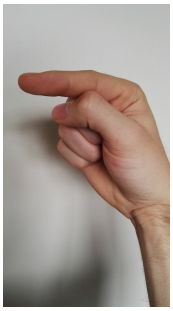
Figure 3-4: Fist Poses.

3.1.3 Index Pointing Poses

These poses involve the index finger being extended while the other finger poses are closed.



(a) Point



(b) Index-Forward

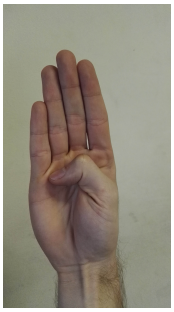
Figure 3-5: Index Pointing Poses.

3.1.4 Open-Palm Poses

These poses involve most of the fingers being extended.

(a) Open Hand

(b) Neutral Hand



(c) ASL-B

(d) Flat Hand

(e) Thumb-Middle Group

(f) Spok

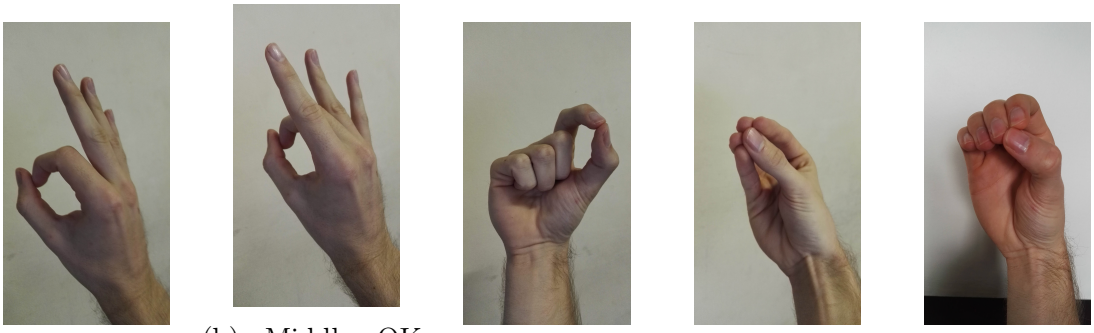
(g) Claw

(h) ASL-C

Figure 3-6: Open-Palm Poses.

3.1.5 Finger Touches and Loops

A finger touch occurs whenever two fingers touch at the fingertips, while a loop is whenever two fingers touch to form a circular shape.

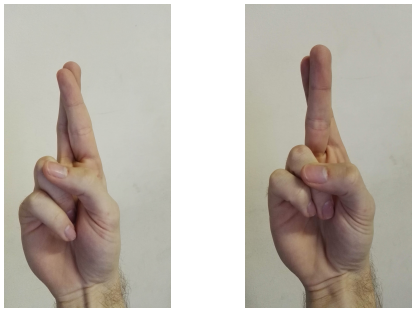


(a) OK-Pose (b) Middle OK-Pose (c) Pinch (d) Finger Purse (e) ASL-O

Figure 3-7: Finger Touches and Loops.

3.1.6 Finger Crosses

These poses involve one non-thumb finger crossing another non-thumb finger.



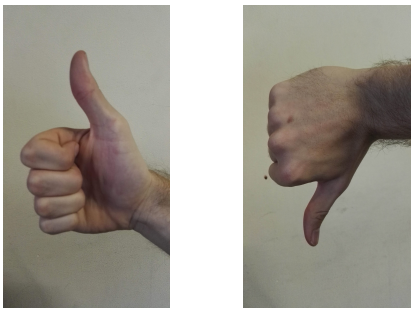
(a) ASl-R (b) Inverse ASL-R

Figure 3-8: Finger Crosses.

3.1.7 Thumbs-Up Poses

These poses involve the Thumbs-up hand shape in different orientations. Each pose below has the hand shape [HS = 36666;2222-333] with varying orientations. As such, only the orientation parameter is shown in the table.

(a) Thumbs-up (b) Thumbs-out (c) Thumbs-in (d) Thumbs-down (e) Thumbs-back



(f) Thumbs-up, fist-in (g) Thumbs-down, fist-in

Figure 3-9: Thumbs-Up Poses.

3.1.8 Analysis of the Pose Set

The aim of the above pose set was to include all the common pose types found in LMC-based VR applications, as well as contain multiple poses that are similar to one-another to test separating power. It is clear that this pose set contains the common poses found in VR applications due to the presence of the *Open Hand*, *Point*, *Classic Fist*, *Pinch*, and *Thumbs-Up* poses in the set. To analyse how this pose set covers a wide array of possible poses, each parameter in the Hand Shape (Subsections 3.1.8.1, 3.1.8.2, and 3.1.8.3) and Hand Orientation (Subsection 3.1.8.4) attributes will be highlighted to describe how the use of that parameter has been covered by the above pose set. Subsection 3.1.8.5 discusses similarities across certain poses.

3.1.8.1 Finger Poses

These poses are represented by the first five digits of the Hand Shape parameter. Each value of the Finger Pose parameter will be listed, and notable hand poses that

use that value will be mentioned alongside the value.

1. **Pointing (Up):** Represented by multiple fingers in the *Open Hand* and *Flat Hand* poses, and by a single finger in all the *Index Pointing Poses* as well as by the thumb in the *ASL-A* pose.

2. **Pointing (Forward):** This value is represented by all the fingers in the *Finger purse* pose, and by only the index finger in *Index-Forward* pose.

3. **Pointing (Side):** Only the thumb is able to point out to the side. The thumb points out to the side notably in the *Thumbs-up Poses*, as well as the *Open Hand* pose.

4. **Neutral:** The *Neutral Hand* pose has all fingers in the neutral pose. This value is not extensively represented since VR applications are expected to generalize this to a *Pointing* pose due to its similarity to such a pose.

5. **Bend:** The *Claw*, *ASL-C*, and *ASL-O* poses have multiple fingers bent, while the *OK-Pose* illustrates the use of fingers being bent to form a loop.

6. **Close:** The *Fist Poses* all have multiple fingers curled all the way closed to form a fist, and the *Index Pointing Poses* have the non-index fingers closed to emphasize the pointing index finger, while the *ASL-B* pose has only the thumb closed.

3.1.8.2 Finger-Thumb Inter-Relations

The inter-relation between the thumb and any other finger is represented by the 6^{th} through 9^{th} digits of the Hand Shape parameter. Below, the possible values of these parameters are listed, with examples from the pose set alongside them:

1. **Neutral:** This value occurs whenever the thumb is not touching a finger, but is at the same time not far separated from it. As with the neutral finger poses,

VR applications are expected to generalize this to either *Separate* or *Group*. As a result, this is only represented in the *Neutral Hand* pose.

2. **Separate:** Most poses involve the thumb separated from the other fingers, and very few poses, such as the *Finger Purse*, has the thumb not separated from other fingers.

3. **Group:** The *ASL-A* and *Flat Hand* poses both have the thumb grouped with the index finger. Since no other finger besides the index finger is adjacent to the thumb, it is difficult to group the thumb with any other finger without calling it a *Cross*. The *Thumb-Middle Group* pose depicts an attempt at grouping the thumb with the middle finger.

4. **Cross ($\mathbf{F}_i$ in front of $\mathbf{F}_j$):** The *ASL-M* and *Hidden Thumb* poses have the thumb crossing over three fingers. Other examples include the *ASL-N* and *Thumb-Middle Group* poses.

5. **Cross ($\mathbf{F}_j$ in front of $\mathbf{F}_i$):** It is difficult to get the thumb to cross behind upright fingers, however, when fingers are bent, this is an easier task. The *Classic Fist* and *ASL-R* poses are examples of this.

6. **Touch:** The *Finger Purse* pose involves all fingers touching the tip of the thumb, while the *Pinch* pose involves only the index finger performing a *Touch* with the thumb.

7. **Loop:** The *ASL-O* pose has two fingers forming a loop with the thumb, while the other two fingers form a loop without touching the thumb. The *OK-Pose* involves only the index finger looping with the thumb, while the *Middle OK-Pose* has the middle finger looping with the thumb.

3.1.8.3 Finger-Finger Inter-Relations

These three inter-relations exist between adjacent non-thumb fingers, and are represented by the final three digits of the Hand Shape parameter. The possible values for

these parameters are the same as the Finger-Thumb Inter-Relations, and are listed below:

1. **Neutral:** The *Neutral Hand* pose is a prime example of all fingers being neutral with other adjacent fingers. Furthermore, any pose where adjacent fingers are separated from one-another by the thumb were labelled has have a neutral inter-relation, such as in the *ASL-M* and *ASL-N* poses.

2. **Separate:** The *Open Hand* pose has all fingers fully separated from one-another. All of the *Index Pointing Poses* have the index finger separated from the middle finger in order to form a pointing pose.

3. **Group:** The *Flat Hand* and *ASL-B* poses have adjacent fingers grouped together. Most of the *Fist Poses* involve fingers curled and grouped to form a fist. The *Spok* pose has two pairs of fingers grouped, with the middle pair of fingers separated.

4. **Cross ($\mathbf{F}_i$ in front of $\mathbf{F}_j$):** This value is represented by the *ASL-R* pose, where the index finger crossed in front of the middle finger.

5. **Cross ($\mathbf{F}_j$ in front of $\mathbf{F}_i$):** The *Inverse ASL-R* pose represents this value by having the index finger cross behind the middle finger.

6. **Touch:** A *Touch* occurs whenever two fingertips touch. It is impossible to do this using two non-thumb fingers.

7. **Loop:** As with the *Touch*, it is impossible to form a circle using adjacent fingers.

3.1.8.4 Hand Orientation

The above pose set covers multiple orientations in the *Thumbs-up Poses*, where the same hand shape in different orientations has different meanings. This is primarily seen in the difference between the *Thumbs-up* and *Thumbs-down* poses, where two semantically opposite poses share the same hand shape at different orientations. The

Palm Orientation parameter has all six of its values covered in this category of poses, however only three of the Fist-Face Orientation's six values are covered. This is due to the unnatural bending of the arm required to make the fist face either outward, down, or backwards.

The *Thumbs-up* pose was chosen here as it is one of the few poses that has its meaning changed according to the orientation of the hand. More specifically, it is generally accepted that the *Thumbs-up* and *Thumbs-down* poses are opposite to one another, despite being the same hand shape. This makes the *Thumbs-Up* pose a good candidate for testing hand orientation classification, as it is a pose that is likely to have its orientation matter in virtual reality scenarios.

3.1.8.5 Pose Similarities

The pose set has been constructed in such a manner that each category contains poses that are similar to one-another. This was done to increase the difficulty of classification and thus highlight problem areas for camera-based pose recognition. For example, the *ASL-A* and *Classic fist* poses both belong to the *Fist Poses* category, because in both poses all four non-thumb fingers are curled up into a fist, and they only differ in the pose of the thumb.

Some cross-category similarities do exist, such as between the *Fist Poses* and *Thumbs-Up Poses*, where the poses differ by the placement of the thumb. Thus, additional noise exists in the dataset when any *Thumbs-Up Poses* are being classified.

3.2 Construction of the Dataset

A total of 25 participants took part in the construction of the dataset, where 102 data captures over 29 hand poses were made for each participant. Each participant entry contains data about the participant as well as the raw data captured by the LMC pertaining to every pose they made. This section describes the setup used to capture the data, followed by a description of the data capturing process.

Participants were seated at a table and given VR peripherals to use. The VR periph-

erals consisted of the Oculus Rift DK2 with the Leap Motion Controller mounted to the front of it. The Rift's head-tracking camera was mounted on a monitor facing the participant. These peripherals are plugged into a computer with the following specification:

- CPU: Intel i5 7600K (4 cores at 3.8GHz)
- RAM: 16GB DDR4
- GPU: AMD HD 7970 GHz Edition
- OS: Windows 10 64bit

The researcher was seated in front of a monitor that displayed what the participant currently sees through the Oculus Rift. The researcher controlled the VR application that captured the participant's hand poses. A VR environment created using the Unity game engine was used to capture the poses. The environment rendered the Leap Motion Controller's output as virtual hands, mimicking the participant's physical hands. Each participant was first given an explanation as to the purpose of the experiment and what they were expected to do. They were all requested to use their dominant hand to create the poses. In the event of the virtual hands displaying a significantly different output to the pose they're attempting, the participants were instructed to remove then re-introduce their hands to the scene in the same pose. Should there still be an erroneous output, the data is captured regardless. Upon entering the environment, participants were given a few minutes to familiarize themselves with the VR environment before getting started.

Once they were ready, they were asked to make the hand pose displayed in front of them at any orientation of their choosing. Once the experimenter was satisfied that they were making the correct pose, the data is captured and stored into memory. They were then asked to do the same pose, but in a different orientation of their choosing again. This is followed by capturing the same pose twice in the orientation displayed to them, known as the requested orientation. This results in two poses being made at any orientation, followed by two poses in the requested orientation.

Figure 3-10: A participant interacting with the experimental setup.

The requested orientation is the orientation of a particular pose that attempts to minimize the number of occluded fingers such that the LMC can detect finger positions accurately. This process was repeated for the *Fist Poses*, *Index Pointing Poses*, *Open-Palm Poses*, *Finger Touches and Loops*, and *Finger Crosses* poses. These poses will be referred to as the *Normal Poses*.

For the *Thumbs-Up Poses*, hand orientation plays an important role, thus participants were not asked to choose a random orientation. For these poses, two data captures were made at the requested orientation. The requested orientation is not necessarily optimized for the LMC, but rather illustrates to the participant which thumbs-up hand orientation was required.

Table 3.1 summarizes the number of poses captured for each participant.

Table 3.1: The number of data captures done for each participant according to pose.

	Captures at arbitrary orientation (per pose)	Captures at requested orientation (per pose)	Number of Poses	**Total Captures**
Normal Poses	2	2	22	88
Thumbs-up Poses	0	2	7	14
Sum Total			**29**	**102**

Chapter 4

Pose Recognition

This chapter describes the pose recognition experiments performed to evaluate the performance of the LMC. Section 4.1 describes the features extracted and used by most of the machine learning classifiers in the experiments. Section 4.2 describes the three experiments performed to explore the effectiveness of pose classification algorithms using the dataset. Section 4.3 explains the process of parameter tuning for each of the classifiers.

4.1 Feature Engineering

A set of features used in similar research was extracted from the LMC's data and fed as training data to the classifiers. Both Hand Shape and Hand Orientation features were used to fully describe a pose. The choice of these features are based on the modified version of Choi et al.'s hand pose notation as seen in Figure 3-3. As such, a short discussion on how these features are able to separate the different values of the notation's pose attributes from one another is included.

4.1.0.1 Hand Shape Features

The sets of features to be extracted to describe hand shape are listed below. Each set of features will be accompanied by a description of the extracted features as well as a motivation as to why it was chosen. It is important that these features are orientation

independent, allowing these features to avoid any conflict with the *Hand Orientation Features* in Subsection 4.1.0.2.

1. **Normalized tip-to-palm distances**

   - *Description:* A set of five length measurements representing a normalized distance from each of the fingertips to the centre of the palm. Each distance is normalized by dividing the tip-to-palm distance of a finger by the maximum extended length of that finger. The normalized tip-to-palm distance (d_{ni}) of finger i in hand pose n is calculated as follows:

$$d_{ni} = \frac{1}{e_i} \|\vec{p_n} - \vec{f_{ni}}\|$$

   where:
   d_{ni} is the normalized tip-to-palm distance of finger i in pose n.
   e_i is the maximum extended length of finger i.
   $\vec{p_n}$ is the 3D co-ordinate of the centre of the palm in pose n.
   $\vec{f_{ni}}$ is the 3D co-ordinate of the fingertip of finger i in pose n.

   The vectors $\vec{p_n}$ and $\vec{f_{ni}}$ are calculated and provided by the Leap Motion software. e_i is calculated to be the maximum tip-to-palm distance for finger i across all N poses made by a particular user. It is calculated as follows:

$$e_i = \max\left(\|\vec{p_n} - \vec{f_{ni}}\| \quad \forall n \in \mathbb{Z} \text{ where } 1 \leq n \leq N\right)$$

   - *Motivation:* This feature vector provides an efficient means of representing how curled each finger is. The more curled a finger, the closer it gets to the centre of the palm, resulting in a smaller tip-to-palm distance. By normalizing this length, we achieve a hand-size independent vector. Ad-

ditionally, this vector will not experience a change in values if the hand is oriented differently or moved to a different location in space. A similar feature vector was used in [38], however tip-to-palm distances were normalized by dividing by the length of the middle finger. This feature vector was utilized in our previous research [10].

2. **Finger Tri-Areas**

   - *Description:* A set of four area measurements, each representing the area of the triangular space between adjacent fingers. Each triangle between any two adjacent fingers is defined as having vertices at each of the two fingertips, and one vertex between the metacarpophalangeal joints (knuckles) of the two fingers. This point between adjacent knuckles will be referred to as the MCP-midpoint. Figure 4-1 depicts one of these four Tri-Areas.

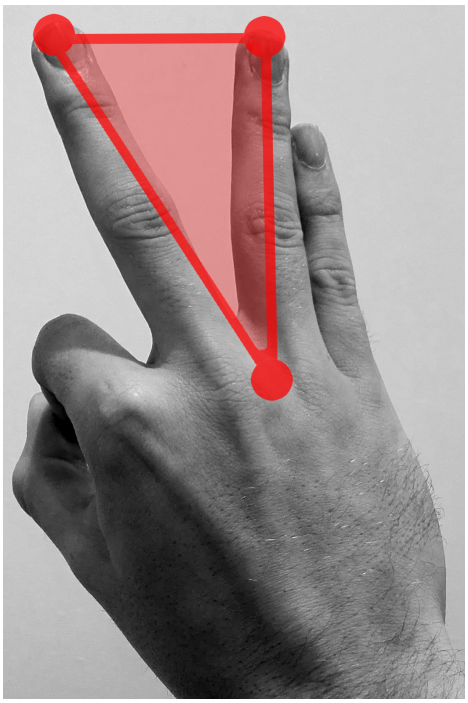


Figure 4-1: The Tri-Area defined between the middle and ring fingers. One vertex is positioned at each fingertip, and the third vertex is at the MCP-midpoint.

Each Tri-Area for a particular hand pose is calculated as follows:

$$A_{i,i+1} = \frac{1}{2}\|(\vec{f_i} - \vec{m_{i,i+1}}) \times (\vec{f_{i+1}} - \vec{m_{i,i+1}})\|$$

where:

$A_{i,i+1}$ is the Tri-Area between fingers i and $i+1$.

$\vec{f_i}$ is the 3D co-ordinate of the fingertip of finger i.

$\vec{m_{i,i+1}}$ is the 3D co-ordinate of the MCP-midpoint of fingers i and $i+1$.

While fingertip positions are provided by the Leap Motion software, the MCP-midpoint is not. The software does however provide the positions of the knuckles. Thus, the MCP-midpoint between fingers i and $i+1$ can be simply calculated as follows:

$$\vec{m_{i,i+1}} = \frac{1}{2}(\vec{m_i} + \vec{m_{i,i+1}})$$

where m_i is the 3D co-ordinate of the metacarpophalangeal joint (knuckle) of finger i. This data is provided by the Leap Motion software.

- *Motivation:* Since the tip-to-palm distances may not be able to effectively track lateral displacements of the fingers, the Finger Tri-Areas vector should be able to cover such a gap. As with the tip-to-palm distances, this feature vector is hand orientation and position independent. Finger Tri-Areas as a feature was applied to Sign Language recognition by Chuan et al. [9], however not directly. They instead summed up the four Tri-Areas and averaged the sums across multiple frames to obtain a single measurement. However, using four separate values will allow for the data pertaining to individual finger spaces to be preserved.

4.1.0.2 Hand Orientation Features

The sets of features to be extracted to describe hand orientation are listed below. As with the hand shape features, descriptions and motivations for the chosen features are listed as well.

1. **Palm Normal Vector**

    - *Description:* A three dimensional normalized vector depicting the normal direction of the palm in Cartesian coordinates. This feature can be extracted directly from the Leap Motion API.
    - *Motivation:* This feature vector is able to fully describe the Palm Orientation attribute of the taxonomy.

2. **Palm Direction Vector**

    - *Description:* A three dimensional normalized vector depicting the direction from the palm to the base of the fingers in Cartesian coordinates. This feature can be extracted directly from the Leap Motion API.
    - *Motivation:* This feature vector is able to fully describe the Fist Face Orientation attribute of the taxonomy.

4.1.0.3 Separation of Notation Attribute Values

This subsection will describe how the chosen features are able to separate the various values of the Hand Pose notation by Choi et al. Each feature will have the values it can separate out described alongside it below.

The six different values of the Finger Pose attribute are primarily separated by the Normalized Tip-to-Palm Distance; the *Pointing*, *Neutral*, *Bend*, and *Close* values can all be separated from one another by measuring the distance from the tip to the palm. The *Pointing (Up)*, *Pointing (Forward)*, and *Pointing (Side)* may only have minor differences in their Tip-to-Palm Distances, however they will be further separated by differences in Tri-Areas between their neighbouring fingers.

In terms of Finger Inter-Relations, the *Neutral*, *Separate*, and *Group* values are easily separable via the Finger Tri-Area feature. The two *Cross* values may have minor differences in Tri-Areas, but will be further separated by their respective distances to the palm. A similar issue arises between the *Touch* and *Loop* values, but is also

separated by the Tip-to-Palm distance.

The Hand Orientation attribute is fully described by the Palm Normal and Direction Vectors as both correlate directly to the attribute they respectively describe. Specifically, the Palm Normal Vector describes the normal direction of the palm, which in turn will separate the six directional values of the Palm Orientation attribute, and the Palm Direction Vector performs the same role with the Fist Face Orientation attribute.

4.2 Machine Learning for Pose Recognition

In Chapter 2, the Support Vector Machine, k-Nearest Neighbour algorithm, and Artificial Neural Network were identified as machine learning classifiers used for pose recognition. The performance of these three classifiers were compared by training and testing on the dataset. They were built, trained, and tested using the machine learning software Weka [66]

In this section, the metrics used to determine the effectiveness of a classifier are listed in Section 4.2.1. Following this, the descriptions of the three pose recognition experiments are given in Section 4.2.2.

4.2.1 Evaluation Metrics

In order to determine the efficacy of a classifier, a set of metrics were defined. The most common metric used in pose recognition is recognition accuracy, which measures how often the classifier is able to correctly classify a pose.

When evaluating the LMC's performance in VR, an additional performance metric, the recognition latency, was measured. This is the time it takes for a hand pose to be recognized. Recognition latency is an important factor for real-time 3D gestural interaction [29]. A high recognition latency might slow the VR application, causing drops in the application's frame rate, and resulting in a higher motion-to-photon latency. This motion-to-photon latency is an important factor for head-mounted VR

displays, and it measures the time it takes for any user motion to be correctly reflected on the display [3]. A high motion-to-photon latency increases the risk of the VR system inducing a feeling of cybersickness, or motion sickness [30].
Finally, a minor metric to consider is the training time of a classifier. Excessively long training times (such as several hours) make it difficult to near-impossible for a hand pose recognition system to have new poses added or old ones removed.
The three metrics used are summarized as follows:

- **Average Recognition Accuracy:** The percentage of times a single hand pose is classified correctly.
- **Average Recognition Latency:** The average time in milliseconds it takes for a single pose to classified.
- **Training Time:** The time it takes to train a classifier.

4.2.2 Experiment Descriptions

Each classifier was trained and tested over three experiments, utilizing the parameters chosen in Section 4.3. Each experiment differs from the other according to training and testing data sets. Each of the classifiers will be tested within the same experiment to obtain comparable results. Results will be obtained through stratified k-fold cross validation, with $k = 25$. The following list describes each experiment:

1. **Orientation-Independent Experiment**
    - **Purpose:** To determine the effectiveness of a classifier in classifying hand poses regardless of orientation.
    - **Data used:** All captured data, except that all *Thumbs-Up Poses* are grouped into the *Thumbs-Up* pose.
2. **Requested Orientation Experiment**

- **Purpose:** To determine the effectiveness of a classifier in classifying hand poses at the requested orientation. The dataset used should have a reduced amount of incorrect data captured by the LMC, as the hand poses in a requested orientation attempt to minimize the number of occluded fingers. The results of this experiment will be compared to the *Orientation-Independent Experiment* to determine the effect of using a requested orientation.
- **Data used:** Only requested orientation data, excluding most *Thumbs-Up Poses*. The *Thumbs-up, fist-in* pose was the only pose amongst the *Thumbs-Up Poses* to be used as it puts all the fingers in the LMC's view.

3. **Thumbs-Orientation Experiment**

   - **Purpose:** To determine the effectiveness of a classifier in classifying pose shape and orientation simultaneously. The *Thumbs-Up Poses* group differ from other pose groups by hand shape, and from one another by hand orientation. By classifying the poses in this group, a classifier would have its hand shape and orientation-determining capabilities tested to achieve a correct classification. The results of this will be compared to the *Thumbs-Up Pose* accuracy in the *Orientation-Independent Experiment* to determine how many *Thumbs-Up Poses* had correct hand shape classifications, yet incorrect orientation classifications. This provides insight into the orientation-distinguishing capabilities of a classifier.
   - **Data used:** All poses form part of the training set, however only the *Thumbs-Up Poses* group are tested.

4.3 Parameter Tuning

In this section, the process by which parameters for the various classifiers were chosen is described. Using the feature set in Section 4.1, tests were performed under different parameter sets for each classifier. This is to find a strong set of parameters for each of

the classifiers. Each parameter set was tested using k-fold cross validation, with the value of k varying according to the training time of the model, so as to not impede rapid testing. Each tested parameter set for a classifier used the same value of k for cross-validation so that results could be directly compared to other parameter sets for the same classifier.

For each of these experiments, poses were classified in an orientation-independent manner to ascertain the recognition accuracy using certain parameter sets. Average recognition latency will be determined by picking a random pose in the set, recording the system time, classifying that pose, then recording the system time again to determine the time that has passed (latency). The latency process is repeated 50,000 times and averaged.

4.3.1 k-Nearest Neighbour

This classifier has only one parameter, k, and varying values of it were tested to find the best value. Before any tests, it was found that the classifier executes quickly enough on the benchmark set, thus a value of twenty-five folds was chosen for the cross validation. Note that due to its nature, the k-Nearest Neighbour algorithm does not require any training time. The training time recordings have thus been omitted as they are always zero. Table 4.1 contains these results, which are further illustrated in graph form in Figure 4-2.

Table 4.1: Initial results for k-Nearest Neighbour classifier across different values of k.

Value of k	Recognition Accuracy	Average Latency
1	**64.3333%**	**0.6655ms**
2	56.2222%	0.7990ms
3	54.7407%	0.8545ms
5	54.2593%	0.8978ms
10	55.6296%	0.9604ms
15	55.0370%	1.0083ms
25	54.3704%	1.1100ms
100	50.8519%	1.5646ms

4.3.1.1 Using Raw Data as Input

It is possible that the feature vector described in Section 4.1 does not provide a significant advantage in accuracy over using all the raw data from the LMC. To disprove this statement, another test was set up where the classifier uses raw data instead of the extracted features. The input vector will contain the following data extracted from the LMC: *Hand Direction Vector, Palm Normal Vector, Palm Position Vector, Wrist Position Vector, Grab Angle, Grab Strength, Palm Width, Pinch Distance, Pinch Strength, Fingertip Position Vector of each Finger, Distal Interphalangeal Joint Position Vector of each Finger, Proximal Interphalangeal Joint Position Vector of each Finger, Metacarpophalangeal Joint Position Vector of each Finger, and whether or not each Finger was detected as being "extended" by the LMC.* This is a feature vector consisting of eighty-two real number entries. Table 4.2 displays the results of the experiment, and they are shown in Figure 4-2. It is clear from Figure 4-2 that using

Table 4.2: Initial results for k-Nearest Neighbour classifier with raw data input.

Value of k	Recognition Accuracy	Average Latency
1	**60.1852%**	**2.0668ms**
2	49.4074%	3.1277ms
3	44.6667%	3.2682ms
5	44.5556%	3.8256ms
10	44.5185%	4.8571ms
15	43.1111%	5.1325ms
25	43.4074%	6.0038ms
100	40.5556%	9.3170ms

the extracted features has a higher accuracy and lower latency than the raw features. Setting the value of k to 1 has a significantly positive effect on performance compared to larger values of k. As such, a value of $k = 1$ was chosen as the optimal value of k, with the extracted features as an input vector.

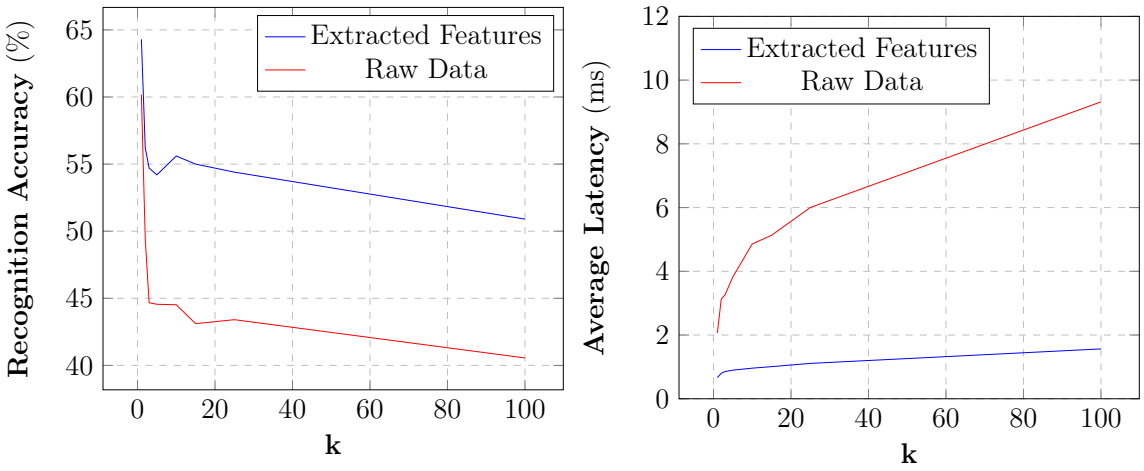


Figure 4-2: Accuracy and latency dependence on k for the k-Nearest Neighbour algorithm.

4.3.2 Artificial Neural Network

The chosen implementation for a Neural Network is the Multilayer Perceptron, as used in [36], where a single hidden layer was used to classify sign language poses. Recognition accuracy is measured through five-fold cross validation. The training time for each model is measured by training a network five separate times on the same data, and averaging the collected times. The following parameter tuning experiments were performed:

- Determining the optimal number of nodes within the single hidden layer of the perceptron.
- Determining whether having multiple hidden layers of nodes provide a significant accuracy improvement.
- Determining the optimal learn rate for the perceptron.

The multilayer perceptron was structured as follows:

Input nodes: One node per element in the input feature vector.

Hidden nodes: Variable number of hidden nodes, each with a sigmoid activation function.

Output nodes: One node per pose class. The output node with the highest value

implies that its corresponding pose class has been chosen.

4.3.2.1 Optimizing the Node Count in the Single Hidden Layer

In this subsection, the number of hidden nodes in a single hidden layer will be varied, while the following parameters automatically chosen by Weka were fixed:

Learning Rate: 0.3

Momentum: 0.2

Number of epochs: 1000

Input Features: Tip-to-palm distances and finger tri-areas.

Output class adjustments: All thumbs poses grouped together as *Thumbs Up* pose to maintain orientation independence.

Number of hidden nodes	Recognition Accuracy	Average Latency	Average Training Time
1	21.8148%	**0.0280ms**	**3.844s**
2	32.4815%	0.0329ms	4.424s
3	44.1481%	0.0367ms	5.142s
5	51.5926%	0.0400ms	6.613s
7	54.3333%	0.0443ms	7.962s
10	55.7037%	0.0510ms	10.014s
15	56.7778%	0.0596ms	13.528s
30	56.6667%	0.0912ms	23.855s
60	56.8148%	0.1469ms	43.384s
100	**57.6296%**	0.2242ms	75.361s
500	57.037%	1.1489ms	328.879s
1000	56.037%	2.39122ms	719.835s

Table 4.3: Initial results for the multilayer perceptron with a single hidden layer.

From Table 4.3, it is evident that any number of nodes above 5 in the hidden layer provides roughly similar accuracies. The maximum acceptable latency is 100ms [6], and the accuracy started decreasing after a hundred hidden nodes. The maximum accuracy of 57.6% is below the 64.3% accuracy achieved in the k-Nearest Neighbour experiment (Table 4.1).

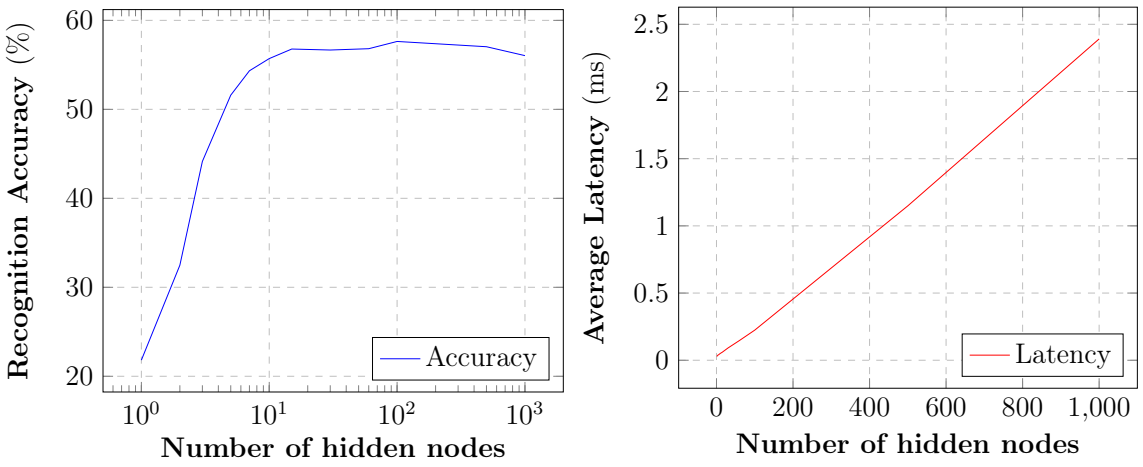


Figure 4-3: Accuracy and latency dependence on the number of hidden nodes in a single hidden layer for the Artificial Neural Network.

4.3.2.2 Testing Multiple Hidden Layers

Here, the effect of having more than one hidden layer in the perceptron was measured. In this experiment, seven nodes were used in a single layer, with each subsequent experiment increasing the number of hidden layers. Table 4.4, containing these results,

Number of Hidden Layers (7 nodes each)	Recognition Accuracy	Average Latency	Average Training Time
2	**51.5185%**	**0.0627ms**	**19.283s**
3	50.7037%	0.0646ms	23.903s
4	47.6667%	0.0739ms	27.352s
5	12.963 %	0.0714ms	31.453s

Table 4.4: Initial results for the multilayer perceptron with varying numbers of hidden layers.

implies that accuracies and performance both start to degrade as more layers are added. However, it was worth further exploring the effect of having two hidden layers to determine if accuracies would increase. In the following experiment, every two-hidden-layer combination of **5, 10, 20, 30, and 60** nodes were tested. The accuracies measured using these two layer set-ups are depicted in Table 4.5. Using two layers failed to make a significant difference in accuracy compared to a single hidden layer. One can conclude from these experiments that using a single hidden layer is as effective as multiple hidden layers.

Number of nodes in: Hidden Layer 1 (Rows) Hidden Layer 2 (Columns)	**5**	**10**	**20**	**30**	**60**
5	46.6667%	53%	53.148%	52.3333%	53.5556%
10	47.5926%	53.5556%	54.5926%	54.7407%	53.8519%
20	48.9259%	54%	55.5556%	55.8519%	54.2222%
30	52.037%	53.8889%	54.6667%	55.6296%	**57.3333%**
60	53.963%	53.7778%	55.7037%	55.7037%	56.7778%

Table 4.5: Initial accuracies for the multilayer perceptron with varying node counts across two hidden layers.

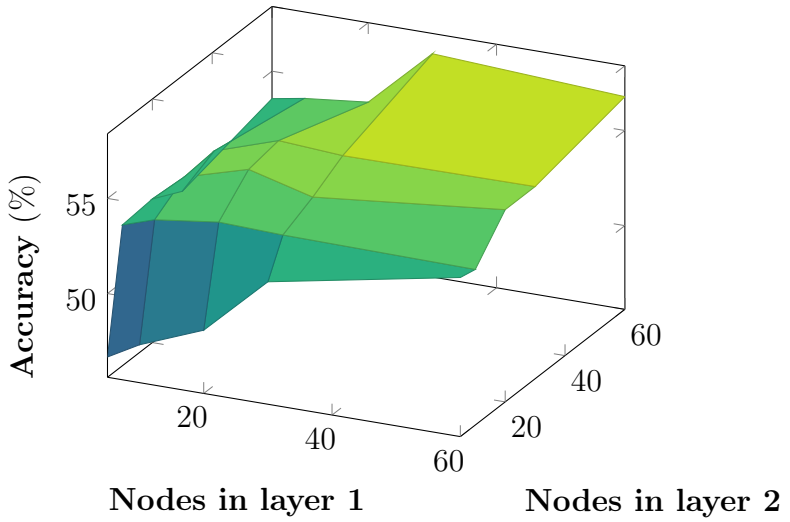


Figure 4-4: Accuracy dependence of the Artificial Neural Network on varying node counts in two hidden layers.

4.3.2.3 Optimizing the Learning Rate

In this subsection, the number of hidden nodes remained constant at ten in a single layer. The learning rate of the perceptron was modified to ascertain its effect on the performance metrics of the classifier. Table 4.6 depicts the effect of different learning rates on the perceptron. While the learning rates of neural networks are usually kept between zero and one, the increasing performance with a higher learning rate warranted exploration of greater-than-one learning rates. However, even after increasing the rate to 1000.0 a significant accuracy improvement was not observed. A spike in latency was seen with the learning rate set to 1.0. This spike is only 16% higher than the lowest recorded latency, and does not warrant further investigation.

Table 4.6: Initial results for the multilayer perceptron with varying learning rates.

Learning Rate	Recognition Accuracy	Average Latency	Average Training Time
0.001	16.2593%	**0.0505ms**	9.315s
0.01	52.2593%	0.0531ms	9.097s
0.05	54.4444%	0.0541ms	9.074s
0.1	55.2222%	0.0544ms	**9.038s**
0.5	54.2593%	0.0567ms	9.288s
1.0	54.8148%	0.0589ms	9.355s
2.0	55.4074%	0.0554ms	9.601s
5.0	54.4074%	0.0512ms	9.207s
10.0	54.9259%	0.0520ms	9.190s
100.0	55%	**0.0505ms**	9.505s
1000.0	**55.4444%**	0.0510ms	9.235s

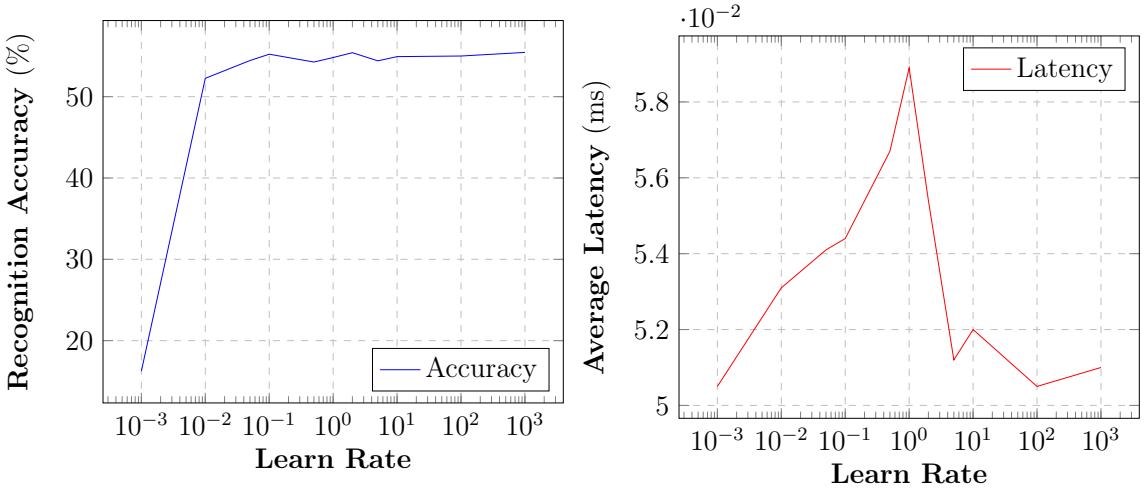


Figure 4-5: Accuracy and latency dependence on the learn rate of the Artificial Neural Network with ten hidden nodes in a single layer.

4.3.2.4 Using Raw Data as Input

After tuning multiple parameters for the multilayer perceptron, it is evident that these parameters may not be to blame for its poor performance. Another avenue for exploration would be to change the input vector for the perceptron from the extracted features (Tri-Areas and Tip-to-Palm Distances) to the raw input mentioned in Section 4.3.1.1. Neural Networks are generally able to handle raw data well, and thus changing the input to a raw format may cause improvements in accuracies.

The neural network using this data has the following parameters:

Learning Rate: 0.3

Momentum: 0.2

Number of epochs: 1000

Input Features: Raw Data.

Output class adjustments: All thumbs poses grouped together as *Thumbs Up* pose to maintain orientation independence.

Hidden Node Structure: Ten hidden nodes in a single hidden layer.

With these parameters, the multilayer perceptron achieved the following results:

Recognition Accuracy: 55%

Average Latency: 0.0841ms

Average Training Time: 20.376s

These results still do not show a significant improvement in accuracy. However, it is likely that an increase in the number of input features will require an increase in the number of hidden nodes. For the next experiment, the parameters from the previous experiment are kept the same, however the number of hidden nodes was be varied.

The results displayed in Table 4.7 and Figure 4-6 show a significant increase in performance when raw input data is used. One hundred or more nodes in a single hidden layer provided good results. As such, the chosen parameters for the neural network are as follows:

Table 4.7: Initial results for the multilayer perceptron with raw data input.

Number of hidden nodes	Recognition Accuracy	Average Latency	Average Training Time
15	55.6296%	**0.1270ms**	**24.670s**
20	56.3333%	0.1533ms	36.287s
50	58.8148%	0.3117ms	83.618s
100	61.5185%	0.5421ms	166.544s
200	**61.5556%**	0.9850ms	299.828s

Input features: Raw input data

Hidden layer structure: One layer, 100 nodes

Learn Rate: 0.3

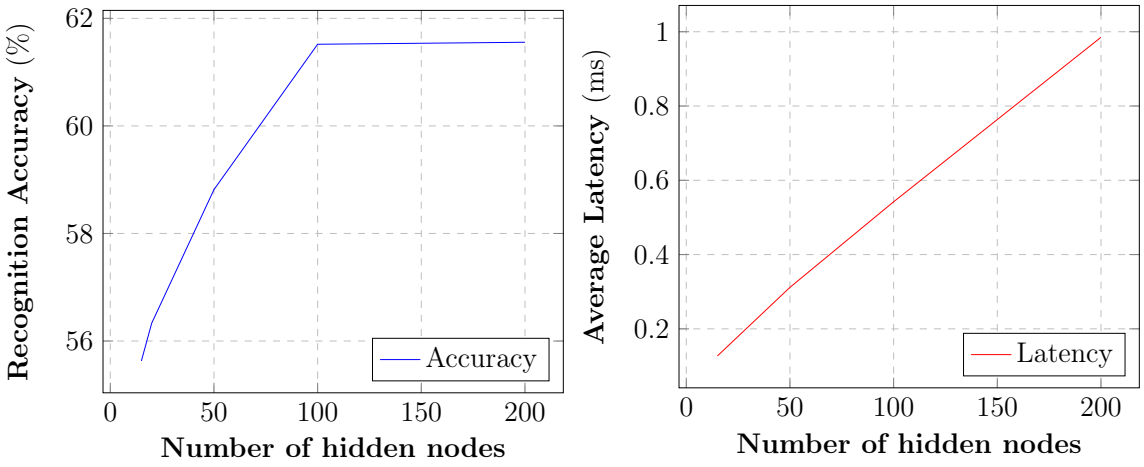


Figure 4-6: Accuracy and latency dependence on the number of hidden nodes in the Artificial Neural Network with raw data input.

Momentum: 0.2

4.3.3 Support Vector Machine

The Support Vector Machine was implemented through Weka's SMO (Sequential Minimal Optimization) algorithm based on [48]. As with the neural networks, accuracies were measured through five-fold cross-validation, and training times are measured as the average over five separate training periods. The following parameter tuning experiments were performed:

- Determining the most effective Kernel function of the available Weka kernels.
- Determining the optimal complexity constant for the Support Vector Machine.
- Determining the optimal parameters for the chosen kernel.

4.3.3.1 Optimizing the Kernel Function

The Normalized Polynomial, Polynomial, PUK (Pearson VII function), and RBF (Radial Basis Function) kernels were compared experimentally in Weka, and the complexity constant was set to 1.0. Since the use of raw input features made a significant impact on performance during the neural network experiments, the kernels will be tested with the raw data input in addition to the extracted feature data. Table 4.8

Table 4.8: Initial accuracies for the Support Vector Machine across various kernel functions.

Kernel	Accuracy with Extracted Feature Data	Accuracy With Raw Input Data
Normalized Polynomial	50.2222%	45.2222%
Linear Polynomial	53.7778%	53%
PUK	**56.2963%**	56.1111%
RBF	31.7037%	37.6296%

shows that the PUK kernel was the most accurate kernel here, however there is no significant performance gain with this kernel when raw data is used. An analysis of the average latencies for each of the kernels have revealed that the PUK kernel has an average latency of 18.3ms and 142ms on the processed and raw data respectively, while the linear polynomial kernel has latencies of 0.09ms and 0.4ms respectively. This is a significant decrease in latency for an accuracy of trade-off of less than 3%. As such, both the PUK and linear polynomial kernels will be further tested in the next subsection.

4.3.3.2 Optimizing the Complexity Constant

The following experiment will illustrate the effect of changing the complexity constant for the SVM using both the PUK and linear kernels. Since Table 4.8 showed no significant difference in accuracy between raw and processed input data, only processed data will be used to lower the average recognition latencies. For these experiments, the latency average will be obtained over 500 individual tests instead of 50000, as the significantly higher latencies with the PUK kernel would slow experimentation time considerably. Tables 4.9 and 4.10 and Figure 4-7 show that the PUK kernel provides significantly higher accuracies at the cost of a higher recognition latency. The accuracies from the PUK kernel and the latencies from the linear kernel are comparable to the neural network's metrics in Table 4.7.

Good parameters were found at a complexity constant of 100.0 for both the PUK and linear polynomial kernels. It is a difficult decision to choose one kernel over the other. As such, both kernels will be used for further testing in the following subsections.

Table 4.9: Initial results for the Support Vector Machine with varied complexity constants using a **PUK kernel**.

Complexity Constant	Recognition Accuracy	Average Latency	Average Training Time
0.001	12.963%	58.5680ms	3.038s
0.01	12.963%	59.0500ms	2.972s
0.1	52.1852%	36.3220ms	1.750s
1.0	56.7407%	20.2843ms	**1.275s**
10.0	59.1111%	15.9212ms	1.517s
100.0	60.3704%	15.8198ms	2.521s
1000.0	60.7778%	**14.6372ms**	6.323s
10000.0	**60.8889%**	14.8397ms	19.242s

Table 4.10: Initial results for the Support Vector Machine with varied complexity constants using a **linear polynomial kernel**.

Complexity Constant	Recognition Accuracy	Average Latency	Average Training Time
0.001	12.963%	**0.0703ms**	0.503s
0.01	22.3333%	0.0705ms	0.293s
0.1	42.963%	0.0751ms	**0.262s**
1.0	54.7407%	0.0788ms	0.283s
10.0	56.0741%	0.0721ms	0.380s
100.0	**56.1852%**	0.0826ms	1.062s
1000.0	55.8889%	0.0806ms	6.689s
10000.0	55.8148%	0.0808ms	70.359s

4.3.3.3 Optimizing the PUK and polynomial kernel parameters

The PUK kernel has two parameters available for modification: the sigma and omega values, which both default to 1.0. Five different values were given to each of the two parameters, creating twenty-five parameter combinations. For each of the two following experiments, the complexity constant is kept at 100.0. Table 4.11 shows the effects of different Puk kernel parameter combinations on the accuracy of the SVM. The values of 5.0 and 0.1 for sigma and omega respectively yield the highest accuracy of 65.18% with an average latency of 32.18ms. The highest average latency recorded was 72.77ms when sigma and omega were both at 0.1, and the lowest latency was 6.21ms when sigma and omega were both 2.0.

One parameter, the exponent, is exposed by Weka for the polynomial kernel. By default, its value is 1.0, making it linear. Table 4.12 and Figure 4-9 illustrate the

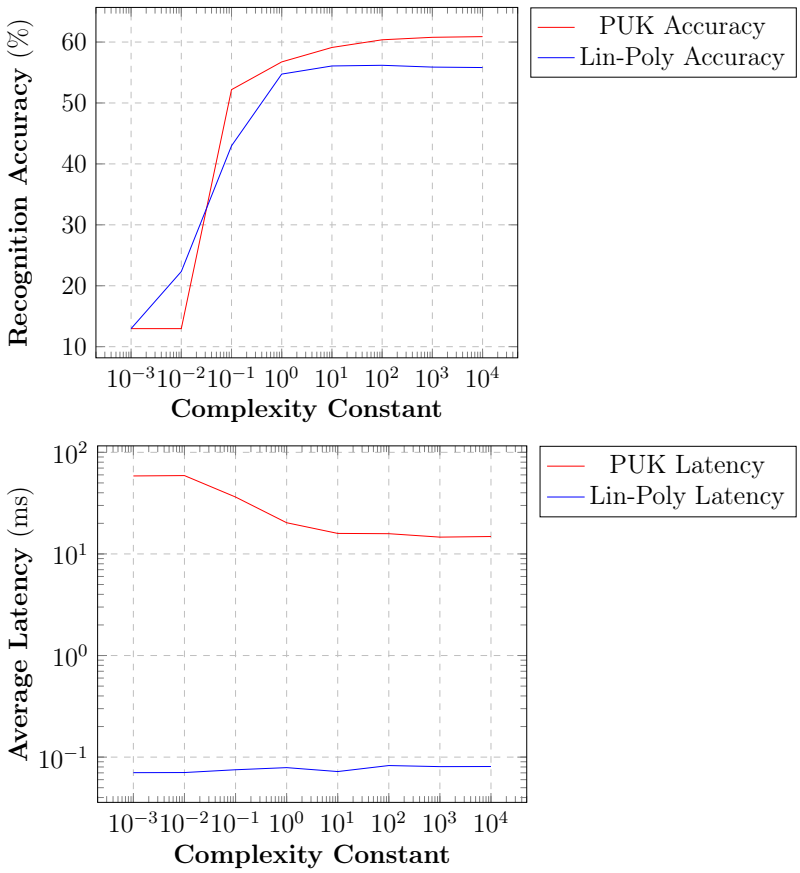


Figure 4-7: Accuracy and latency dependence on the Support Vector Machine complexity constant for two different kernels.

effect of making the kernel non-linear by changing the exponent. Raising the exponent to ten and above results in training times of close to an hour each, and have thus been omitted. An exponent of 1.5 leads to a small accuracy increase over 1.0, however the average latency increases by a couple of orders of magnitude. This makes the latency comparable to that of the PUK kernel, where accuracies are much better than 58%. Thus, an exponent of 1.0 (linear) is considered the best for this kernel, as it maintains its advantage over the PUK kernel in terms of average latency.

4.3.4 Improving the Initial Results

Multiple groups of hand poses in the dataset have been designed to be very similar to one another. One could thus make the assumption that it is unlikely that all similar poses would be used in a single VR application, and rather a single pose would be

Table 4.11: Initial accuracies for the Support Vector Machine with varied PUK kernel parameters.

Sigma (Rows) Omega (Columns)	**0.1**	**0.5**	**1.0**	**2.0**	**5.0**
0.1	63.1111%	64.8519%	63.5556%	56.4815%	46.1481%
0.5	63.7778%	64%	62.1111%	61.4444%	60.7407%
1.0	64.8889%	62%	61.0741%	58.2593%	58.8519%
2.0	65.1111%	58.8889%	58.3704%	58.2222%	57.4444%
5.0	**65.1852%**	57.5926%	56.7407%	57.4444%	56.7407%

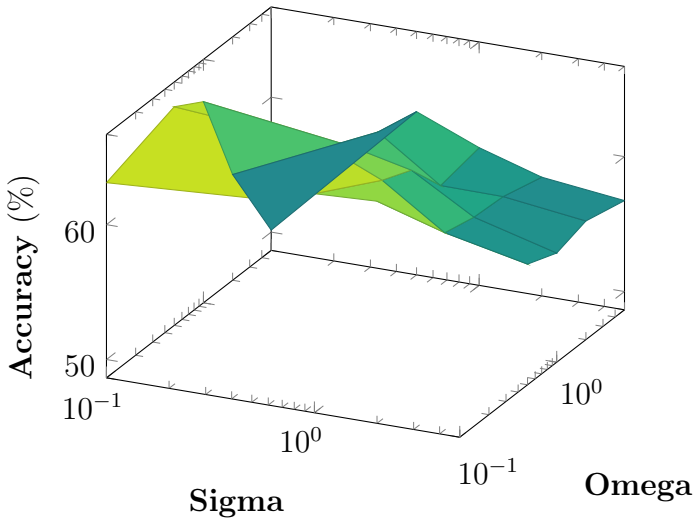


Figure 4-8: Accuracy dependence of the Support Vector Machine on various PUK kernel parameters.

chosen out of the set. For example, the *Fist Poses* are all similar enough to one another that a VR application would pick just one of these poses should they need a *Fist Pose*. Based on this assumption, one could re-run some of the parameter tuning experiments with certain poses grouped to form a single pose class. The grouping of these poses are as follows:

- All *Fist Poses* are grouped under *Classic Fist*.
- *ASL-R* and *Inverse ASL-R* have been grouped under *ASL-R*.
- All *Thumbs-Up Poses* are grouped under *Thumbs-Up*. This was already done for previous experiments.

Table 4.12: Initial results for the Support Vector Machine with varied polynomial kernel parameters.

Exponent	Recognition Accuracy	Average Latency	Average Training Time
0.1	53.9259%	12.477ms	1.214s
0.5	53.2222%	9.0000ms	1.228s
1.0	56%	**0.0939ms**	**1.129s**
1.5	**58.1481%**	11.0654ms	2.614s
2.0	56.2963%	2.9500ms	4.026s
3.0	57%	11.0441ms	14.074s
5.0	54.7037%	11.9457ms	123.429s

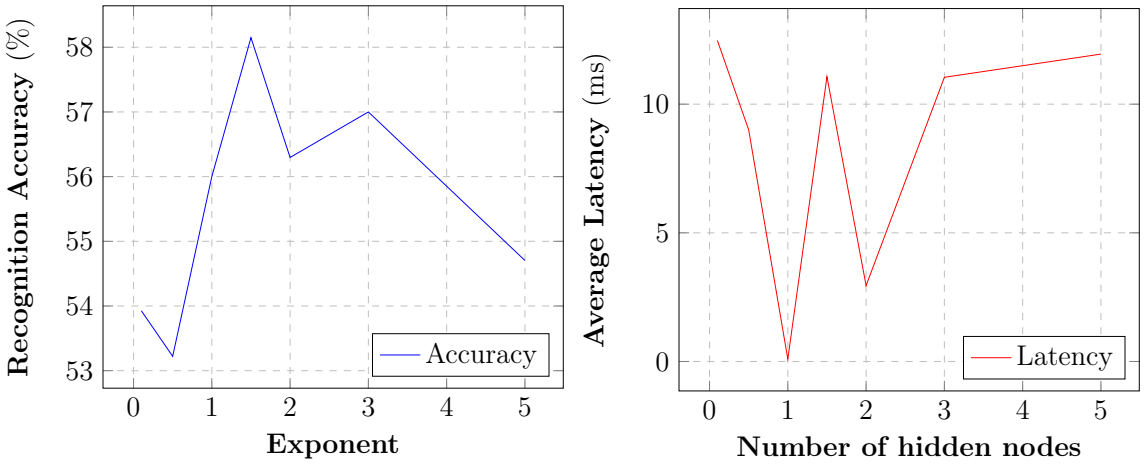


Figure 4-9: Accuracy and latency dependence on the exponent factor of the Support Vector Machine.

4.3.4.1 k-Nearest Neighbour

With a value of $k = 1$, the following metrics with grouped poses were measured:

Recognition Accuracy: 73.2963%

Average Latency: 0.7351ms

4.3.4.2 Artificial Neural Network

Raw data was used as the input vector for the network, with one hundred nodes in a single hidden layer, with a learning rate of 0.3. All other parameters were kept the same as in the previous experiments. The following metrics were calculated after grouping poses:

Recognition Accuracy: 73.5185%

Average Latency: 0.4983ms

Average Training Time: 176.527s

4.3.4.3 Support Vector Machine (PUK)

Using the PUK kernel with *Sigma = 5.0* and *Omega = 0.1*, a complexity constant of 100.0 and with processed input features, the following metrics were obtained after grouping poses:

Recognition Accuracy: 75.2222%

Average Latency: 16.591ms

Average Training Time: 2.385s

4.3.4.4 Support Vector Machine (Linear Polynomial)

Using the Linear Polynomial kernel with a complexity constant of 100.0 and with processed input features, the following metrics were obtained after grouping poses:

Recognition Accuracy: 70.3704%

Average Latency: 0.0550ms

Average Training Time: 1.169s

It is clear that there is another significant improvement in accuracies after pose grouping. However, accuracies between 70% and 76% are still not remarkable. This could be due to the fact that there still exist several poses that are similar to one another, and any slight error in the input data from the LMC will cause classification errors.

Chapter 5

Results and Analysis

This chapter presents the results of the pose recognition experiments described in Chapter 4. The results show that the techniques are not able to classify the poses with a high enough accuracy, primarily due to occlusion issues affecting the input data. Therefore, a smaller reliable set was empirically derived using a novel algorithm, which utilized a confusion matrix from the machine learning experiments as well as a table of Hamming Distances between pose types. This improved the recognition accuracy above 99%, making this set more suitable for real-world use.

5.1 Experimental Results and Analysis

This section presents the results of the three experiments described in the previous chapter. Three classifiers were tested over the three experiments. These classifiers are the k-nearest neighbour, neural network, and support vector machine algorithms. Two different kernels of the support vector machine were tested, namely the PUK kernel and linear kernel.

5.1.1 Orientation-Independent Experiment

Each classifier was tested on a dataset modified such that hand orientation does not matter. Table 5.1 displays the results of this experiment.

Table 5.1: Results for the Orientation-Independent Experiment.

Classifier	Average Accuracy	Average Latency	Average Training Time
k-Nearest Neighbour	66.6667%	0.7835ms	**0s**
Artificial Neural Network	63.0980%	0.5874ms	145.691s
Support Vector Machine (PUK)	**70.3922%**	31.5743ms	2.525s
Support Vector Machine (Linear)	59.098%	**0.0727ms**	2.605s

Tables 5.2 through 5.5 are the confusion matrices for each of the four classifiers tested, where higher classification counts are shown in deeper shades of red.

Table 5.2: k-Nearest Neighbour confusion matrix for the Orientation-Independent Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	51	9	6	7	7	0	5	0	0	2	1	0	0	0	0	0	0	1	1	4	1	0	5
Classic Fist = B	10	42	16	9	7	0	1	0	0	0	1	0	0	0	1	0	0	4	1	5	0	1	2
Hidden Thumb = C	4	12	52	18	8	0	2	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0
ASL-M = D	5	5	19	44	15	0	3	0	0	0	0	0	0	0	2	0	0	0	1	4	0	0	2
ASL-N = E	5	6	11	13	50	0	0	0	0	2	0	1	0	0	1	0	0	3	2	3	0	0	3
Point = F	0	0	0	0	0	77	8	0	0	4	0	0	0	0	2	0	0	0	1	0	4	4	0
Index Forward = G	3	2	3	4	2	10	52	0	0	0	0	0	0	2	1	0	1	5	2	4	4	3	2
Open Hand = H	0	0	0	0	0	0	0	69	7	1	1	3	17	1	0	0	0	0	0	0	0	0	1
Neutral Hand = I	0	0	0	0	0	0	0	8	57	0	0	6	5	16	2	2	0	0	3	0	1	0	0
ASL-B = J	0	0	1	0	2	0	2	2	0	57	14	8	1	2	4	1	0	0	0	0	3	3	0
Flat Hand = K	1	0	0	0	0	0	2	1	1	6	71	7	3	1	2	1	0	0	2	0	1	1	0
Thumb-Middle Group = L	0	0	0	0	0	0	2	1	8	8	10	58	1	1	2	0	0	0	6	0	1	2	0
Spok = M	0	0	0	0	0	0	1	11	9	1	1	1	66	6	0	0	1	0	1	0	0	1	1
Claw = N	0	0	0	0	0	1	0	2	14	0	1	3	4	65	4	2	0	0	2	0	1	1	0
ASL-C = O	0	0	0	1	0	1	0	1	5	1	0	1	1	7	68	0	1	1	8	1	1	1	1
OK-Pose = P	1	0	0	0	0	0	0	0	0	0	1	1	0	2	0	86	3	1	1	2	2	0	0
Middle OK-Pose = Q	0	0	0	0	0	3	0	0	2	0	0	1	0	1	1	2	86	1	3	0	0	0	0
Pinch = R	2	2	1	2	3	1	6	0	0	0	0	0	0	1	0	0	1	67	5	7	1	0	1
Finger Purse = S	3	0	1	1	2	0	2	0	1	0	2	2	0	0	5	2	0	2	69	5	1	0	2
ASL-O = T	2	6	3	5	5	0	2	0	1	0	0	0	0	0	4	1	0	3	5	62	1	0	0
ASL-R = U	1	1	1	0	0	5	4	0	1	2	1	2	0	0	1	0	3	0	0	0	55	22	1
Inverse ASL-R = V	0	2	0	0	0	8	3	1	0	1	1	3	0	1	0	0	2	0	2	0	19	56	1
Thumbs-Up = W	1	3	2	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	340

In all the confusion matrices, the *Thumbs-Up Poses* were often misclassified as one another. The *Thumbs-Up Poses* were also frequently misclassified as the *ASL-O* and *Thumbs-Up* poses. The *Point* pose had one of the highest accuracies, where most errors involved it being misclassified as the *Index Forward* pose. A very high number of misclassifications occurred between the *Spok* and *Open Hand* poses. The *OK-Pose* had a very high classification accuracy compared to other poses, with the highest average accuracy of 96% belonging to the *Thumbs-Up* pose. The *ASL-R* and *Inverse ASL-R* poses were often confused with one another.

Table 5.3: Artificial Neural Network confusion matrix for the Orientation-Independent Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	38	9	16	6	7	0	4	1	1	0	1	0	0	0	1	0	0	7	1	2	0	0	6
Classic Fist = B	9	33	14	11	15	0	5	0	1	2	0	0	0	0	0	0	0	1	0	6	0	0	3
Hidden Thumb = C	7	16	34	16	9	0	2	1	0	0	0	0	0	0	0	0	0	3	3	4	1	0	4
ASL-M = D	4	8	16	47	11	0	0	0	0	0	0	0	0	0	1	1	0	3	1	6	0	0	2
ASL-N = E	4	9	10	20	41	2	2	1	1	0	1	0	0	0	0	0	0	2	2	4	0	0	1
Point = F	0	0	2	0	0	74	8	0	0	1	0	1	1	0	0	0	2	0	0	1	1	8	1
Index Forward = G	1	1	0	6	4	11	60	1	0	2	0	0	0	0	0	0	0	6	3	3	0	1	1
Open Hand = H	0	0	0	0	0	0	0	65	0	0	4	1	28	1	0	0	0	0	0	0	0	1	0
Neutral Hand = I	1	0	0	0	0	1	0	3	57	4	4	3	6	13	3	2	1	0	0	0	0	0	2
ASL-B = J	2	1	1	1	0	0	0	3	4	54	11	13	0	0	0	1	1	0	1	0	4	1	2
Flat Hand = K	1	0	0	0	0	0	0	3	4	4	71	7	6	0	1	0	0	0	1	0	0	0	2
Thumb-Middle Group = L	0	0	0	0	0	2	0	2	5	21	14	37	3	0	1	3	1	0	6	1	4	0	0
Spok = M	0	0	0	0	0	1	0	20	0	2	6	1	65	2	1	0	1	0	1	0	0	0	0
Claw = N	0	0	0	0	0	1	0	0	9	2	0	1	0	70	14	0	0	0	0	0	0	0	3
ASL-C = O	2	0	1	1	0	0	0	0	4	0	0	1	1	20	62	0	0	0	3	1	0	0	4
OK-Pose = P	0	0	1	0	0	0	0	0	1	1	1	2	0	0	0	85	4	0	2	0	2	0	1
Middle OK-Pose = Q	0	0	0	0	0	3	0	2	1	1	0	1	0	1	1	3	83	0	2	0	0	2	0
Pinch = R	3	3	2	6	3	1	2	0	0	0	0	0	0	0	0	0	0	67	6	6	0	1	0
Finger Purse = S	3	1	1	2	1	0	3	2	3	1	1	0	2	0	0	3	1	4	63	4	1	3	1
ASL-O = T	4	5	4	2	2	0	2	0	1	0	0	0	0	0	1	0	0	5	7	62	0	0	5
ASL-R = U	0	1	0	1	0	8	4	0	1	2	0	1	0	0	0	1	2	1	3	2	51	22	0
Inverse ASL-R = V	1	0	0	1	1	10	1	0	1	1	0	4	0	1	0	1	2	0	0	1	21	53	1
Thumbs-Up = W	3	1	3	1	0	1	0	2	0	0	0	0	0	0	0	0	0	1	0	1	0	0	337

Table 5.4: Support Vector Machine (Pearson VII function kernel) confusion matrix for the Orientation-Independent Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	56	12	4	6	4	0	2	0	0	0	1	0	0	0	0	1	0	1	1	4	0	0	8
Classic Fist = B	8	40	19	6	10	0	1	0	0	1	0	1	0	0	1	0	0	4	0	7	0	0	2
Hidden Thumb = C	4	16	51	16	5	0	1	0	0	0	0	0	0	0	0	0	0	1	2	4	0	0	0
ASL-M = D	5	7	16	52	10	0	2	0	0	0	0	0	0	0	1	0	0	0	2	1	0	0	4
ASL-N = E	6	8	8	14	51	0	1	0	0	1	0	1	0	0	1	0	0	3	2	2	0	0	2
Point = F	0	0	0	0	0	84	6	0	0	1	0	0	1	0	0	0	0	0	0	1	3	4	0
Index Forward = G	5	3	4	1	1	10	59	0	0	2	0	1	1	0	4	0	0	2	3	1	0	2	1
Open Hand = H	0	0	0	0	0	0	0	79	6	1	1	2	10	0	0	0	1	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	1	0	8	63	1	4	3	2	10	3	2	0	0	2	0	1	0	0
ASL-B = J	1	0	1	0	2	1	1	0	0	65	10	7	1	1	1	1	0	0	2	0	3	2	1
Flat Hand = K	1	0	0	0	0	0	0	0	3	5	78	8	2	1	0	0	0	0	1	0	0	0	1
Thumb-Middle Group = L	0	0	0	0	0	1	0	2	2	9	10	61	1	2	0	0	0	0	9	0	3	0	0
Spok = M	0	0	0	0	0	0	0	12	7	0	6	0	67	3	1	0	2	0	0	0	1	0	1
Claw = N	0	0	0	0	0	1	0	2	16	0	0	2	1	69	4	1	1	0	2	0	0	1	0
ASL-C = O	0	0	0	0	0	1	1	0	5	2	0	1	4	3	74	0	0	0	4	5	0	0	0
OK-Pose = P	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	94	2	0	1	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	0	2	1	0	0	0	0	1	0	3	87	1	2	0	1	2	0
Pinch = R	4	2	0	1	1	1	4	0	0	0	0	0	0	1	0	0	0	71	5	9	1	0	0
Finger Purse = S	2	0	1	1	2	0	4	0	1	1	1	2	2	0	5	2	0	1	68	5	1	0	1
ASL-O = T	4	7	2	4	4	0	4	0	0	0	0	0	0	0	3	1	0	2	3	64	0	0	2
ASL-R = U	0	1	0	0	1	9	3	0	0	1	1	1	0	1	0	1	1	0	0	0	58	22	0
Inverse ASL-R = V	0	0	0	0	0	13	1	1	0	1	1	2	0	1	0	0	0	0	1	0	17	61	1
Thumbs-Up = W	0	2	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	343

Table 5.5: Support Vector Machine (linear kernel) confusion matrix for the Orientation-Independent Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	31	10	15	10	6	0	2	0	0	0	0	0	1	0	0	1	0	6	3	6	0	0	9
Classic Fist = B	14	22	13	15	10	0	2	0	0	0	0	1	0	0	0	0	0	4	1	14	1	0	3
Hidden Thumb = C	10	12	29	25	10	0	1	0	0	0	0	1	0	0	0	0	0	0	0	7	1	0	4
ASL-M = D	3	9	28	38	8	0	1	0	0	0	0	0	0	0	1	0	0	0	1	8	0	0	3
ASL-N = E	11	9	21	23	18	1	4	0	0	0	0	0	0	0	1	0	0	1	1	5	0	1	4
Point = F	1	0	0	1	0	85	7	0	0	0	0	1	1	0	0	0	0	0	0	0	0	4	0
Index Forward = G	3	0	3	4	2	20	44	0	3	1	1	1	0	0	1	0	0	1	6	4	2	2	2
Open Hand = H	0	0	0	0	0	0	0	82	4	1	0	1	10	0	0	0	1	0	0	0	0	0	1
Neutral Hand = I	0	0	0	0	0	1	2	10	50	4	4	3	3	14	3	2	0	0	1	1	1	1	0
ASL-B = J	2	0	0	1	1	1	2	0	2	65	14	3	1	1	0	1	0	0	2	0	3	0	1
Flat Hand = K	1	0	0	0	0	0	0	0	4	8	77	4	2	1	0	1	0	0	1	0	0	0	1
Thumb-Middle Group = L	0	0	0	0	0	1	2	0	6	21	16	36	1	2	1	0	0	0	8	0	4	2	0
Spok = M	0	0	0	0	0	0	0	9	8	1	3	0	71	4	1	0	2	0	0	0	0	0	1
Claw = N	0	0	0	0	0	1	0	4	24	0	0	1	3	53	8	1	1	0	2	0	1	1	0
ASL-C = O	1	0	0	0	0	0	3	0	4	3	0	0	3	5	72	0	0	0	5	2	0	0	2
OK-Pose = P	1	0	0	0	0	0	0	0	0	0	1	1	0	1	0	89	4	1	1	1	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	2	0	1	2	1	0	0	0	5	0	4	80	2	0	0	2	1	0
Pinch = R	5	8	1	1	0	3	5	0	0	0	0	0	0	1	0	0	0	67	5	4	0	0	0
Finger Purse = S	2	1	3	0	2	0	9	0	2	1	0	3	2	1	2	2	0	0	63	5	0	2	0
ASL-O = T	5	7	9	10	3	0	4	0	0	2	0	0	0	0	2	0	0	6	6	45	0	0	1
ASL-R = U	0	2	0	1	1	8	3	0	0	3	1	1	0	2	0	0	1	0	1	0	30	46	0
Inverse ASL-R = V	0	0	0	0	0	14	1	1	1	1	1	0	0	0	0	0	0	1	1	0	47	32	0
Thumbs-Up = W	5	0	1	3	1	1	1	4	1	0	0	0	0	1	2	0	0	0	0	2	0	0	328

5.1.2 Requested Orientation Experiment

As listed in Section 4.2.2, each classifier was tested on a dataset modified such that hand orientation does not matter. Table 5.6 displays the results of this experiment.

Table 5.6: Results for the Requested Orientation Experiment.

Classifier	Average Accuracy	Average Latency	Average Training Time
k-Nearest Neighbour	76.6087%	0.4037ms	**0s**
Artificial Neural Network	**88.1739%**	0.5845ms	76.77s
Support Vector Machine (PUK)	81.3913%	17.2373ms	0.725s
Support Vector Machine (Linear)	78.6087%	**0.0721ms**	0.609s

Table 5.7: k-Nearest Neighbour confusion matrix for the Requested Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	17	2	10	3	10	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
Classic Fist = B	0	39	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	5	0	0	0
Hidden Thumb = C	12	1	12	4	19	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
ASL-M = D	1	5	4	35	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
ASL-N = E	10	0	18	5	15	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	0	0	0	0	49	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Index Forward = G	2	0	2	0	5	3	23	0	3	0	1	0	0	1	0	0	0	2	6	0	2	0	0
Open Hand = H	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	2	2	24	0	1	1	0	13	3	0	1	0	3	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	1	0	3	2	38	3	0	0	0	0	0	0	3	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	0	0	1	1	1	43	0	1	0	0	0	0	2	1	0	0	0
Spok = M	0	0	0	0	0	0	0	1	0	0	0	0	48	1	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	1	0	0	0	0	49	0	0	0	0	0	0	0	0	0
ASL-C = O	0	0	0	0	0	0	2	0	3	0	0	0	1	6	33	0	0	0	5	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0
Middle OK-Pose = Q	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	45	0	0	0	1	0	0
Pinch = R	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	47	0	0	0	0	0
Finger Purse = S	0	1	2	0	0	0	6	0	5	0	3	2	0	0	6	0	0	1	23	0	1	0	0
ASL-O = T	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0
ASL-R = U	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	46	2	0
Inverse ASL-R = V	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0
Thumbs-Up = W	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48

In these confusion matrices, similar patterns can be seen across the kNN, SVM-PUK and SVM-Lin classifiers, while the ANN has a completely different pattern. Furthermore, the ANN has a significantly higher accuracy than the other classifiers at 88%.

Table 5.8: Artificial Neural Network confusion matrix for the Requested Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	41	3	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1
Classic Fist = B	5	27	5	6	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0
Hidden Thumb = C	1	2	37	5	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ASL-M = D	1	1	5	37	3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0
ASL-N = E	0	2	0	2	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	0	0	0	0	48	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Index Forward = G	0	1	0	0	0	1	47	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Open Hand = H	0	0	0	0	0	0	0	48	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	0	1	46	0	0	1	1	1	0	0	0	0	0	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	44	2	4	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	0	0	1	7	4	29	1	0	0	3	1	0	3	0	0	1	0
Spok = M	0	0	0	0	0	0	0	2	2	0	3	0	43	0	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	2	0	0	0	1	47	0	0	0	0	0	0	0	0	0
ASL-C = O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	47	0	0	0	0	0	0
Pinch = R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0
Finger Purse = S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	46	2	0	0	0
ASL-O = T	0	1	0	1	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	44	0	0	1
ASL-R = U	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41	8	0
Inverse ASL-R = V	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	46	0
Thumbs-Up, Fist-In = W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50

Table 5.9: Support Vector Machine (Pearson VII function kernel) confusion matrix for the Requested Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	23	1	11	0	11	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Classic Fist = B	0	40	0	5	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	3	0	0	0
Hidden Thumb = C	9	1	13	3	20	0	3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
ASL-M = D	1	7	3	36	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
ASL-N = E	15	0	16	3	14	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	1	0	0	0	49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Index Forward = G	5	0	1	0	3	3	23	0	2	0	1	0	0	0	3	0	0	1	6	0	2	0	0
Open Hand = H	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	0	2	31	0	2	2	1	6	5	0	0	0	1	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	0	0	0	1	48	1	0	0	0	0	0	0	0	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	0	0	0	1	0	47	0	0	0	0	0	0	2	0	0	0	0
Spok = M	0	0	0	0	0	0	0	0	1	0	0	0	48	1	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	1	0	0	0	0	49	0	0	0	0	0	0	0	0	0
ASL-C = O	0	0	0	0	0	0	2	0	4	0	0	0	0	2	39	0	0	0	3	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	46	0	1	0	1	0	0
Pinch = R	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	47	0	0	0	0	0
Finger Purse = S	0	1	0	0	0	0	6	0	3	0	1	1	0	0	0	0	0	1	36	0	1	0	0
ASL-O = T	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0	0	0
ASL-R = U	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0	0
Inverse ASL-R = V	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0
Thumbs-Up = W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50

Table 5.10: Support Vector Machine (linear kernel) confusion matrix for the Requested Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	28	1	12	0	4	0	3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Classic Fist = B	0	31	1	7	1	0	0	0	0	0	0	1	0	0	0	0	0	2	0	7	0	0	0
Hidden Thumb = C	6	0	19	4	17	0	3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
ASL-M = D	0	6	3	36	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0
ASL-N = E	7	0	16	3	21	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	1	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Index Forward = G	5	1	2	0	4	4	23	0	3	0	0	0	0	0	1	0	0	0	4	0	3	0	0
Open Hand = H	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	0	2	33	0	2	1	1	7	3	0	0	0	1	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	49	1	0	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	0	0	1	4	43	2	0	0	0	0	0	0	0	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	1	0	0	4	3	40	0	1	0	0	0	0	1	0	0	0	0
Spok = M	0	0	0	0	0	0	0	0	2	0	0	0	48	0	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	5	0	0	1	0	43	0	0	0	0	1	0	0	0	0
ASL-C = O	0	1	0	0	0	0	1	0	6	0	0	0	0	1	41	0	0	0	0	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	0	45	0	1	0	1	0	0
Pinch = R	0	3	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	42	0	2	0	0	0
Finger Purse = S	0	0	0	0	0	0	5	0	5	0	1	3	0	1	0	0	0	1	33	1	0	0	0
ASL-O = T	0	6	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	40	0	0	0
ASL-R = U	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	45	2	0
Inverse ASL-R = V	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	47	0
Thumbs-Up = W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	49

5.1.3 Thumbs-Orientation Experiment

As listed in Section 4.2.2, each classifier was trained on all available data, and only the *Thumbs-Up* poses were tested. Table 5.11 displays the results of this experiment.

Table 5.11: Results for the Thumbs-Orientation Experiment.

Classifier	Average Accuracy	Average Latency	Average Training Time
k-Nearest Neighbour	**96.5714%**	0.6547ms	**0s**
Artificial Neural Network	92.8571%	0.5314ms	141.305s
Support Vector Machine (PUK)	96.0%	44.8328ms	2.475s
Support Vector Machine (Linear)	92.8571%	**0.1419ms**	2.177s

Table 5.12: k-Nearest Neighbour confusion matrix for the Thumbs-Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W1	W2	W3	W4	W5	W6	W7
ASL-A = A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Classic Fist = B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hidden Thumb = C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-M = D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-N = E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Index Forward = G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Open Hand = H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Spok = M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-C = O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pinch = R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Finger Purse = S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-O = T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-R = U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Inverse ASL-R = V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumbs-Up = W1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	0	0	2	0	0	0
Thumbs-Out = W2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	48	1	0	0	0	0
Thumbs-In = W3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0	0	0	0
Thumbs-Down = W4	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	47	0	0	0
Thumbs-Back = W5	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0
Thumbs-Up, Fist-In = W6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0
Thumbs-Down, Fist-In = W7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50

In all the classifiers, high accuracies of over 90% were recorded. Roughly half of misclassifications involved a *Thumbs-Up* pose being recognized as a differently oriented *Thumbs-Up* pose. Most of the other misclassifications were caused by a *Thumbs-Up* pose being recognized as a *Fist Pose*.

Table 5.13: Artificial Neural Network confusion matrix for the Thumbs-Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W1	W2	W3	W4	W5	W6	W7
ASL-A = A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Classic Fist = B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hidden Thumb = C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-M = D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-N = E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Index Forward = G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Open Hand = H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Spok = M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-C = O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pinch = R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Finger Purse = S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-O = T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-R = U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Inverse ASL-R = V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumbs-Up = W1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	47	1	0	0	0	0	0
Thumbs-Out = W2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	48	1	0	0	0	0
Thumbs-In = W3	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	46	0	0	0	0
Thumbs-Down = W4	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	44	0	0	1
Thumbs-Back = W5	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	45	1	0
Thumbs-Up, Fist-In = W6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	48	0
Thumbs-Down, Fist-In = W7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	47

Table 5.14: Support Vector Machine (Pearson VII function kernel) confusion matrix for the Thumbs-Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W1	W2	W3	W4	W5	W6	W7
ASL-A = A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Classic Fist = B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hidden Thumb = C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-M = D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-N = E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Index Forward = G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Open Hand = H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Spok = M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-C = O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pinch = R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Finger Purse = S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-O = T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-R = U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Inverse ASL-R = V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumbs-Up = W1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	0	0	2	0	0	0
Thumbs-Out = W2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	48	1	0	0	0	0
Thumbs-In = W3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	48	0	0	0	0
Thumbs-Down = W4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	47	0	0	0
Thumbs-Back = W5	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	0
Thumbs-Up, Fist-In = W6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0
Thumbs-Down, Fist-In = W7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50

Table 5.15: Support Vector Machine (linear kernel) confusion matrix for the Thumbs-Orientation Experiment.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W1	W2	W3	W4	W5	W6	W7
ASL-A = A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Classic Fist = B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hidden Thumb = C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-M = D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-N = E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Point = F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Index Forward = G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Open Hand = H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-B = J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Flat Hand = K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumb-Middle Group = L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Spok = M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Claw = N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-C = O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OK-Pose = P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pinch = R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Finger Purse = S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-O = T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ASL-R = U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Inverse ASL-R = V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thumbs-Up = W1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	1	0	1	0	0	0
Thumbs-Out = W2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	47	1	0	0	0	0
Thumbs-In = W3	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	47	0	0	0	0
Thumbs-Down = W4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	2	0	0	43	0	0	2
Thumbs-Back = W5	3	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44	0	0
Thumbs-Up, Fist-In = W6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0
Thumbs-Down, Fist-In = W7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	48

5.2 Pose Similarity and Simplification

In Section 4.3.4, potentially problematic hand poses were grouped together to illustrate the positive effect this would have on recognition rates. However, the poses that were grouped were not chosen through a clear and systematic means, and thus a more thorough process of pose selection is required. In order to simplify the pose set, a grouping approach could be taken where similar poses are grouped together under a single pose name, as in Section 4.3.4. Another approach would be to select a certain number of poses that are well separated from one another, and discard the rest. A particular problem with the grouping approach can be best described by an example. Suppose that the pose set contains poses A, B, and C, and that pose A is similar to B, and B similar to C. By taking a grouping approach, there is no clear means of creating two separate groups. By grouping A and B together, half of the group will be similar to pose C. If a selection approach is taken instead, B could be eliminated from the dataset, creating a dataset of two clearly separated poses: A and C. Thus, in simplifying this set, the selection approach shall be used, and problematic poses shall be discarded.

This subsection first illustrates the similarity between poses by classifying them and displaying the results through a confusion matrix. Then, similarity is measured more formally by comparing the notation strings given by Choi et al.'s taxonomy [8] of each of the poses in the dataset. Finally, using these similarity measures, certain poses will be selected to form a reliable pose set for further testing.

5.2.1 Measuring Similarity via Confusion Matrix

A simple way to illustrate which poses are often classified as one another is through a confusion matrix. Table 5.16 is such a matrix, with high classification occurrences highlighted in red. This matrix is equivalent to the one found in Table 5.4, where the SVM-PUK classifier was tested in the orientation-independent experiment.

From this table, it becomes evident that the *Fist Poses* (Poses A through E) are

Table 5.16: SVM-PUK confusion matrix of orientation-independent data in the benchmark pose set.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	56	12	4	6	4	0	2	0	0	0	1	0	0	0	0	1	0	1	1	4	0	0	8
Classic Fist = B	8	40	19	6	10	0	1	0	0	1	0	1	0	0	1	0	0	4	0	7	0	0	2
Hidden Thumb = C	4	16	51	16	5	0	1	0	0	0	0	0	0	0	0	0	0	1	2	4	0	0	0
ASL-M = D	5	7	16	52	10	0	2	0	0	0	0	0	0	0	1	0	0	0	2	1	0	0	4
ASL-N = E	6	8	8	14	51	0	1	0	0	1	0	1	0	0	1	0	0	3	2	2	0	0	2
Point = F	0	0	0	0	0	84	6	0	0	1	0	0	1	0	0	0	0	0	0	1	3	4	0
Index Forward = G	5	3	4	1	1	10	59	0	0	2	0	1	1	0	4	0	0	2	3	1	0	2	1
Open Hand = H	0	0	0	0	0	0	0	79	6	1	1	2	10	0	0	0	1	0	0	0	0	0	0
Neutral Hand = I	0	0	0	0	0	1	0	8	63	1	4	3	2	10	3	2	0	0	2	0	1	0	0
ASL-B = J	1	0	1	0	2	1	1	0	0	65	10	7	1	1	1	1	0	0	2	0	3	2	1
Flat Hand = K	1	0	0	0	0	0	0	0	3	5	78	8	2	1	0	0	0	0	1	0	0	0	1
Thumb-Middle Group = L	0	0	0	0	0	1	0	2	2	9	10	61	1	2	0	0	0	0	9	0	3	0	0
Spok = M	0	0	0	0	0	0	0	12	7	0	6	0	67	3	1	0	2	0	0	0	1	0	1
Claw = N	0	0	0	0	0	1	0	2	16	0	0	2	1	69	4	1	1	0	2	0	0	1	0
ASL-C = O	0	0	0	0	0	1	1	0	5	2	0	1	4	3	74	0	0	0	4	5	0	0	0
OK-Pose = P	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	94	2	0	1	0	0	0	0
Middle OK-Pose = Q	0	0	0	0	0	0	0	2	1	0	0	0	0	1	0	3	87	1	2	0	1	2	0
Pinch = R	4	2	0	1	1	1	4	0	0	0	0	0	0	1	0	0	0	71	5	9	1	0	0
Finger Purse = S	2	0	1	1	2	0	4	0	1	1	1	2	2	0	5	2	0	1	68	5	1	0	1
ASL-O = T	4	7	2	4	4	0	4	0	0	0	0	0	0	0	3	1	0	2	3	64	0	0	2
ASL-R = U	0	1	0	0	1	9	3	0	0	1	1	1	0	1	0	1	1	0	0	0	58	22	0
Inverse ASL-R = V	0	0	0	0	0	13	1	1	0	1	1	2	0	1	0	0	0	0	1	0	17	61	1
Thumbs-Up = W	0	2	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	343

all often misclassified as one another. Additionally, the *ASL-O* and *Index Forward* poses are often confused with the *Fist Poses* and vice versa.

The *Point* pose has a high classification accuracy, however when the index finger is dipped forward to form the *Index Forward* pose, accuracy drops significantly.

In the *Open-Palm* poses (poses H through O), significant misclassification occurred, but was not as widespread as the *Fist Poses*. The *Open Hand* pose had the best accuracy, and was sometimes confused with the *Spok* and *Neutral Hand* poses. The *Neutral Hand* pose exists as an intermediate step in poses between the *Open Hand* and *Claw* poses, and is thus misclassified as each often. The *ASL-B*, *Flat Hand*, and *Thumb-Middle Group* poses are often classified correctly, albeit with some misclassifications as one another. The *Flat Hand* pose was classified correctly the second most often in the *Open-Palm* pose group. The *Thumb-Middle Group* pose had the lowest accuracy with only 61% correct classifications, where it was sometimes even classified as the *Finger Purse* and *ASL-R* poses. The *Spok* pose had a high accuracy, but was sometimes confused with the *Open Hand*, *Flat Hand* and *Neutral Hand* poses. The *Claw* pose was often confused with the *Neutral Hand* pose, and vice versa. *ASL-C* had a high accuracy of 74%, but was sometimes classified as a *Neutral Hand*.

Amongst the *Finger Loop* poses (poses P through T), the *OK-Pose* had the highest accuracy of 94%, and was only confused with the *Middle OK-Pose* twice. These two poses, although being quite similar to one another, were well separated. The *Pinch*, *Finger Purse*, and *ASL-O* poses were all sometimes misclassified as a *Fist Poses* and the *Index Forward* pose. Additionally, these three poses are often misclassified as one another, leading to a low recognition accuracy for all of them.

The *ASL-R* and *Inverse ASL-R* poses (poses U and V) both have a low recognition accuracy rate. Both were misclassified as one another very often, and were regularly misclassified as the *Point* pose.

The *Thumbs-Up Poses* had a high recognition accuracy of 343 out of 350 (98%) with most errors occurring when they were incorrectly recognized as *Fist Poses*.

5.2.2 Measuring Similarity via String Distance

Since the notation strings are all of the same length, a simple means by which similarity could be measured is through Hamming Distance. Hamming Distance is the number of occurrences of differences between two strings of equal lengths, and was introduced in [19]. For example, the *ASL-R* and *Inverse ASL-R* poses have the notation strings 61166;2252-423 and 61166;2252-523 respectively, giving them a Hamming Distance of 1. The *ASL-R* and *Open Hand* poses have notation strings 61166;2252-423 and 31111;2222-222, giving them a distance of 6. From this example, one could predict that the Hamming Distance between two poses is low if the poses appear to be visually similar to one-another, and high if they're visually different. In order to verify this prediction, Table 5.17 illustrates the Hamming Distances between all poses.

When comparing the distances seen in this table to the misclassification errors in Table 5.16, some observations can be made about several groups of poses.

The *Fist Poses* (Poses A through E) maintain their similarity with one another, with distances ranging from 3 to 6. If one were to compare the *ASL-M* pose with the *ASL-A* pose, one could argue that a distance of 6 is too large for poses that are

Table 5.17: Heatmap of Hamming distances between poses.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	0	3	4	6	5	5	5	9	9	6	4	7	7	9	6	9	10	3	9	7	7	7	2
Classic Fist = B	3	0	3	6	5	3	3	10	10	6	7	8	8	10	7	10	10	4	9	7	7	7	3
Hidden Thumb = C	4	3	0	3	3	5	4	11	11	7	8	8	9	11	8	11	11	5	9	8	7	7	4
ASL-M = D	6	6	3	0	4	8	7	12	11	10	10	10	11	11	9	10	10	7	10	9	10	10	6
ASL-N = E	5	5	3	4	0	7	6	11	10	9	9	9	9	10	8	9	9	6	10	8	7	7	5
Point = F	5	3	5	8	7	0	2	7	10	5	7	7	7	8	7	10	8	5	10	8	5	5	4
Index Forward = G	5	3	4	7	6	2	0	9	10	7	8	7	9	9	8	10	10	4	9	8	7	7	5
Open Hand = H	9	10	11	12	11	7	9	0	9	4	5	6	2	5	8	6	6	9	12	10	6	6	7
Neutral Hand = I	9	10	11	11	10	10	10	9	0	9	9	9	9	9	9	6	7	9	12	10	10	10	9
ASL-B = J	6	6	7	10	9	5	7	4	9	0	2	4	2	8	5	6	6	6	9	7	5	5	5
Flat Hand = K	4	7	8	10	9	7	8	5	9	2	0	3	3	9	6	6	7	6	9	7	7	7	6
Thumb-Middle Group = L	7	8	8	10	9	7	7	6	9	4	3	0	5	10	8	6	6	8	10	8	8	8	8
Spok = M	7	8	9	11	9	7	9	2	9	2	3	5	0	7	6	6	6	7	10	8	5	5	5
Claw = N	9	10	11	11	10	8	9	5	9	8	9	10	7	0	3	7	7	9	12	5	8	8	8
ASL-C = O	6	7	8	9	8	7	8	8	9	5	6	8	6	3	0	7	7	6	9	2	8	8	5
OK-Pose = P	9	10	11	10	9	10	10	6	6	6	6	6	6	7	7	0	4	9	12	7	9	9	9
Middle OK-Pose = Q	10	10	11	10	9	8	10	6	7	6	7	6	6	7	7	4	0	10	12	7	9	9	9
Pinch = R	3	4	5	7	6	5	4	9	9	6	6	8	7	9	6	9	10	0	6	7	7	7	3
Finger Purse = S	9	9	9	10	10	10	9	12	12	9	9	10	10	12	9	12	12	6	0	9	11	11	9
ASL-O = T	7	7	8	9	8	8	8	10	10	7	7	8	8	5	2	7	7	7	9	0	10	10	7
ASL-R = U	7	7	7	10	7	5	7	6	10	5	7	8	5	8	8	9	9	7	11	10	0	1	6
Inverse ASL-R = V	7	7	7	10	7	5	7	6	10	5	7	8	5	8	8	9	9	7	11	10	1	0	6
Thumbs-Up = W	2	3	4	6	5	4	5	7	9	5	6	8	5	8	5	9	9	3	9	7	6	6	0

visually not that different. This argument is further reinforced by the fact that the distance calculated between the *Point* and *ASL-A* poses is less than the distance of 6, implying that a *Point* is considered to be more similar to the *ASL-A* pose than *ASL-M*. This could be resolved by scaling the distances between poses if there is a significant change in the silhouette of a pose. For example, the difference between a fully curled and fully extended finger should be much larger than that of a fully curled and partially curled finger. Another example of this issue exists between the ASL-A and Flat Hand poses, where an unintuitive distance of 4 is calculated.

The theme of poses with large silhouette differences yet small Hamming Distances is evident throughout the table. The Flat Hand and ASL-A pose have a distance of 4, Point and Classic Fist have a distance of 3, and Index Forward have a distance of 3. Cases of the inverse being true also exist, where a high distance between similar poses is calculated. The Claw and Finger Purse poses have the highest possible distance of 12, and the distances between the Fist Poses and Finger Purse pose vary from 9 to 10.

The examples listed above can be compared to the results in the confusion matrix in Table 5.16. Flat Hand and ASL-A were never misclassified as one-another, yet have a low distance of 4. Conversely, Claw and Finger Purse have a single misclassification

between them, which should never happen between two poses at maximum distance from one another with twenty-two other poses to be chosen from.

A possible solution to this problem is to create a weighted Hamming Distance that would take silhouette changes into account. Where previously any change between notation string elements increases Hamming Distance by one, weighted Hamming Distance would increase the distance according to how much the visual silhouette of the pose is changed. The change in visual silhouette is not objectively measured, but is rather used as a tool to estimate the weights to be assigned. These weightings are depicted in Tables 5.18 and 5.19.

Table 5.18: Finger Pose Distance Weightings.

Finger Pose Notator	**1**	**2**	**3**	**4**	**5**	**6**
Point (Up): 1	0.0					
Point (Forward): 2	1.0	0.0				
Point (Side): 3	1.0	1.0	0.0			
Neutral: 4	0.3	0.5	0.5	0.0		
Bend: 5	1.0	0.5	1.0	0.2	0.0	
Close: 6	2.0	1.5	1.5	1.5	0.7	0.0

Table 5.19: Finger Inter-Relation Distance Weightings.

Finger Inter-Relation Notator	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Neutral: 1	0.0						
Separate: 2	0.2	0.0					
Group: 3	0.6	1.0	0.0				
Cross (i on palm-side of j): 4	0.6	1.0	0.2	0.0			
Cross (j on palm-side of i): 5	0.6	1.0	0.2	0.1	0.0		
Touch: 6	0.6	1.0	0.3	0.3	0.3	0.0	
Loop: 7	2.0	2.0	1.5	0.9	0.9	0.5	0.0

Table 5.20 depicts the Hamming Distances between the poses after the distances were weighted. Weighting the differences between notation strings did solve the aforementioned problems to some degree. The *Pointing Poses* are no longer as similar to the *Fist Poses* as they were before, and the *Flat Hand* and *ASL-A* poses have been further separated. The *Claw* and *Finger Purse* poses were originally calculated to be at maximum distance from one another, after weightings they now have a distance of 9.5. These problems were solved while maintaining the similarities between poses

Table 5.20: Heatmap of weighted Hamming Distances between poses.

Ground Truth (Rows) / Classified As (Cols)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
ASL-A = A	0.0	3.2	4.2	4.8	3.8	7.0	5.7	13.0	8.7	11.0	8.0	9.8	11.0	7.8	4.8	11.0	12.5	2.8	10.3	7.3	9.2	9.2	2.0
Classic Fist = B	3.2	0.0	1.2	3.5	2.5	4.0	2.6	14.5	10.9	10.0	11.2	10.9	12.5	8.5	5.5	11.1	11.1	4.3	10.1	5.3	8.2	8.2	3.5
Hidden Thumb = C	4.2	1.2	0.0	2.3	1.4	5.1	3.6	15.5	11.9	11.0	12.2	11.8	13.5	9.5	6.5	12.1	12.1	5.3	9.4	6.3	7.3	7.3	4.5
ASL-M = D	4.8	3.5	2.3	0.0	2.3	7.4	5.9	15.2	11.0	13.3	12.8	12.4	14.6	9.0	7.4	11.8	11.8	5.9	8.3	7.2	9.6	9.6	5.6
ASL-N = E	3.8	2.5	1.4	2.3	0.0	6.4	4.9	14.2	10.0	12.3	11.8	11.4	12.2	8.0	6.4	10.8	10.8	4.9	9.0	6.2	7.1	7.1	4.6
Point = F	7.0	4.0	5.1	7.4	6.4	0.0	2.0	10.5	8.9	8.0	11.0	9.4	10.5	6.8	5.8	12.1	7.7	5.5	11.3	7.7	6.0	6.0	5.5
Index Forward = G	5.7	2.6	3.6	5.9	4.9	2.0	0.0	12.5	9.5	10.0	11.2	9.4	12.5	7.3	6.3	10.5	9.7	3.8	9.6	6.1	8.0	8.0	6.0
Open Hand = H	13.0	14.5	15.5	15.2	14.2	10.5	12.5	0.0	2.5	4.5	5.0	5.2	2.0	5.0	8.0	4.6	4.6	12.0	12.0	12.0	8.5	8.5	11.0
Neutral Hand = I	8.7	10.9	11.9	11.0	10.0	8.9	9.5	2.5	0.0	4.7	3.9	4.3	3.3	1.8	3.0	3.3	3.5	7.9	7.9	6.8	7.7	7.7	8.5
ASL-B = J	11.0	10.0	11.0	13.3	12.3	8.0	10.0	4.5	4.7	0.0	3.0	4.6	2.5	7.7	4.7	5.5	5.5	9.5	9.5	8.7	6.2	6.2	9.5
Flat Hand = K	8.0	11.2	12.2	12.8	11.8	11.0	11.2	5.0	3.9	3.0	0.0	1.8	3.0	9.0	6.0	5.3	6.8	8.3	8.3	8.5	9.2	9.2	10.0
Thumb-Middle Group = L	9.8	10.9	11.8	12.4	11.4	9.4	9.4	5.2	4.3	4.6	1.8	0.0	4.6	9.2	7.6	5.1	5.7	9.9	8.2	8.0	10.6	10.6	11.6
Spok = M	11.0	12.5	13.5	14.6	12.2	10.5	12.5	2.0	3.3	2.5	3.0	4.6	0.0	7.0	6.0	5.4	5.4	10.0	10.0	10.0	6.7	6.7	9.0
Claw = N	7.8	8.5	9.5	9.0	8.0	6.8	7.3	5.0	1.8	7.7	9.0	9.2	7.0	0.0	3.0	5.6	5.6	7.1	9.5	7.0	7.1	7.1	6.8
ASL-C = O	4.8	5.5	6.5	7.4	6.4	5.8	6.3	8.0	3.0	4.7	6.0	7.6	6.0	3.0	0.0	6.8	6.8	4.1	6.5	4.0	6.3	6.3	3.8
OK-Pose = P	11.0	11.1	12.1	11.8	10.8	12.1	10.5	4.6	3.3	5.5	5.3	5.1	5.4	5.6	6.8	0.0	6.0	9.3	9.3	6.8	10.1	10.1	11.5
Middle OK-Pose = Q	12.5	11.1	12.1	11.8	10.8	7.7	9.7	4.6	3.5	5.5	6.8	5.7	5.4	5.6	6.8	6.0	0.0	11.0	9.3	6.8	10.1	10.1	11.5
Pinch = R	2.8	4.3	5.3	5.9	4.9	5.5	3.8	12.0	7.9	9.5	8.3	9.9	10.0	7.1	4.1	9.3	11.0	0.0	7.5	5.6	7.7	7.7	3.5
Finger Purse = S	10.3	10.1	9.4	8.3	9.0	11.3	9.6	12.0	7.9	9.5	8.3	8.2	10.0	9.5	6.5	9.3	9.3	7.5	0.0	5.5	11.0	11.0	11.0
ASL-O = T	7.3	5.3	6.3	7.2	6.2	7.7	6.1	12.0	6.8	8.7	8.5	8.0	10.0	7.0	4.0	6.8	6.8	5.6	5.5	0.0	10.3	10.3	7.8
ASL-R = U	9.2	8.2	7.3	9.6	7.1	6.0	8.0	8.5	7.7	6.2	9.2	10.6	6.7	7.1	6.3	10.1	10.1	7.7	11.0	10.3	0.0	0.1	7.7
Inverse ASL-R = V	9.2	8.2	7.3	9.6	7.1	6.0	8.0	8.5	7.7	6.2	9.2	10.6	6.7	7.1	6.3	10.1	10.1	7.7	11.0	10.3	0.1	0.0	7.7
Thumbs-Up = W	2.0	3.5	4.5	5.6	4.6	5.5	6.0	11.0	8.5	9.5	10.0	11.6	9.0	6.8	3.8	11.5	11.5	3.5	11.0	7.8	7.7	7.7	0.0

within the same group seen in the unweighted table (Table 5.16). For example, the *Fist Poses* group is still visible as a block of very similar poses. The *Open-Palm Poses* are also able to maintain similarity within the group, with the exception of the *Claw* and *ASL-C* pose. These two poses involve all five fingers bending down in some manner, causing a relatively large distance between themselves and the rest of the *Open-Palm Poses*. The *Finger Touches and Loops Poses* (Poses P through T) do not show as strong a similarity to one another as expected. A pose with a loop between two fingers has a high distance between itself and a pose with a loop between two other fingers. This can be seen in the distance of 6 between the *OK* and *Middle OK* poses. The silhouette of the hand does not change much, however the string distance between these poses is large. This is due to the fact that Hamming Distance does not take adjacent string elements into account, such as the index and middle finger loops in this case. Furthermore, adding additional loops in a pose should not increase the distance by as much as adding in the first loop.

Using a weighted Hamming Distance does not provide a perfect solution to measuring pose similarities, but it does provide a rough picture of which poses are very different from one another and which are not.

5.2.3 Final Pose Selection

In order for a camera to provide an immersive and reliable pose recognition experience in VR, a target of 99% recognition accuracy is desirable. By aggressively discarding problematic poses until the target is reached, a reliable pose set was created to provide researchers and developers a practical set of poses to work with. This set shall be hereon referred to as the reliable pose set. To create this set, the average weighted Hamming distances of each pose is measured. Ideally, the poses with the lowest average distances are systematically removed until the target accuracy is reached. However, the weighted Hamming distance measurement is not a perfect way of measuring pose similarity. As such, some exceptions have to be made.

Firstly, certain key poses in the set should have preference over others and should be kept in the set. For example, if *Classic Fist* and *ASL-M* show that *Classic Fist* has a lower average distance to other poses, an exception would be made and *Classic Fist* should not be discarded. This is because the pose is commonly used in VR applications and the real world, much more so than other similar poses. The other key poses are the *Point*, *OK-Pose*, and *Thumbs-up* poses. Either the *Open Hand* or *Neutral Hand* is also a key pose, but not both. This exception will be referred to as **exception one**.

The second exception occurs where a pose has a high average distance to other poses, yet is misclassified often according to the SVM-PUK confusion matrix (Table 5.16). Any poses that have these characteristics need to be removed early, otherwise they will be problematic later on. Detecting the occurrence of this exception will be by calculating the product of its scaled average Hamming distance and its scaled confusion matrix inaccuracy. The scaled average Hamming distance is a number between zero and one, where one represents the highest Hamming distance amongst other poses in the reliable set, and zero being the lowest. Similarly, the scaled confusion matrix inaccuracy lies between zero and one, where zero represents the pose with the highest accuracy, and one represents the pose with the lowest accuracy. If this factor is above a certain threshold, the pose is removed. This factor will be high whenever there is

a mismatch in what the confusion matrix and the weighted Hamming distance imply about a pose, specifically when the confusion matrix highlights it as problematic while the weighted Hamming shows it to be unproblematic. This exception will be referred to as **exception two**.

More formally, let the benchmark pose set be `Set A` and the reliable set be `Set B`. The creation of the reliable pose set follows the algorithm outlined in Algorithm 1. Following the algorithm, the reliable pose set consists of three poses. These poses are the *Open Hand*, *OK-Pose*, and *Thumbs-Up* poses. The dataset consisting of these three poses at arbitrary orientations achieved an accuracy of 99.45% with the SVM-PUK classifier.

This algorithm could be used to produce other reliable pose sets. For example, the key poses to be kept or the target accuracy could be changed if desired. By changing the `factorThreshold`, one could alter the frequency at which the poses with a high `factor` are removed. Other parameters that could be changed are the input set and classification algorithm.

Input: Set A

Output: Set B

$factorThreshold \leftarrow 0.6$

$setB \leftarrow clone(setA)$

while *SVM-PUK accuracy on Set B < 99%* **do**

    generate new confusion matrix

    generate new Hamming Distance matrix

    if *only key poses remain in Set B* **then**

        ignore exception one

    end

    foreach *pose P in set B* **do**

        foreach *pose Q in set B where Q != P* **do**

            $P.totalDistance \leftarrow P.totalDistance + weightedHamming(P, Q)$

            $P.averageDistance \leftarrow P.totalDistance \div (poseCount(setB) - 1)$

        end

    end

    $mcMin \leftarrow$ smallest misclassification % in confusion matrix

    $mcMax \leftarrow$ largest misclassification % in confusion matrix

    $hamMin \leftarrow$ smallest weighted Hamming distance

    $hamMax \leftarrow$ largest weighted Hamming distance ▷ `Used in below normalization step.`

    foreach *pose P in set B* **do**

        $mc \leftarrow$ misclassification% of pose P in confusion matrix

        $scaledMc \leftarrow (mc - mcMin) \div (mcMax - mcMin)$ ▷ `Outputs [0,1]`

        $ham \leftarrow P.averageDistance$

        $scaledHam \leftarrow (ham - hamMin) \div (hamMax - hamMin)$ ▷ `Outputs [0,1]`

        $P.factor \leftarrow scaledMc \times scaledHam$

    end

    $PoseR \leftarrow$ pose in set B with the highest factor ▷ `Check for valid exception two.`

    while *R is a key pose* **do**

        $R \leftarrow$ pose with the next highest factor ▷ `Ensure validity of exception two pose.`

    end

    if *R.factor >= factorThreshold* **then**

        remove R from set B

        continue to next loop

    end

    $PoseR \leftarrow$ pose with lowest averageDistance in set B ▷ `Check for exception one.`

    while *R is a key pose* **do**

        $R \leftarrow$ pose with next lowest averageDistance

    end

    remove R from set B

    reset all totalDistances

end

return *Set B*

Algorithm 1: The process followed to create the reliable pose set.

Chapter 6

Discussion

6.1 Introduction

This research aimed to explore the effectiveness of using cameras, specifically the LMC, instead of hand-held controllers to provide control through pose recognition in VR applications.

Four objectives were listed as part of the research. First, a taxonomy was required to create the benchmark pose set. The taxonomy by Choi et al. [8] was chosen for this purpose. The second objective was to use the chosen taxonomy along with poses from VR applications to form a benchmark pose set. Five key poses from LMC-based VR applications along with the taxonomy were used to form the benchmark pose set. This pose set allows for researchers to use a common pose set when comparing machine learning algorithms for pose recognition in VR. These poses were performed by twenty-five participants to form the benchmark pose dataset, which was then used for the machine learning experiments.

Evaluating the performance of machine learning classifiers on the benchmark pose dataset was the third objective of the research. No classifier was found to be objectively the best. The SVM-PUK classifier had the highest average accuracy across experiments, yet had the highest average latency of the classifiers. Conversely, the SVM-Lin classifier had the lowest average accuracy, but the lowest average latency. Findings from these experiments include the fact that a 16.4% increase in average

accuracy was observed when poses at the requested orientation were used, and that the LMC is able to distinguish hand orientations better than hand shapes.

The final objective was to form a reliable pose set from the benchmark set. To do this, a novel algorithm was developed that used a Hamming Distance matrix and a confusion matrix to systematically remove poses from the benchmark set until an accuracy of over 99% was achieved. As a result, the reliable pose set was formed out of three poses only, and had an accuracy of 99.45% with the SVM-PUK classifier. This set is useful for creating reliable pose recognition for VR applications. Furthermore, the algorithm can be used to produce other reliable sets by tweaking the parameters and inputs.

This chapter is structured as follows: In Section 6.2, the creation, evaluation, and motivation behind the benchmark pose set are discussed. This is followed by Section 6.3, where the results of the three machine learning experiments on the benchmark dataset are described, and interesting findings are discussed. Finally, in Section 6.4 the motivation, creation, and implications of the reliable pose set are discussed.

6.2 Formation of the Benchmark Pose Set

The first two objectives of the research were to select and use a taxonomy along with the poses used in LMC-based VR applications to form a benchmark set. This section justifies the choice of creating the benchmark pose set, and outlines the process of creating it.

A review of previous literature showed that very few established pose sets exist, and the ones that do exist did not take VR scenarios into account. Furthermore, most researchers in the field use their own defined pose set without providing much justification, as the research is not focused on the poses used. In other research, the pose set used is the letters of a Sign Language alphabet. Certain poses are commonly used in VR applications for the LMC, such as the *Fist*, *Point*, *Open Hand*, *Pinch*, and *Thumbs-Up* poses.

Gesture taxonomies provide an effective means to form the benchmark set. By analysing the means by which gestures are broken down into their constituent parts, one could ensure that all the relevant parts are represented in the set. Most taxonomies reviewed were too broad and did not contain sufficient detail to effectively derive a pose set. However, the taxonomy of Choi et al. [8] provided an effective means of dissecting all possible gestures into intuitive and detailed categories. This taxonomy was chosen over others largely due to the detailed categorization of the *Hand Shape* dimension of a gesture, which suited the purpose of this research. Not all categories of their taxonomy were relevant for the purposes of this research, thus only the *Hand Shape* and *Hand Orientation* categories were used. Furthermore, ambiguities in the *Cross* finger inter-relation had to be resolved, and certain values that were dependent on the left or right hand were made hand independent. Figure 3-3 depicts the adjusted notation. In their research, Choi et al. provided a method to notate hand shape and orientation. Each pose that an individual finger could make was denoted by one of six values, and seven possible values denoted the relationship between fingers. Similar notations were used for the hand orientation.

This notation method provided a straightforward means to form the benchmark pose set. Ideally, one could take every combination of values to form an exhaustive pose set. This is not a feasible solution as capturing such a large amount of data from multiple participants is too time-consuming. Instead, the benchmark set was formed analytically by ensuring that all values of the categories were represented at least once in the set, and that the common poses from LMC-based VR applications were all represented. Many of these poses were also taken from common poses used in literature, such as sign language poses. Other poses were created to represent certain values in the notation that were not yet represented. Once all values were represented, additional poses that were similar to the existing poses were included to test the separating power of the LMC and the machine learning classifiers. Collectively, these poses formed the benchmark set.

Previous literature does not describe the process taken to form their pose sets. Furthermore, no VR pose pose set exists in existing literature. The benchmark pose set

on the other hand was created for VR poses specifically, and used existing taxonomies and poses in the field.

6.3 Pose Recognition Experiments

Three pose recognition experiments, namely the orientation independent, requested orientation, and thumbs-orientation experiments were performed on the benchmark dataset. These experiments were designed to determine the effectiveness of each classifier on the set, where effectiveness was measured by classification accuracy, latency, and training times. These experiments provide insight into which classifier is best suited for hand pose recognition, how hand orientation could affect results, and which poses are problematic among others.

It could be argued that an evaluation of a depth camera should have its performance compared to the current popular mode of input for VR, namely hand-held remotes. However, it is logical to assume that these remotes will have 100% accuracy, as they mostly rely on button presses to perform actions. They also detect hand movement, and it is possible to have inaccuracies in that regard, but this research is more interested in recognizing entire poses including individual finger data, rather than just the position of the hand.

Three machine learning algorithms, namely the k-Nearest Neighbour (kNN), Artificial Neural Network (ANN), and Support Vector Machine (SVM) classifiers, were evaluated for pose classification on the benchmark pose dataset. Two variants of the SVM classifier with different kernels were used in the experiments, namely the Linear (SVM-Lin) and Pearson VII function kernels (SVM-PUK).

While determining suitable parameters for each of these classifiers, some interesting points were noted. Firstly, the ANN classifier had significantly worse accuracies than the other classifiers in these exploratory tests. Only once the raw data captured by the LMC was used instead of extracted features did its accuracy begin to match the other classifiers. When using this raw data on the other classifiers, decreases in accuracy were noted, thus any classification tests with the ANN used raw data, while the

other classifiers always used extracted feature data. Secondly, when the ANN started using raw data over feature data, which is a large increase in input data, the number of hidden-layer nodes had to be increased to accommodate this change. Thirdly, the kNN classifier had a much higher accuracy when the value of $k = 1$ was used. Any higher values caused drops in accuracy, with the largest drop being between $k = 1$ and $k = 2$ with an 8% decrease.

When it comes to comparing the results of the experiments to existing literature, some points need to be noted: First, the experiments in this research were performed with participants in a VR environment, where the LMC was mounted onto the headset, as opposed to being stationary in the other literature. This additional noise may have a negative impact on the input data sent to the Leap software. Second, the poses created by the participants in this experiment included poses where they could put their hands at arbitrary orientations, which will cause finger occlusion issues, further reducing accuracies.

6.3.1 Orientation-Independent Experiment

The orientation-independent experiment tested a classifier's ability to distinguish different hand shapes, regardless of orientation. All *Thumbs-Up Poses* were grouped into a single pose as they are all the same shape with different orientations. In this experiment, the SVM-PUK classifier had the highest accuracy of 70.3922%, with the SVM-Lin classifier achieving the lowest accuracy of 59.098%. The second highest accuracy of 66.6667% belonged to the kNN classifier, and the ANN had the third highest accuracy of 63.098%. The SVM-PUK classifier had an average latency of 31.57ms, while the second highest latency was 0.78ms by the kNN classifier. The SVM-PUK classifier is not necessarily the best classifier tested, as the choice in classifier might also depend on recognition latency. The kNN classifier provides roughly forty times faster recognition latencies at the cost of a 4% accuracy drop. The lowest latency of 0.0727ms was observed in the SVM-Lin classifier. An interesting note from this experiment is that all four classifier options had their own strengths and weaknesses,

and no classifier was objectively the best. The kNN classifier has no training period, and had the second highest accuracy and a low latency. The ANN had a slightly lower latency than the kNN, but more than a 3% decrease in accuracy and the highest training times by a significant margin. The SVM-PUK classifier had the best accuracy by 4%, but it also takes about forty times as long to classify a pose when compared to kNN. Finally, the SVM-Lin classifier had the worst accuracy, more than 11% below SVM-PUK, but was close to ten times faster than the ANN recognition latency.

6.3.2 Requested-Orientation Experiment

The requested orientation experiment used only poses that were oriented to reduce finger occlusion. Similar to the orientation-independent experiment, only hand shapes were classified and the *Thumbs-Up Poses* were grouped. A significant increase in accuracies were seen when compared to the orientation independent experiment, with the largest increase being 25% in the ANN. The average accuracy across the classifiers in this experiment is 81.2%, compared to the average accuracy of 64.8% in the orientation-independent experiment. This increase of 16.4% in average accuracy occurred with a subset of the data used in the previous experiment, where only the poses made in a particular orientation were used. This shows that certain hand orientations that reduce finger occlusion will have a positive effect on recognition rates. Thus, a reduction in finger occlusion may lead to an increase in accuracy.

In terms of latencies, similar trends to the previous experiment are seen in this experiment: The SVM-Lin classifier had the lowest latency by a significant margin, while the SVM-PUK classifier had the highest. In the previous experiment, the kNN latency was higher than that of the ANN, while in this one, the kNN is in fact lower. This is likely due to the smaller dataset size. The latency of an ANN does not scale with the size of the dataset, and thus remained constant across the two experiments. The ANN training time did decrease with the decrease in dataset size however.

In all the classifiers, but the ANN, a consistent inaccuracy pattern can be seen in

the *Fist Poses* in the confusion matrices. All three of these classifiers used the same extracted features, as opposed to the raw data of the ANN. Thus, a preliminary exploration into modifying this set was made to determine possible causes. When removing the orientation features only, accuracies changed from 76.6%, 81.4%, and 78.6% to 88.2%, 88.3%, and 65.7% in the kNN, SVM-PUK, and SVM-Lin classifiers respectively. Large increases that put their accuracies above that of the ANN can be seen in the kNN and SVM-PUK classifiers, while the SVM-Lin classifier had its accuracy decrease. This shows how sensitive these classifiers can be with input data, while the data fed into the ANN can be left as raw data with no modification.

6.3.3 Thumbs-Orientation Experiment

In the thumbs-orientation experiment, both hand shape and orientation were classified. All classifiers were trained on all poses, and the *Thumbs-Up Poses* formed the training set. Thus, for a correct classification, both the *Thumbs-Up* hand shape and correct orientation had to be determined. All classifiers achieved accuracies of over 90%, with the kNN algorithm having the highest accuracy of 96.57%, while the ANN, SVM-PUK, and SVM-Lin classifiers had accuracies of 92.86%, 96%, and 92.86% respectively. These accuracies are much higher than the previous two experiments due to the fact that only one easily-recognizable hand shape was tested, along with its orientation. In terms of latencies, a similar trend to the previous two experiments is observed: The SVM-PUK has the highest latency of 44.8ms, while the SVM-Lin has the lowest of 0.1ms. Due to the larger dataset, the kNN latency is once again higher than that of the ANN.

In the orientation-independent experiment, the kNN classifier classified 340 out of 350 (97%) *Thumbs-Up* poses correctly, where all the different orientations of the pose were grouped under this pose. In this experiment, an accuracy of 96.57% is achieved. This implies that classifying a a *Thumbs-Up* pose's shape and orientation only results in a drop of a half a percent when compared to classifying only its shape. A similar pattern can be seen with the other three classifiers. This illustrates the strength of

the hand orientation resolving capabilities of the LMC.

6.3.4 The Classifiers

Across all three experiments, some interesting findings are noted. As mentioned earlier, there is no classifier that is objectively the most effective for pose classification, as each classifier has its own strengths and weaknesses.

The kNN classifier had an average accuracy of 79.9% across the three experiments, the second lowest out of the four. However, the classifier has no training phase, and all of its average latencies were sub-millisecond, despite having to iterate through a set of 2550 pose data vectors in the first experiment. Overall, this classifier provides accuracies that are not significantly lower than the highest average (only 2.7% less that the best) and will give very good latencies provided that the dataset is not huge. If one were to assume that the latency scales linearly with the number of entries in the dataset, then only once the dataset reaches 325000 pose entries will the latency reach 100ms, which is the maximum latency for unnoticeable delay [6].

The ANN had the second highest average accuracy of 81.4% through the three experiments, with a sub-millisecond latency that remained consistent throughout the experiments. This classifier had by far the highest training time, exceeding a hundred seconds in the experiments that included all the data of the benchmark set, namely the orientation-independent and thumbs-orientation experiments. Many interesting findings about this classifier were made in the initial experiments: Firstly, an increase in the number of hidden layers brought about a decrease in the overall recognition accuracy of the ANN. Secondly, the ANN performed significantly better when the raw captured data from the LMC was used as input over the processed extracted features. However, when this was done, the number of hidden nodes in the single layer had to be increased to accommodate the larger input vector. Overall, the ANN provided very good accuracies in the main experiments with very good latency, at a trade-off of long training times. Furthermore, the classifier works well with large amounts of raw pose data, and thus very few decisions need to be made about what data should

be fed in as input.

The SVM-PUK classifier had the highest average accuracy of all classifiers, with 82.6% across the three experiments. This comes at a cost of having the highest latency by a significant margin. However, the latency never exceeded 100ms, and the average training time of the classifier was always below three seconds. Overall, this classifier provides the best accuracies provided that the much higher latency is not an issue.

Finally, the SVM-Lin classifier had the lowest average accuracy of 76.9%, 3% lower than the kNN classifier. With this low accuracy comes the lowest latency out of all the classifiers, with latencies below 0.1ms in the first two experiments, and similar training times to the SVM-PUK classifier. If such low latencies were required with a slight accuracy trade-off, then the SVM-Lin classifier is well-suited to such a task. However, such a scenario is unlikely, and having slightly worse latency for a boost in accuracy is often times a reasonable choice.

6.3.5 The Benchmark Pose Set

After these experiments, remarks about certain poses and pose groups need to be made. In general, all poses within the same group, such as the *Fist Poses* or *Finger Crosses*, were consistently misclassified as one another. However, some exceptions exist where some poses in a group were well separated from each other. This can be primarily seen with the *Open-Palm Poses* and *Finger Touches and Loops* groups. Within the groups, most poses were well-separated from each other. Some exceptions do occur: the *Claw* and *Neutral Hand* poses, the *Open Hand* and *Spok* poses, were often confused with one another, while the *Finger Purse* pose was often misclassified too. This suggests that poses with multiple extended and well-separated fingers, that thus create a strong silhouette, can be easily differentiated by the LMC. Some separating power is lost when two poses in the group have the same fingers extended, as in the case of the *Claw* and *Neutral Hand*.

In the creation of the benchmark set, a number of *Open Palm* poses were added that appeared appropriate. However, following the above findings, additional *Open Palm*

poses could be tested for inclusion into the set in future work. For example, the *Peace* (index and middle fingers outstretched, others curled down) and *L* (index and thumb outstretched, others curled down) poses could be tested.

The *Thumbs-Up* poses had very high recognition accuracies in the first two experiments when grouped up, and even when split up in the third experiment they still had strong accuracies. This pose group has many similarities to the *Fist Poses*, and as such it is expected that the *Thumbs-Up* poses would often get confused with the *Fist Poses*. However, upon analysing the results across in the orientation-independent experiment, 67 *Fist Poses* were misclassified as a *Thumbs-Up* pose, while only 29 *Thumbs-Up* poses were misclassified as a *Fist Pose*. This suggests that the LMC software may be predicting occluded fingers to be in a particular well-known posed based on what the non-occluded fingers show. For example, the thumb may be occluded to some degree, but four curled fingers are in the view of the LMC. The software then makes the assumption that a *Thumbs-Up* pose is being made, thus providing potentially false information on the position of the thumb.

6.3.6 Comparison to Literature

Limited research exists to compare accuracy results. Most VR gestural research did not report performance measurements, and research that did, did not take the measurements in VR. Therefore, the results of these experiments will be compared to other depth-based recognition systems evaluated outside of VR, which mostly comprises Sign Language recognition research. An exception to this is our previous research [10], where an accuracy of 82.5% was achieved on a small set of four distinct poses. Significant pose recognition research using the LMC in the field of Sign Language Recognition has been performed by other researchers. Accuracies of 79.8% and 72.8% using the SVM and kNN classifiers respectively on the American Sign Language alphabet was achieved [9]. The highest accuracy achieved in this research for poses made at any orientation was 70.4% using the SVM-PUK classifier. This is lower than the accuracy in [9]. However, the ANN achieved an accuracy of 88.2% on the

requested orientation poses. This illustrates the impact poor input data could have on accuracies.

As with recognition accuracy, very little VR research exists to compare latencies. Studies have shown that having a motion-to-photon latency under 20ms will minimize unwanted effects [3], and a high recognition latency could increase this motion-to-photon latency beyond acceptable bounds. Documentation on best practices for VR recommend that the frame rate displayed to the Oculus Rift DK2 should be kept above 75 frames per second [12, 46]. However, if the recognition system were run in parallel to the rendering of the VR application, it should have no noticeable impact on the rendering frame rate. A more important latency metric would be the response time of the recognition system when a pose is made in front of the camera. Studies have shown that users won't notice a delay in user interfaces when the response time of the interface is below 100 milliseconds (ms) [6]. As such, any classifier with a recognition latency below this threshold is considered acceptable in terms of latency.

6.4 Formation of the Reliable Pose Set

The experiments on the benchmark pose set had accuracies that were well below what might be expected in the VR industry. To combat this problem, specific poses were picked out of the set to form the reliable VR pose set. This set had the goal of achieving an average of 99% accuracy to deem it useful and reliable for VR applications involving the LMC. By eliminating poses to form the reliable set, a trade-off is made where the broader coverage of poses in the benchmark set is lost in favour of very high accuracies in the reliable set.

The reliable set was formed using the confusion matrix generated by the SVM-PUK classifier in the orientation-independent experiment (Table 5.16), as well as a table of the weighted Hamming Distances between poses (Table 5.20). By following Algorithm 1, the reliable pose set was formed.

The reliable pose set contains the *Open Hand*, *OK-Pose*, and *Thumbs-Up* poses, where an accuracy of 99.45% was reached. In creating this set, a point was reached where

the only remaining poses were the key poses. Of the key poses, the *Classic Fist*, and *Point* poses were removed in order to achieve this accuracy. While the reliable set contains very few pose types, it is important to note that all these poses were made at arbitrary rotations. If the same algorithm were followed with the requested orientation poses, it is likely that more poses would have been added.

The reliable pose set provides a distinct set of poses should anyone wish to create a scenario with reliable camera-based pose recognition. Furthermore, the algorithm that was created to form the reliable set would be useful to researchers or others in industry. Parameters in the algorithm such as the key poses or target accuracy could be tweaked to form an entirely different reliable pose set.

6.5 Final Remarks

Finger occlusion has been a persistent issue throughout this research. Often, poses that have their features hidden from the camera were not detected correctly. During the data gathering process, it was often noted that the participant's virtual hands did not correctly mimic their real hands. This usually occurred whenever the pose being made included some form of finger occlusion, either due to the shape of the hand itself or the orientation of the hand. The fact that the user's virtual hands were not in the correct pose strongly indicates that the script controlling the virtual hands' poses is receiving incorrect input data. This implies that finger occlusion significantly impacts recognition performance, possibly more so than the choice in machine learning classifier.

Finger occlusion can be caused by the hand orientation, the hand shape being made, or both. Some research has attempted to solve this problem, such as [44], where two LMCs were placed at right angles to gain different perspectives on the same hand. A solution such as this may not be suitable for VR as it limits users' hand movements. They will be required to keep their hands in a stationary space for tracking, whereas a single camera can be mobile by attaching it to the front of the head-mounted display.

A solution needs to be found where a second camera could be used without restricting user movement.

The primary advantage of using cameras over hand-held controllers is the freedom of being able to make any arbitrary hand pose and having the environment react accordingly. In this research, the advantages and disadvantages of camera-based approaches were discussed, specifically with the LMC. In creating a VR application with hand pose controls, it is up to the creator to decide whether the freedom and convenience of a camera-based approach is preferable to the reliability of remote-based controls. When making such a decision, the creator could restrict the set of available hand poses of the camera-based approach to have its reliability match up to the remote-based approach.

Chapter 7

Conclusion

This work evaluated the performance of the Leap Motion Controller to capture hand pose input in a virtual reality environment. A set of poses to evaluate the camera was derived from the taxonomy of Choi et al. [8]. These poses were performed by twenty-five participants in virtual reality, and the data was classified into poses using four popular machine learning techniques. The four techniques were applied in three different experiments to compare their performances with different subsets of the data.

A contribution of this research is the exploration of depth cameras effectiveness for pose input for virtual reality. Currently, hand-held controllers are used for VR application input. The findings of this research have shown that cameras suffer from input inaccuracies too often to replace controllers. However, cameras allow for a wider array of poses, provide more freedom, and are less cumbersome than hand-held controllers. Further contributions and findings have been made in this research. First, a benchmark pose set was derived, and can be used by other researchers to compare machine learning classifiers for VR pose recognition. Secondly, of the different machine learning classifiers, the SVM-PUK classifier had the highest average accuracy and latencies, while the SVM-Lin classifier had the lowest average accuracy and latencies. Thirdly, the poor input data is generally caused by finger occlusion. By using specific hand poses with minimal occlusion and keeping the hand in an orientation that minimizes occlusion, recognition accuracy can be significantly improved. Finally, a reliable pose

set was created using a novel algorithm. This reliable pose set could be used if high accuracies are desired, and the algorithm could be used for others to create their own reliable VR pose sets.

This research only worked with hand poses and not complete dynamic gestures in virtual reality. Further experimentation could be done with full dynamic gestures to evaluate the Leap Motion Controller's performance when it comes to hand motions. A major hindrance to accuracy in this research was finger occlusion, which could be significantly reduced by using multiple cameras, as seen in [44] and [38]. However, the use of multiple cameras might inhibit the freedom of the user in virtual reality. While the first camera can be made mobile by mounting it onto the user's display, the second camera would have to be placed at a separate location and angle, which is most likely in a stationary position. This would force the user to remain stationary so as to keep his or her hands in range of the second camera. Future research could tackle this problem by designing a means of making the second camera mobile, or alternatively use an array of cameras mounted in a room to always keep the hands visible from multiple angles.

Future research could also explore the capabilities of different cameras, however accommodations may need to be made to make the camera convenient for VR use. For example, a large camera cannot be mounted onto the VR display, thus forcing the user to keep their hands in the stationary camera's field of view. Furthermore, multiple image recognition techniques could be compared to determine their strengths and weaknesses under various levels of finger occlusion.

Chapter 8

Appendix

8.1 Benchmark Pose Set

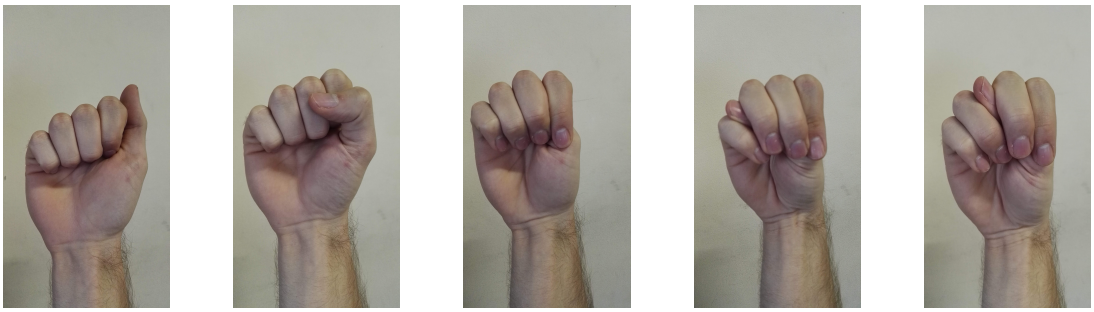
In the following tables, whenever orientation is not specified, the pose can be made at any orientation. Note that some pose types will have the *ASL-* prefix, meaning that the pose described is an alphabetical letter in American Sign Language.

8.1.1 Fist Poses

These hand poses involve the non-thumb fingers being curled closed.

Table 8.1: Fist Poses of the Benchmark Set.

Pose Name	**Notation**	**Description**
(a) ASL-A	[HS = 16666;3222-333]	Fist with upward facing thumb.
(b) Classic fist	[HS = 66666;5522-333]	Fist with thumb crossing index and middle fingers.
(c) Hidden thumb	[HS = 66666;4442-333]	Fist with thumb tucked behind the index, middle and ring fingers.
(d) ASL-M	[HS = 56666;4445-331]	Fist with thumb tucked behind the index, middle and ring fingers, but poking out between the ring and pinky fingers.
(e) ASL-N	[HS = 56666;4452-313]	Fist with thumb tucked behind the index and middle fingers, but poking out between the middle and ring fingers.



(a) (b) (c) (d) (e)

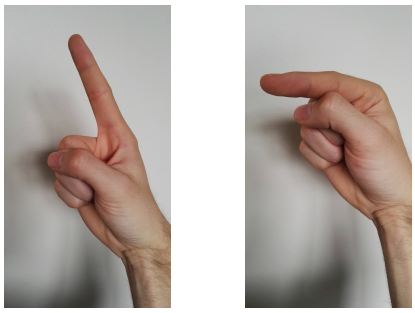
Figure 8-1: Fist Poses. Each letter corresponds to a pose in Table 8.1.

8.1.2 Index Pointing Poses

These poses involve the index finger being extended while the other finger poses are closed.

Table 8.2: Index Pointing Poses of the Benchmark Set.

Pose Name	**Notation**	**Description**
(a) Point	[HS = 61666;2522-233]	Index finger extended out, thumb crossing over middle finger.
(b) Index-Forward	[HS = 62666;4522-233]	Index finger pointing forward by resting it on the thumb, such that the thumb crosses under the index finger and over the middle finger.



(a) (b)

Figure 8-2: Index Pointing Poses. Each letter corresponds to a pose in Table 8.2.

8.1.3 Open-Palm Poses

These poses involve most of the fingers being extended.

Table 8.3: Open-Palm Poses of the Benchmark Set.

Pose Name	**Notation**	**Description**
(a) Open Hand	[HS = 31111;2222-222]	All fingers extended, no fingers bending.
(b) Neutral Hand	[HS = 44444;1222-111]	All fingers extended, but slightly bent.
(c) ASL-B	[HS = 61111;2222-333]	All non-thumb fingers extended up and grouped, thumb curled down.
(d) Flat Hand	[HS = 11111;3222-333]	All fingers extended, all fingers grouped together.
(e) Thumb-Middle Group	[HS = 11111;4322-133]	Similar to *Flat Hand*, however the thumb crosses in front of the index finger and goes between the index and middle fingers.
(f) Spok	[HS = 31111;2222-323]	All fingers extended and separated, but with grouping between the index and middle fingers, as well as the pinky and ring fingers.
(g) Claw	[HS = 55555;2222-222]	All fingers separated and bent, but not fully closed.
(h) ASL-C	[HS = 55555;2222-333]	All fingers bent, but non-thumb fingers are together to form a C-shape.

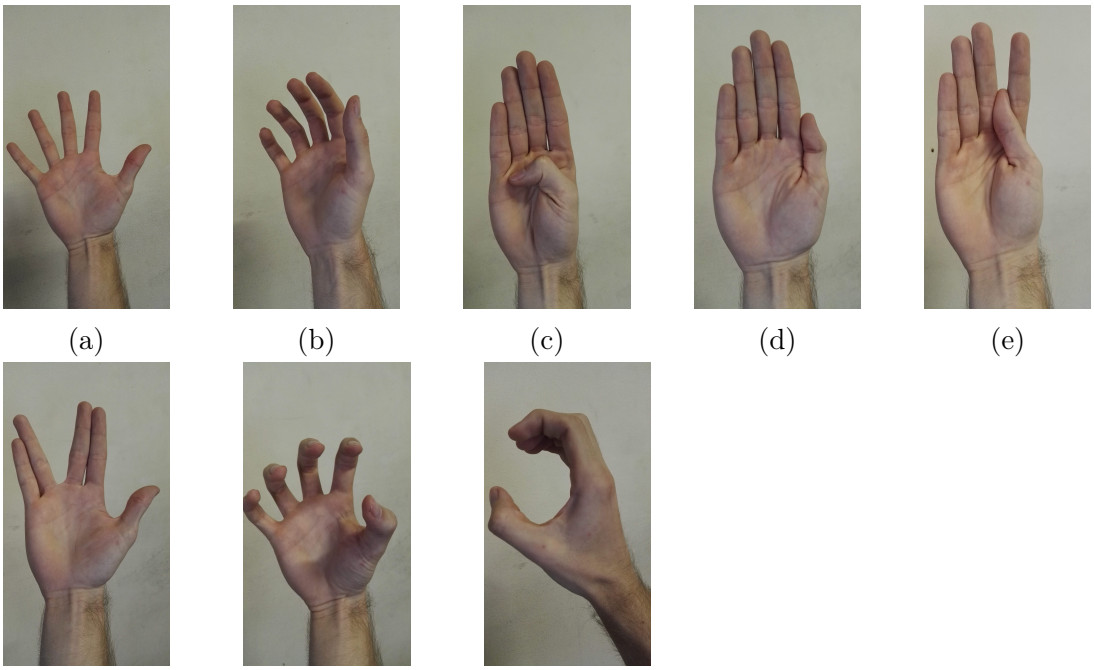


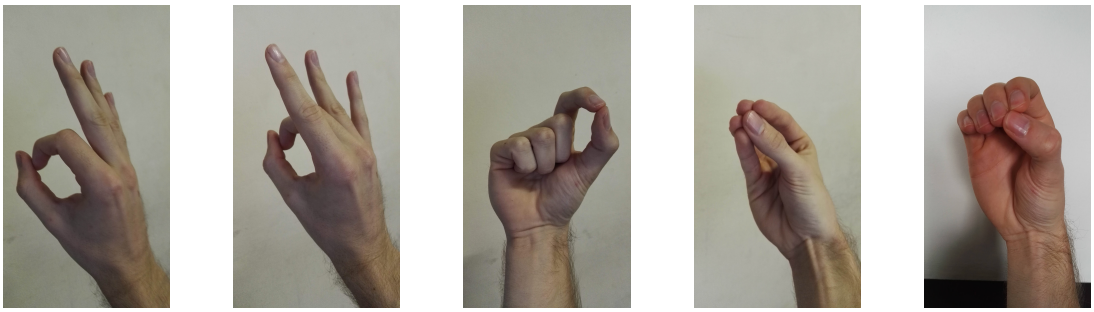
Figure 8-3: Open-Palm Poses. Each letter corresponds to a pose in Table 8.3.

8.1.4 Finger Touches and Loops

A finger touch occurs whenever two fingers touch at the fingertips, while a loop is whenever two fingers touch to form a circular shape.

Table 8.4: Finger Touches and Loops of the Benchmark Set.

Pose Name	**Notation**	**Description**
(a) OK-Pose	[HS = 55111;7222-111]	Index finger and thumb make a loop, other fingers upright.
(b) Middle OK-Pose	[HS = 51511;2722-111]	Middle finger and thumb make a loop, other fingers upright.
(c) Pinch	[HS = 22666;6222-333]	Index finger and thumb point forward and touch their fingertips. Other fingers are closed.
(d) Finger Purse	[HS = 22222;6666-333]	All fingers touch the tip of the thumb.
(e) ASL-O	[HS = 55555;7722-333]	All fingers are bent to make a loop, but the thumb only makes contact with the index and middle fingers.



(a) (b) (c) (d) (e)

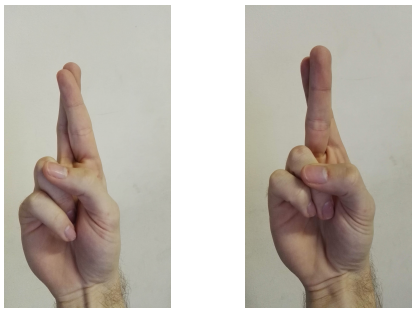
Figure 8-4: Finger Touches and Loops. Each letter corresponds to a pose in Table 8.4.

8.1.5 Finger Crosses

These poses involve one non-thumb finger crossing another non-thumb finger.

Table 8.5: Finger Crosses of the Benchmark Set.

Pose Name	**Notation**	**Description**
(a) ASL-R	[HS = 61166;2252-423]	Ring and pinky fingers closed, with the thumb holding the ring finger down. Index and middle fingers are upright, with the index finger crossing in front of the middle finger.
(b) Inverse ASL-R	[HS = 61166;2252-523]	Similar to ASL-R, except the index finger crosses behind the middle finger.



(a) (b)

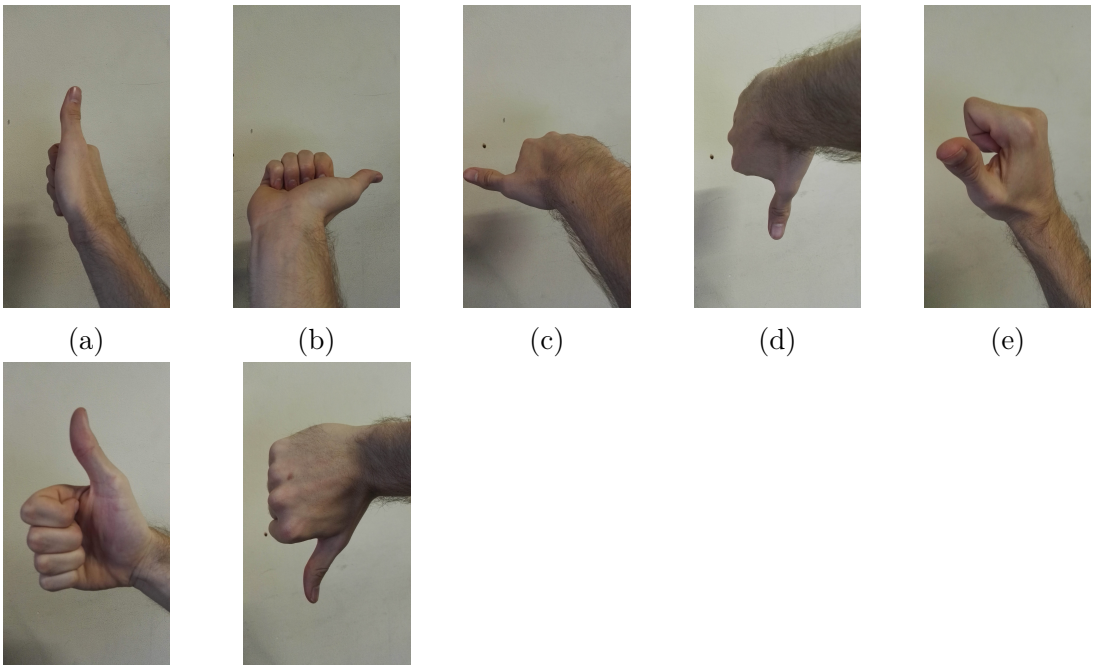
Figure 8-5: Finger Crosses. Each letter corresponds to a pose in Table 8.5.

8.1.6 Thumbs-Up Poses

These poses involve the Thumbs-up hand shape in different orientations. Each pose below has the hand shape [HS = 36666;2222-333] with varying orientations. As such, only the orientation parameter is shown in the table.

Table 8.6: Thumbs-Up Poses of the Benchmark Set.

Pose Name	**Notation**	**Description**
(a) Thumbs-up	[HO = 4;5]	Fist with the thumb pointing away from fingers, oriented in such a way that the thumb points up.
(b) Thumbs-out	[HO = 1;5]	Thumb points outwards and the palm points up.
(c) Thumbs-in	[HO = 2;5]	Thumb points inwards and the palm points down.
(d) Thumbs-down	[HO = 3;5]	Thumb points down and the palm points outward.
(e) Thumbs-back	[HO = 4;1]	Thumb points back towards the gesturer.
(f) Thumbs-up, fist-in	[HO = 6;4]	*Thumbs-up* with the fist pointing inwards instead of forwards.
(g) Thumbs-down, fist-in	[HO = 5;4]	*Thumbs-down* with the fist pointing inwards instead of forwards.



(a) (b) (c) (d) (e)

(f) (g)

Figure 8-6: Thumbs-Up Poses. Each letter corresponds to a pose in Table 8.6.

Bibliography

[1] M. Alimanova, S. Borambayeva, D. Kozhamzharova, N. Kurmangaiyeva, D. Ospanova, G. Tyulepberdinova, G. Gaziz, and A. Kassenkhan. Gamification of Hand Rehabilitation Process Using Virtual Reality Tools: Using Leap Motion for Hand Rehabilitation. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 336–339, April 2017.

[2] S. Aliyu, M. Mohandes, M. Deriche, and S. Badran. Arabic sign language recognition using the Microsoft Kinect. In *2016 13th International Multi-Conference on Systems, Signals Devices (SSD)*, pages 301–306, March 2016.

[3] C. Anthes, R. J. García-Hernández, M. Wiedemann, and D. Kranzlmüller. State of the art of virtual reality technology. In *2016 IEEE Aerospace Conference*, pages 1–19, March 2016.

[4] Nathan Beattie, Ben Horan, and Sophie McKenzie. Taking the LEAP with the Oculus HMD and CAD - Plucking at thin Air? *Procedia Technology*, 20:149–154, January 2015.

[5] J. Blaha and M. Gupta. Diplopia: A virtual reality game designed to help amblyopics. In *Virtual Reality (VR), 2014 iEEE*, pages 163–164, March 2014.

[6] Stuart K. Card, Allen Newell, and Thomas P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.

[7] Lingchen Chen, Feng Wang, Hui Deng, and Kaifan Ji. A Survey on Hand Gesture Recognition. In *2013 International Conference on Computer Sciences and Applications (CSA)*, pages 313–316, December 2013.

[8] Eunjung Choi, Heejin Kim, and Min K. Chung. A taxonomy and notation method for three-dimensional hand gestures. *International Journal of Industrial Ergonomics*, 44(1):171–188, January 2014.

[9] Ching-Hua Chuan, E. Regina, and C. Guardino. American Sign Language Recognition Using Leap Motion Sensor. In *2014 13th International Conference on Machine Learning and Applications (ICMLA)*, pages 541–544, December 2014.

[10] Andrew Clark and Deshendran Moodley. A System for a Hand Gesture-Manipulated Virtual Reality Environment. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '16, pages 10:1–10:10, New York, NY, USA, 2016. ACM.

[11] A.S. Elons, M. Ahmed, H. Shedid, and M.F. Tolba. Arabic sign language recognition using leap motion sensor. In *2014 9th International Conference on Computer Engineering Systems (ICCES)*, pages 368–373, December 2014.

[12] Epic Games. Virtual Reality Best Practices. Accessed online at `https://docs.unrealengine.com/latest/INT/Platforms/VR/ContentSetup/index.html#vrandsimulationsickness` on 2017-09-28.

[13] Julien Epps, Serge Lichman, and Mike Wu. A Study of Hand Shape Use in Tabletop Gesture Interaction. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 748–753, New York, NY, USA, 2006. ACM.

[14] Haoqi Fan and Siyuan Yao. Fingertips-based Gesture Recognition for Interaction. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, VRCAI '12, pages 347–347, New York, NY, USA, 2012. ACM.

[15] Bruno Fanini. A 3d Interface to Explore and Manipulate multi-scale Virtual Scenes using the Leap Motion Controller. In *ACHI 2014 : The Seventh International Conference on Advances in Computer-Human Interactions*, 2014.

[16] Jože Guna, Grega Jakus, Matevž Pogačnik, Sašo Tomažič, and Jaka Sodnik. An Analysis of the Precision and Reliability of the Leap Motion Sensor and Its Suitability for Static and Dynamic Tracking. *Sensors*, 14(2):3702–3720, February 2014.

[17] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning Rich Features from RGB-D Images for Object Detection and Segmentation. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 345–360. Springer International Publishing, 2014.

[18] Isabelle Guyon, Vassilis Athitsos, Pat Jangyodsuk, and Hugo Jair Escalante. The ChaLearn Gesture Dataset (CGD 2011). *Mach. Vision Appl.*, 25(8):1929–1951, November 2014.

[19] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950.

[20] Mike Harris. Virtual Strangers by mike harris. Accessed online at `https://harris.itch.io/virtual-strangers` on 2016-08-02.

[21] D. E. Holmes, D. K. Charles, P. J. Morrow, S. McClean, and S. M. McDonough. Using Fitt's Law to Model Arm Motion Tracked in 3d by a Leap Motion Controller for Virtual Reality Upper Arm Stroke Rehabilitation. In *2016 IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS)*, pages 335–336, June 2016.

[22] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-Time Plane Segmentation Using RGB-D Cameras. In Thomas Rofer, N. Michael Mayer, Jesus Savage, and Uluc Saranl, editors, *RoboCup 2011: Robot Soccer World Cup XV*, Lecture Notes in Computer Science, pages 306–317. Springer Berlin Heidelberg, 2012.

[23] Alvin Jude, G. Michael Poor, and Darren Guinness. Grasp, Grab or Pinch? Identifying User Preference for In-Air Gestural Manipulation. In *Proceedings of the 2016 Symposium on Spatial User Interaction*, SUI '16, pages 219–219, New York, NY, USA, 2016. ACM.

[24] Maria Karam and M. C. Schraefel. A Taxonomy of Gestures in Human Computer Interactions. Master's thesis, North Dakota State University, 2005.

[25] Stoyan Kerefeyn and Stoyan Maleshkov. Manipulation of virtual objects through a LeapMotion optical sensor. *ResearchGate*, 12(5):52–57, October 2015.

[26] S. Khattak, B. Cowan, I. Chepurna, and A. Hogue. A real-time reconstructed 3d environment augmented with virtual objects rendered with correct occlusion. In *2014 IEEE Games Media Entertainment (GEM)*, pages 1–8, October 2014.

[27] C. Khundam. First person movement control with palm normal and hand gesture interaction in virtual reality. In *2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 325–330, July 2015.

[28] Jong-Oh Kim, Mihye Kim, and Kwan-Hee Yoo. Real-Time Hand Gesture-Based Interaction with Objects in 3d Virtual Environments. *International Journal of Multimedia and Ubiquitous Engineering*, 8(6):339–348, November 2013.

[29] Joseph J. LaViola. 3d Gestural Interaction: The State of the Field. *ISRN Artificial Intelligence*, 2013:1–18, 2013.

[30] Joseph J. LaViola, Jr. A Discussion of Cybersickness in Virtual Environments. *SIGCHI Bull.*, 32(1):47–56, January 2000.

[31] Leap Motion. Blocks | Leap Motion Developers. Accessed online at `https://developer.leapmotion.com/gallery/blocks` on 2016-08-01.

[32] Leap Motion. How Does the Leap Motion Controller Work? Accessed Online at `http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/` on 2016-06-24.

[33] Leap Motion. Leap Motion Store. Accessed Online at `http://store-us.leapmotion.com/` on 2016-07-12.

[34] Hoo Yong Leng, Noris Mohd Norowi, and Azrul Hazri Jantan. A User-Defined Gesture Set for Music Interaction in Immersive Virtual Environment. In *Proceedings of the 3rd International Conference on Human-Computer Interaction and User Experience in Indonesia*, CHIuXiD '17, pages 44–51, New York, NY, USA, 2017. ACM.

[35] W. Lu, Z. Tong, and J. Chu. Dynamic Hand Gesture Recognition With Leap Motion Controller. *IEEE Signal Processing Letters*, 23(9):1188–1192, September 2016.

[36] Rajesh B. Mapari and Govind Kharat. American Static Signs Recognition Using Leap Motion Sensor. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, ICTCS '16, pages 67:1–67:5, New York, NY, USA, 2016. ACM.

[37] Tomas Mariancik and Karel Hulec. World of Comenius. Accessed online at `https://frooxius.itch.io/world-of-comenius` on 2016-07-27.

[38] G. Marin, F. Dominio, and P. Zanuttigh. Hand gesture recognition with leap motion and kinect devices. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 1565–1569, October 2014.

[39] A. Messaci, N. Zenati, A. Bellarbi, and M. Belhocine. 3d interaction techniques using gestures recognition in virtual environment. In *2015 4th International Conference on Electrical Engineering (ICEE)*, pages 1–5, December 2015.

[40] Microsoft. Buy Kinect for Xbox One. Accessed Online at `https://www.microsoftstore.com/store/msusa/en_US/pdp/Kinect-for-Windows-Developer-Bundle/productID.314513600` on 2016-07-12.

[41] Microsoft. Kinect API Overview. Accessed Online at `https://msdn.microsoft.com/en-za/library/dn782033.aspx` on 2016-06-28.

[42] Microsoft. Kinect for Windows Sensor Components and Specifications. Accessed Online at `https://msdn.microsoft.com/en-us/library/jj131033.aspx` on 2016-06-28.

[43] Zhenyao Mo. *Gesture interface engine*. Dissertation, University of Southern California, November 2007.

[44] M. Mohandes, S. Aliyu, and M. Deriche. Prototype Arabic Sign language recognition using multi-sensor data fusion of two leap motion controllers. In *2015 12th International Multi-Conference on Systems, Signals Devices (SSD)*, pages 1–6, March 2015.

[45] Javier Molina, José A. Pajuelo, Marcos Escudero-Viñolo, Jesús Bescós, and José M. Martínez. A natural and synthetic corpus for benchmarking of hand gesture recognition systems. *Machine Vision and Applications*, 25(4):943–954, May 2014.

[46] Oculus Rift. Introduction to Best Practices. Accessed online at `https://developer.oculus.com/design/latest/concepts/bp_intro/` on 2017-09-28.

[47] Thai Phan. Using Kinect + OpenNI to Embody an Avatar in Second Life. Accessed online at `http://projects.ict.usc.edu/mxr/play/using-kinect-openni-to-embody-an-avatar-in-second-life-gesture-emotion-transference/` on 2016-07-26.

[48] John Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. *Advances in Kernel Methods: Support Vector Learning*, pages 185–208, February 1999.

[49] Leigh Ellen Potter, Jake Araullo, and Lewis Carter. The Leap Motion Controller: A View on Sign Language. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13, pages 175–178, New York, NY, USA, 2013. ACM.

[50] Jaime Ruiz, Yang Li, and Edward Lank. User-defined Motion Gestures for Mobile Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 197–206, New York, NY, USA, 2011. ACM.

[51] K. Sabir, C. Stolte, B. Tabor, and S.I. O'Donoghue. The Molecular Control Toolkit: Controlling 3d molecular graphics via gesture and voice. In *2013 IEEE Symposium on Biological Data Visualization (BioVis)*, pages 49–56, October 2013.

[52] Martin Schubert. Geometric | Leap Motion Developers. Accessed online at `https://developer.leapmotion.com/gallery/geometric` on 2016-08-01.

[53] D. Shukla, Ö Erkent, and J. Piater. A multi-view hand gesture RGB-D dataset for human-robot interaction scenarios. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 1084–1091, August 2016.

[54] Gurminder Singh, Steven K. Feiner, and Daniel Thalmann. *Virtual Reality Software & Technology: Proceedings of the VRST '94 Conference, 23-26 August 1994, Singapore*. World Scientific, 1994. Google-Books-ID: ywTGrWPf518C.

[55] M. Sourial, A. Elnaggar, and D. Reichardt. Development of a virtual coach scenario for hand therapy using LEAP motion. In *2016 Future Technologies Conference (FTC)*, pages 1071–1078, December 2016.

[56] Jeremy Sutton. Air Painting with Corel Painter Freestyle and the Leap Motion Controller: A Revolutionary New Way to Paint! In *ACM SIGGRAPH 2013 Studio Talks*, SIGGRAPH '13, pages 21:1–21:1, New York, NY, USA, 2013. ACM.

[57] Michail Theofanidis, Saif Iftekar Sayed, Alexandros Lioulemes, and Fillia Makedon. VARM: Using Virtual Reality to Program Robotic Manipulators. In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '17, pages 215–221, New York, NY, USA, 2017. ACM.

[58] V. Tiwari, V. Anand, A. G. Keskar, and V. R. Satpute. Sign language recognition through kinect based depth images and neural network. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 194–198, August 2015.

[59] P. Trigueiros, F. Ribeiro, and L.P. Reis. A comparison of machine learning algorithms applied to hand gesture recognition. In *2012 7th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, June 2012.

[60] Fereydoon Vafaei. Taxonomy of Gestures in Human Computer Interaction. Master's thesis, North Dakota State University, August 2013.

[61] Radu-Daniel Vatavu and Ionut-Alexandru Zaiti. Leap Gestures for TV: Insights from an Elicitation Study. In *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*, TVX '14, pages 131–138, New York, NY, USA, 2014. ACM.

[62] Jun Wan, Yibing Zhao, Shuai Zhou, Isabelle Guyon, Sergio Escalera, and Stan Z. Li. ChaLearn Looking at People RGB-D Isolated and Continuous Datasets for Gesture Recognition. pages 56–64, 2016.

[63] Q. Wang, Y. Wang, F. Liu, and W. Zeng. Hand gesture recognition of Arabic numbers using leap motion via deterministic learning. In *2017 36th Chinese Control Conference (CCC)*, pages 10823–10828, July 2017.

[64] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the Accuracy and Robustness of the Leap Motion Controller. *Sensors*, 13(5):6380–6393, May 2013.

[65] J. Weissmann and R. Salomon. Gesture recognition for virtual reality applications using data gloves and neural networks. In *International Joint Conference on Neural Networks, 1999. IJCNN '99*, volume 3, pages 2043–2046 vol.3, 1999.

[66] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann, Amsterdam, fourth edition, December 2016.

[67] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined Gestures for Surface Computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1083–1092, New York, NY, USA, 2009. ACM.

[68] Deyou Xu. A Neural Network Approach for Hand Gesture Recognition in Virtual Reality Driving Training System of SPG. In *18th International Conference on Pattern Recognition, 2006. ICPR 2006*, volume 3, pages 519–522, 2006.