Depth Camera-Based Hand Gesture Recognition for Training a Robot to Perform Sign Language

Da Zhi

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

May 2018

Abstract

This thesis presents a novel depth camera-based real-time hand gesture recognition system for training a human-like robot hand to interact with humans through sign language.

We developed a modular real-time Hand Gesture Recognition (HGR) system, which uses multiclass Support Vector Machine (SVM) for training and recognition of the static hand postures and N-Dimensional Dynamic Time Warping (ND-DTW) for dynamic hand gestures recognition. A 3D hand gestures training/testing dataset was recorded using a depth camera tailored to accommodate the kinematic constructive limitations of the human-like robotic hand.

Experimental results show that the multiclass SVM method has an overall 98.34% recognition rate in the HRI (Human-Robot Interaction) mode and 99.94% recognition rate in the RRI (Robot-Robot Interaction) mode, as well as the lowest average run time compared to the k-NN (k-Nearest Neighbour) and ANBC (Adaptive Naïve Bayes Classifier) approaches. In dynamic gestures recognition, the ND-DTW classifier displays a better performance than DHMM (Discrete Hidden Markov Model) with a 97% recognition rate and significantly shorter run time.

In conclusion, the combination of multiclass SVM and ND-DTW provides an efficient solution for the real-time recognition of the hand gesture used for training a robot arm to perform sign language.

Acknowledgment

I would like to express my most profound gratitude to my supervisor, Dr. Emil M. Petriu, for his full support and encouragement, insightful advice, and expert guidance throughout my study and research. Without his incredible patience and counsel, my thesis could not be completed. It has been a great honour to work alongside him in the past two years, and I wish him a very successful academic career!

I wish to say many thanks to all the members of the Bio-Inspired Robotics Lab for their company throughout my master's years and thoughtful suggestions on my studies. My sincere thanks first go to Thiago for his invaluable assistance and guidance throughout my research works. I also thank Vinicius for his technical suggestions and the initial code for this project. I am so happy to have shared their company and experiences, all the best with the completion of their doctoral researches.

Finally, I would like to express my sincere appreciation to my beloved family, my parents, and my girlfriend, Zhijun Hou, for their endless love and understanding during the years of my master's study and the process of writing the thesis. This accomplishment would not have been possible without their support.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
API	Application Program Interface
ASL	American Sign Language
ANBC	Adaptive Naïve Bayes Classifier
CHMM	Continous Hidden Markov Model
DHMM	Discrete Hidden Markov Model
DOF	Degree of Freedom
DTW	Dynamic Time Warping
GRT	Gesture Recognition Toolkits
HCI	Human-Computer Interaction
HGR	Hand Gesture Recognition
HMM	Hidden Markov Model
HRI	Human-Robot Interaction
KNN	K-Nearest Neighbour
Leap SDK	Leap Motion Controller Software Development Kit
LMC	Leap Motion Controller
MA Filter	Moving Average Filter
NAI	Native Application Interface
ND-DTW	N-Dimensional Dynamic Time Warping
PLA	Polylactic Acid
ROS	Robot Operating System
RRI	Robot-Robot Interaction
SVM	Support Vector Machine

Chapter 1. Introduction

Nonverbal body-language communication through hand and arm gestures, and facial expressions, covers approximately two-thirds of human communicating and interacting modalities [1].

Hand gestures are one of the widely used modalities for daily interaction and communication because they produce linguistic contents like the spoken language while the other body language modalities provide more general meanings. Based on this unique feature, hand gestures are mainly applied in sign language and human-computer interaction (HCI) systems.

Sign languages using hand gestures are not only used for assisting communications between people in different circumstances, such as for “talking” in noisy environments, or when the persons are too far away, or even when the talkers have language barriers, but they also serve as a communication method for the deaf-mute conversation. According to an analysis [2] provided by World Health Organization (WHO), almost 360 million people were suffering of disability of hearing and this number has raised by around 14% in the past decade. Besides, there also are some 1.1 billion young people having a high risk of hearing disabling because of the environment they live. Because the sign languages provide such a valuable communication modality for an important segment of human population, the development of hand gestures recognition based HCIs has become a hot topic in computer vision in the past decades.

The related research can be also beneficial to other fields. Over the past decades, researchers have explored the possibilities of applying hand gesture recognition (HGR) techniques for other applications. For instance, a vision-based hand gesture recognition system was developed for touchless interaction in sterile environments, like the surgery room, so that the medical doctors can interact with the computers to analyze medical images through simple hand gestures [3]. Also, Lange et al. [4] developed an interactive game-based balance training tool to efficiently assist the post-operative patient rehabilitate from a surgery.

1.1. Motivation

Human-computer interfaces for gesture recognition have to address the challenging problem of providing for an engaging and effective interaction as natural as possible. The earliest works on hand gesture recognition used wearable devices (such as data glove, markers, and bands) as well as standard video cameras to capture the hand gestures. However, those methods are rather cumbersome and inflexible because the user must wear equipment with many cables. In addition, the standard video methods are not as reliable as the contact-based interaction methods since the extracted image information is always affected by the variations in hand size and velocity, complex background, poor lighting conditions, etc. Therefore, most of the regular camera-based hand gesture recognition approaches require special hand detection and feature extraction algorithms to improve the quality of the extracted information, resulting in a too low running time and cannot be efficiently applied to real-time applications.

In the recent years, with the development of new learning algorithms and more efficient low-cost and high-resolution 3D sensors, such as the Microsoft Kinect [5] and the Leap Motion Controller [6], the HGR approaches have been significantly improved. The new depth cameras have overcome the limitations of the old regular RGB cameras and allowed for a revolution in vision-based gesture recognition by providing high accuracy and reliable hand tracking information. This new technology allows for the skeletal data to be transferred into high-level and meaningful gesture features that will allow enhancing the success rate of the classifiers.

Nevertheless, the development of depth camera-based HGR is still a challenging task in the case of gestures made by dexterous robot hands, because of the peculiarities and limitations of the not yet enough human-like movements of the robot hands. This thesis aims to develop a more efficient human-robot and robot-robot interaction system using the Leap Motion Controller for a robust and efficient real-time recognition of the static hand postures and dynamic hand gestures made by human hand as well as by dexterous robot hands.

1.2. Thesis Contributions

This thesis proposes a novel depth camera-based hand real-time hand gesture recognition system for training a human-like robot hand to interact with humans through sign language allowing for efficient human-robot interaction (HRI) as well as robot-robot interaction (RRI). A 3D-printed dexterous human-like robot hand was developed and used to perform sign language gestures for interaction with humans or other robots. We developed a general framework that can recognize in real-time static hand postures and dynamic hand gestures based on the American Sign Language (ASL). We used a pre-recorded data set of human hand gestures to test the recognition accuracy of our system compared with other popular hand gesture recognition approaches.

The main contributions of this thesis are:

- Development of an efficient and real-time hand gesture recognition system, which uses a multiclass Support Vector Machine (SVM) method for training the recognition of the static hand postures, and a N-Dimensional Dynamic Time Warping (ND-DTW) method for the dynamic hand gestures recognition. The static hand posture recognition provides two interaction modalities: human-robot interaction and robot-robot interaction using a human-like dexterous robot hand that we specially developed.
- We recorded a 3D hand gestures training/testing dataset of static postures and dynamic gestures tailored to accommodate the kinematic constructive limitations of the human-like robot hand. The static postures consist of the ten digits (numeric symbols), the handshapes of selected letters based on ASL alphabet, and some simple postures. Meanwhile, the dynamic gestures subset includes hand rotation movements and non-rotation movements.
- We tested the recognition accuracy of the system by comparing the experimental results when using the Adaptive Naive Bayes Classifier (ANBC) and k-Nearest Neighbors (k-NN) approaches for static posture classification. Meanwhile, we used recognition result of discrete Hidden Markov Model (Discrete-HMM) classifier as the comparison to exam the dynamic hand gestures recognition accuracy as well as the average running time.

1.3. Publications Arising from this Thesis

The following publications have arisen from the work presented in this thesis:

- **D. Zhi**, T.E.A. de Oliveira, V.P. da Fonseca, E.M. Petriu, "Teaching a Robot Sign Language using Vision-Based Hand Gesture Recognition." (accepted), *2018 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, June 2018
- V.P. da Fonseca, D.J. Kucherhan, T.E.A. de Oliveira, **D. Zhi**, E.M. Petriu, "Fuzzy controlled object manipulation using a three-fingered robotic hand." *Proceedings of the 2017 Annual IEEE International Systems Conference (SysCon)*, pp. 1-6. April 2017
- **D. Zhi**, "Teaching a Robot Sign Language using Vision-Based Hand Gesture Recognition" poster in *10th Annual Engineering and Computer Science Graduate Poster Competition*, Univ. of Ottawa, Canada, March 27, 2018

1.4. Thesis Outline

The thesis is organized as follows:

- **Chapter 2** presents the background and literature reviews of the hand gesture recognition problems. The literature review focused on two vision-based hand gesture recognition aspects: static posture and dynamic gesture recognition.
- **Chapter 3** introduces the hardware and software components of the developed system and explains how these two are working together. Then we discuss the major limitations of the robot hand and of the Leap Motion Controller.
- **Chapter 4** presents the classification algorithms developed for the static hand posture recognition.
- **Chapter 5** presents the classification algorithms developed for the dynamic hand gesture recognition.
- **Chapter 6** presents conclusions and future work plans.

Chapter 2. Background and Literature Review

This chapter provides background information and literature review for the hand gesture recognition field. Section 2.1 explains some fundamental notations related to the hand gestures and sign language. Subsection 2.1.1 gives the concepts of the hand gestures, sign languages, and their relationship. Then, subsection 2.1.2 presents a comprehensive taxonomy of the hand gestures. Subsection 2.1.3 presents the definition and challenges of the hand gesture recognition for human-robot collaboration applications. Sections 2.2 and 2.3 provide a literature review of the vision-based hand gesture recognition using the conventional RGB cameras, and of the 3D depth sensor technologies, respectively.

2.1. Background

2.1.1. Hand Gestures and Sign Languages

Hand gestures are most commonly used in our real life allowing people to non-verbally communicate intentions and feelings. *"Gestures are a form of nonverbal communication in which visible bodily actions are used to communicate important messages, either in place of speech or together and in parallel with spoken words,"* [7].

Technically, a sign language is a structured subset of hand gestures used by deaf and speech-impacted people for non-verbal communication and interaction. One major difference between the sign language and the generic hand gestures is due to their relationships with the spoken languages. Many linguists [8][9][10] consider that a sign language has an equal functionality as a verbal language providing a meaningful and resourceful communication modality. Throughout the history, people, especially deaf groups, started developing and using sign languages long time ago, the earliest evidence being traced back to the 5^{th} century BC [11]. After centuries of development, sign languages have evolved, and different regional sign languages were developed, such as the American Sign Language (ASL), the Chinese Sign Language (CSL), the French Sign Language (FSL), and etc.

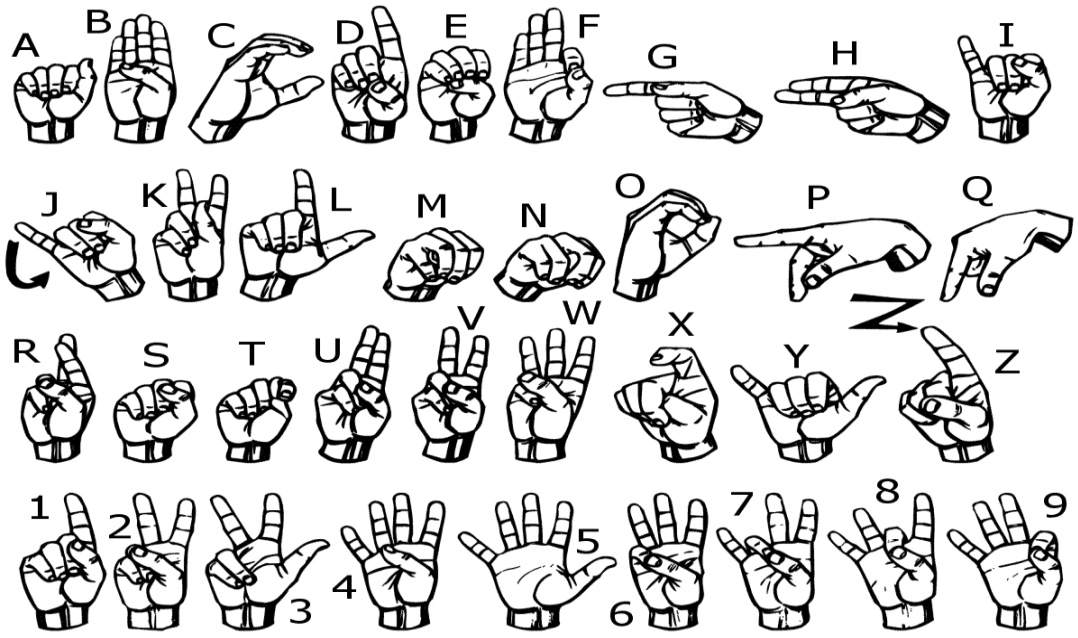


Figure 2.1 American Sign Language (ASL) alphabet and digits

On the other hand, the sign languages retain strong connections with the hand gestures, while they both involve the use of different parts of the human body to express and deliver the information. Sign languages concentrate on the upper human body [12], including hands, fingers, and arms, but the same sign or hand poses may have different meanings in different countries while the hand shapes of the sign that expresses the same sentence may differ either. Therefore, Cheok et al. [13] considered that a sign language could be treated as a category of communicative hand gestures. Thus, the next subsection will explain how to classify the hand gestures.

2.1.2. Categories of Hand Gestures

There are many ways to classify the hand gestures. Some researchers classify them based on their vocabulary meaning: communicative hand gestures and non-communicative hand gestures. The first category represents those hand gestures having a clear meaning in a sign language, like in a spoken language. The non-communicative hand gestures indicate meaningless or ambiguous actions which cannot be related to spoken vocabularies, but still provide information, so that some researchers called them informative hand gestures, as for instance finger pointing, scratching, etc.

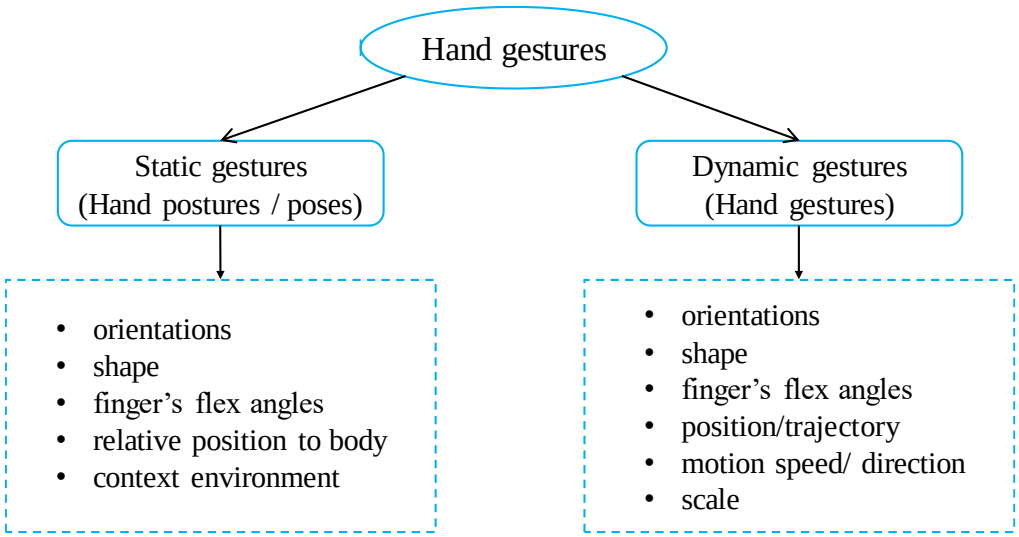


Figure 2.2 Features of the static and dynamic hand gestures

This thesis classifies the hand gestures differently based on the temporal relationships into static hand postures and dynamic hand gestures, as shown in Figure 2.2. The static hand postures, or poses, represent those gestures with stationary hand position during the performing period, characterized by the hand orientation, shape, finger angle, relative position within the body, and environment context. The dynamic hand gestures imply continuously changing of the position. They are characterized by additional dynamic parameters like the movement trajectory, motion velocity and direction, and scale.

2.1.3. Hand Gesture Recognition for Human-Robot Interaction

Hand Gesture Recognition (HGR) has always been a challenging topic in the computer vision and pattern recognition fields. After decades of research, a standard solution has been formulated for this problem, including three major steps: data acquisition, feature extraction, and classification, as shown in Figure2.3.

The data acquisition uses two main approaches: **non-vision** and **vision-based** techniques. The non-vision hand gesture recognition approaches use data gloves or other band-wearable sensing devices. These non-vision sensors provide faster response, more precise tracking information, and a larger working area. However, they are cumbersome due to the many connection cables they use and are not conveniently to put on and off the hand.

Figure 2.3 Standard Hand Gesture Recognition Diagram

The vision-based HGR approaches overcome the limitations of wearable devices by using the touch-free cameras or new depth sensors. They offer several advantages being (i) robust and easy to setup, (ii) contact-free, (iii) better suited for real-time recognition, and (iv) lower computational cost. Subsections 2.2 and 2.3 will present a comprehensive literature review of the conventional camera based and, respectively, of the novel depth sensor based HGR approaches.

Liu et al. [14] proposed a modular model of an HGR system for human-robot interaction. This model consists of five steps: (i) sensor data collection, (ii) gesture identification, (iii) gesture tracking, (iv) gesture classification, and (v) gesture mapping. The first four steps, (i) to (iv), represent hand gesture recognition, while during the additional (v) gesture mapping step, the recognized gesture label is translated into a set of robot hand control commands.

2.2. Conventional Camera-Based Hand Gestures Recognition Approaches

The conventional camera-based approaches use a single camera for hand gesture recognition. The difficulty and computational complexity needed for isolating the hand area from the image background, and the poor lighting contribute to making feature extraction the most critical process in the pattern recognition process.

Another important step to consider is the design and selection of classification algorithms for gestures made by hands with specific handshapes, orientations, etc.

This section reviews the related works from two aspects, the methods of static hand postures recognition and the approaches for recognizing dynamic hand gestures.

2.2.1. Static Hand Postures Recognition

Throughout the past decades, different approaches were proposed for dealing with hand posture recognition: (1) graph-based methods, (2) supervised learning methods, and (3) unsupervised learning methods.

Graph-based methods. While always used in the field of pattern recognition and classification, these methods were not used for hand posture recognition because of their high computational cost. Recently, the use of graph algorithms in gesture classification and pattern recognition problems has regained researcher's attention since the computational cost is no more a barrier. In 1993, Lades et al. [15] introduced the Elastic Graph Matching (EGM) method to recognize 87 persons' faces and office objects. EGM allows developing distortion-invariant Dynamic Link Architecture neural networks that do not need new neurons training in order to detect the features. One of the earlier EGM-based applications to hand gesture recognition were presented by Triesch et al. in [16][17][18]. [16] presents a robust classification system for recognizing 239 grey-scale images of 10 hand postures with complex backgrounds using EGM. It uses vertices as the nodes in graph theory to represent the image segmentation and regions and uses edges to express the relationships between the different image segmentation. This solution can reach an average classification rate of 86.2% for 2-dimensional pre-labelled hand posture images. The work presented in [17], [18] was extended [16] to the human-robot interaction, where the improved algorithms provide scale invariant for independent recognition with implicit hand segmentation. The resulting HRI system allowed for a person-independent HGR. More recently, a hierarchical EGM (HEGM) approach for HGR has been proposed in [19]. This work dramatically improved the original EGM by using a boosting algorithm to distribute the multi-levels of hierarchies to the given graph. The accuracy of the proposed algorithm has been tested using ten hand postures with three different backgrounds, and experimental results showed that the HEGM method can reach an overall recognition rate of 99.85% if the features were pre-extracted using Histogram of Oriented Gradient (HOG).

Bunch graphs approaches are a significant branch of EGM, usually called elastic bunch graph matching [20], which are used to represent the variability of the observed object shape. The bunch of the attribute of the homologous points in the images (a set of objects or the same object) possesses an inherent variability that is captured by each pre-

labeled node. A matching process compares the attributes' value of the bunch to the original image data in the graph in order to find the maximum likelihood as the similarity of the bunch graph. This approach has also been used in [16] and [18] to recognize the hand postures within complex backgrounds.

Supervised Learning algorithms. Since the rapid development of the high-performance computing, new machine learning algorithms spawned and changed the research field. Due to their high recognition accuracy, the machine learning methods, especially the supervised learning, are well fitted for solving gesture recognition problems. Different from conventional graph-based approaches, the supervised learning algorithms have an extra training phase, where a set of pre-labelled training data are fed into the learning algorithm to train the learning model, called classifier.

Zhao et al. [21] proposed an Extended Variable-Valued Logic (EVL) approach to hand posture recognition, which is known as recursive induction learning or Rule Induction by Extended Variable-valued Logic (RIEVL) [21]. This method classifies the hand poses by determining whether this pose satisfies specific rules. To achieve this goal, RIEVL provides a heuristic scheme to learn classification rules from a large amount of training examples and sets of extracted features. The classifier can be refined online by producing more rules while more training features and samples are fed into the model during the training period. There were 30 classes of hand postures used for testing the accuracies of the RIEVL method against other inductive based algorithms; the experimental results showing a recognition rate of 94.4%.

As time passed by, the conventional offline hand posture recognition can no longer meet the growing demands of the real-time applications due to high computational costs and comparatively low recognition rates.

The SVMs (Support Vector Machines) algorithm proposed in [22] has quickly gained the attention of the researchers working on the development of real-time posture recognition applications. Ulrich [23] extended the original SVM binary classifier into the multi-class using a one-against-one strategy, which makes the multi-class postures recognition possible. Chen et al. [24] presented a robust multiple-angle HGR system for static hand postures based on fusing SVM classifiers. This system uses three webcams to record each hand pose from the front, left, and right positions, and then train three

corresponding SVM classifiers. The training data were preprocessed to reduce the impact of poor lighting conditions. The recognition is done by fusing the results from the three classifiers. Experiments have showen that the fusing SVM can efficiently predict three different hand postures from 540 images with a 93% recognition rate.

Huang et al. [25] proposed a novel real-time hand posture recognition method which combines SVM with Gabor Filters. Low-dimension features were calculated by convolution operation using Gabor Filters, further refined using principal components analysis (PCA), and then used to train the SVM classifier. Experimental results obtained testing large-angle hand postures demostrated the robustness of the trained SVM classifier. Two years later, Huang et al. proposed an improvement [26], which increased the average recognition rate to 96.1%. An adaptive skin-color independent algorithm was developed to lower the influence of the poor lighting conditions and complex illuminations. Dardas et al. [27] presented another real-time postures recognition system based on multiclass SVM combining with BOF (Bag-of-Features). This work improved the skin-color invariant method by adding an extra handshape contour comparison process that made the hand area detection more accurate. After the features have been extracted from the raw images, a K-means quantization algorithm was implemented to map these critical values into a normalized dimensional vector as the training input for the multiclass SVM classifier.

An Supervised Locally Linear Embedding (SLLE) algorithm for sign language recognition described in [28] was used for the real-time Chinese Sign Language (CSL) recognition using a low-cost USB webcam. A fast detection algorithm used for a real-time hand region recovery, which may lower the detection accuracy. The feature extraction is then achieved using SLLE. There were 30 hand postures from the CSL alphabet that were used to evaluate the performance of the system, and the experimental results showed an average 90% accuracy.

Unsupervised Learning is another type of machine learning algorithm which uses unlabelled training samples in the classifier training phase. Ge et al. [29] proposed an HGR and hand tracking system for both static postures and dynamic gestures based on distributed locally linear embedding (DLLE) algorithm. The DLLE used for hand posture recognition recovers intrinsic attributes of the high-dimensional data and projects them into a lower dimensional space. Then, a Probabilistic Neural Network (PNN) is used to classify the hand

postures using their distances in the lower dimensional space.

2.2.2. Dynamic Hand Gestures Recognition

The dynamic HGR has to address the challenges of continuously tracking the hand region from each frame of a time series of images. In the feature extraction period, the useful information not only contains the static features recovered from each frame, but more importantly, the temporal coherence between the frames needs to be measured. From the point of view of the classification techniques that are used, there are four major approaches to the dynamic HGR: (i) Hidden Markov Model (HMM) and other statistical methods, (ii) Artificial Neural Networks (ANN), (iii) Dynamic Time Warping (DTW), and (iv) learning-based algorithms and other methods.

Hidden Markov Model (HMM), is one of the most commonly used statistical modelling techniques for solving dynamic hand gesture recognition presented in Baum et al. [30]. Rabiner [31] reviewed the basic HMM theory and identified three basic problems that can be addressed when using HMM in gesture recognition applications. According to him, a gesture class can be represented by an HMM template and a typical dynamic hand gesture recognition problem can be solved by finding the optimal parameters for all the HMMs via the given observed sequences. Then, an unknown gesture sequence can be recognized by computing its similarities (maximum likelihood) with each available gesture template.

Chen et al. [32] introduced a standard dynamic HGR architecture consisting of four steps: real-time hand tracking and segmenting, feature extraction, classifier training, and gestures recognition. In the feature extraction phase, a Fourier descriptor (FD) was used to estimate the spatial features, and motion analysis was used to extraction the temporal features. Then, a combined spatial-temporal feature was fed into the HMM to train the gesture classifier. The HMM classifier computes the likelihoods of each gesture template and identifies the model with the highest probability as the prediction output. The experiment used 20 dynamic gestures to test the system and results showed a 90% recognition rate. Lee et al. [33] used a threshold model based on HMM to filter out the low likelihoods. The extracted feature from the given sequence consists of the hand movement directions which used to characterize each gesture sequence as a movement trajectory. The resulting system

allowed to effectively avoid the misclassification of the gestures that did not belong to the training dataset.

Combined weighted features of hand angle, location and velocity were used to implement the HMM for HGR [34]. The hand region is detected using a skin-color invariant method and movement tracking is achieved by simply connecting the centroids of hand area in the sequence. This approach ranks the hand angular feature as the most important one, followed by location and velocity features. The experiment uses 2400 gestures to train the HMM classifier and uses another 2400 test samples to the test the recognition accuracy of the system. Another similar HMM-based system for the recognition of the gestures for the ten Arabic digits is presented in [35], using the Left-Right Banded (LRB) and Baum-Welch algorithms in the training process in order to improve the success rate of the classifier.

The basic HMM using homogeneous Markov chains to represent the dynamic sequences can only determine the time non-relevant probabilities. Marcel et al. [36] proposed an alternative training algorithm, called Input-Output Hidden Markov Models (IOHMM), which is based on a non-homogeneous Markov Chain. This method trains the HMM learns to map input and output sequences using supervised learning. However, it can only recognize two classes of gestures. Just et al. [37] introduced a gesture recognition system based on IOHMM. The experiments using a large dataset were not limited to a two-class recognition, concluded that IOHMMs actually have a lower success rate than the HMMs.

An HGR system integrating HMM temporal features and static hand shape recognition is presented in [38]. This work considered the hand movement as a sequence of independent hand poses and deployed a Kalman filter to track the hand contours as temporal features. The static shapes are recognized by a skin-color independent method, and gestures descriptors are used for training the HMM classifier. This real-time system shows robustness in recognizing gestures within complex backgrounds as well as reliability while detecting the hand movement trajectories. Different from other gesture segmentation approaches based on backward spotting, which first finds the end point and then traces it back to the starting point, Kim et al. [39] introduced a Forward Spotting Scheme to segment the gestures and recognize them at the same time. This method allows to efficiently compensate the backward algorithm's deficiencies due to the time-delay between the segmentation

and recognition, and considerably improves the system running time.

Other HMM-based HGR methods include Parametric HMMs (PHMMs), Parallel HMMs (PaHMMs), and Coupled HMMs (CHMMs). Wilson et al. [40] introduced an extended HMM method, the PHMMs, for gesture recognition that uses a global parameter in the emission probabilities of the hidden states. Experimental results prove PHMM's superiority comparatively to the common HMM method. Brand et al. [41] used a CHMM approach for training HMMs to recognize two-handed gestures. This approach overcomes the limitations of the single-process HMM (which only supports a relatively small number of states). Vogler et al. [42] developed an ASL-based hand signs recognition system using PaHMMs, which is an improvement of CHMM. This approach models the parallel processes so that all HMM models can be trained independently. The system can classify a vocabulary of 22 ASL signs, using 400 training samples and 99 test instances.

Other **statistical methods** can also support powerful HGR applications. Chen et al. [43] proposed a two-level approach to represent the recognition problem based on the hierarchical attributes of hand gestures. The lower level provides a fast hand detection using a statistical method which combined Haar-like features and AdaBoost algorithm, while the higher-level provides syntactic analysis based on a Stochastic Context-Free Grammar (SCFG). Davis et al. [44] presented a seven-gestures recognition system based on Finite State Machine (FSM) to model the four fixed-ordered distinct phases of a given gesture. The hand regions are tracked in a continuous image sequence by finding the start and stop points. The gestures are represented by extracted temporal features stored in an FSM. Suk et al. [45] proposed a Dynamic Bayesian Network (DBN) system for the recognition of isolated or continuous hand gestures captured from a video sequence. The highlight of the proposed algorithm is in the feature extraction step, where they utilize a distance based real-time decoding to classify one or two-hand gestures. Experimental results show a success rate of 99.59% for ten isolated gestures recognition, and 80.77% for the continuous stream of gestures.

Artificial Neural Networks (ANN) are frequently used for hand gestures recognition applications. Yang et al. [46] proposed a Time-Delay Neural Network (TDNN) method to classify 40 dynamic ASL gestures. In this work, the 2D hand movement trajectories of every single image are extracted using a novel algorithm which first generates hand regions

by performing a multiscale segmentation, and then matches the areas of the adjacent frames in order to get a two-view correspondence. The resulting movement trajectories are then used to train the TDNN model. The decision making is a dynamic process where the network utilizes input data to make a bunch of local decisions which are then integrated to allow for the final decision.

Chan et al. [47] developed a real-time vision-based HGR method using a combination of continuous HMMs and Recurrent Neural Networks (RNNs) to classify five simple gestures, providing better recognition performance then each of these two methods alone. Fourier descriptors utilized to describe the hand shape, were fed to Radial Based Function (RBF) in order to classify the poses. The computed similarities provided by RBF and the movement information were then used to recognize the gestures. Ahsan et al. [48] proposed an ANN-based HGR system using Electromyography (EMG) signal analysis. The hand movement features were extracted from EMG signals, and then an ANN was applied to classify the gestures.

Dynamic Time Warping (DTW) is an efficient and widely used algorithm for dealing with dynamic gesture recognition problems. Tohyama et al. [49] proposed a new feature extraction method based on vector-field assumptions that the hand regions are treated as optic flow. It uses multi-layers, such as retina-V1, MT, and MST, to extract two different kinds of information, the absolute velocities, and the relative velocities. Kuremoto et al. [50] utilized the new feature extraction method presented in [49] and proposed a novel HGR system based on One-Pass Dynamic Programming (One-Pass DP). The hand regions are detected by a skin-color independent method, and movement estimation is achieved by retina-V1 model. This work provides a new direction of estimating the hand gestures which considered the hand movements combined with a set of 40 templates of single moves.

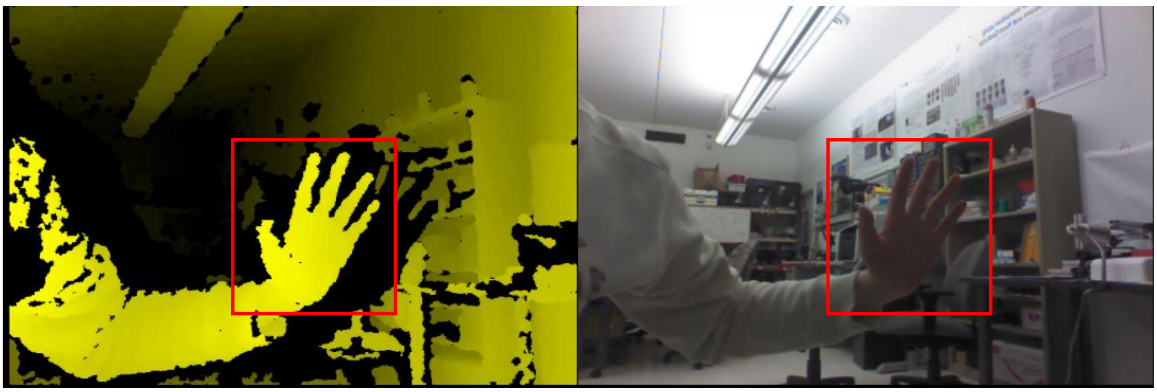
One of the earliest studies of developing hand gestures recognition system based on standard DTW algorithm is proposed by Corradini [51]. In this work, the unknown given video sequence is matched with a set of pre-defined templates and DTW algorithm has been used to find the alignment between the two sequences to be matched. Lichtenauer et al. [52] proposed a Statistical DTW (SDTW) model only for time warping period and utilized two combined classifiers, Combined Discriminative Feature Detectors (CDFSs) and

Quadratic Classification on DF Fisher Mapping (Q-DFFM) were used to perform the gesture classification. This approach separated the warping and classification processes to avoid the confliction the likelihood outputs of DTW and HMM. And the experimental result proves the effectiveness and potentials of recognizing more detailed hand gestures or poses.

Learning-based approaches mainly contain example-based methods, eigenspace-based algorithms, and curve fitting methods. Shen et al. [53] proposed a learning-based approach for gesture recognition using an example-based method. To reduce the impact of the running time caused by a large dataset, this work used a novel hand movement representation based on motion divergence fields which used to normalize the dataset into gray-scale images. The motion divergence maps are used to detect the salient regions from the local movement of each hand region obtained by a local descriptor. A gesture dataset of 1050 video sequences was used to evaluate the recognition performance. Patwardhan et al. [54] presented an eigenspace-based approach to model hand gestures with changing shapes and trajectories. Shin et al. [55] proposed a visualization navigation system based on HGR using Bezier curves to analyze motion trajectory and classify gestures. This work incorporated the hand velocity information in order to obtain reliable and accurate recognition when the speed is changing.

2.3. Depth Sensor Based Hand Gestures Recognition

In recent years, new low-cost depth cameras have created a revolution in gesture recognition field by providing high-quality depth images and providing efficient solutions solving the complex backgrounds and illuminations problems. New commercial grade depth cameras such as the Microsoft Kinect [5] and the Leap Motion Controller (LMC) [6] are increasingly used in the HGR research area since they have unique advantages comparatively to the RGB cameras when recognizing body motions.



Depth image RGB image

Figure 2.4 Hand detection using depth camera and RGB camera

A performance comparison of hand tracking using a depth camera and a RGB camera is shown in figure 2.4. It is difficult to directly detect the hand area and contour using a conventional RGB camera due to the complex background and poor lighting, and special algorithms must be used to separate the hand region from the background. However, the depth sensor camera is not affected by these problems, and it dramatically speeds up the recognition process, which is essential for the real-time recognition applications.

Our review of the depth sensor-based hand gesture recognition literature is considering separately the two HGR functions: the static posture recognition and, respectively, the dynamic gestures recognition.

2.3.1. Recognition of Static Postures

The most of commonly used depth sensors for HGR are Kinect and the LMC. **Kinect** [5] is a widely used sensor in many recent depth-camera based HGR systems.

Li [56] made an early study of real-time hand gesture and sign language recognition using Kinect to classify nine self-defined postures. The system used the OpenNI [57] framework and the Kinect SDK, and it achieved a 84% success rate for single-handed posture recognition and 90% for double-handed postures. Yao et al. [58] proposed a new hand movement detection method using a 14-patch hand partition scheme, to get a real posture training dataset with unconstrained conditions using RGB and depth camera. The hand contour model is used to lower the computational cost in gesture matching process. The

experiments show that this real-time system can reach a total 87 milliseconds for recognizing one posture.

Zhou et al. [59] describe a robust hand posture recognition system addressing the impacts of hand location variations, scale, and orientation differences. This work introduces a new distance-based metric, the Finger-Earth Mover's Distance (FEMD), to measure the discrepancies between hand regions, which matches only the finger movements. Another work based on FEMD is described in [60], which combines depth information and color-marker techniques for hand detection. Experiments proved the robustness and real-time performance of both approaches. Zhou et al. [61] developed a real-time Sudoku game based on FEMD-related hand posture recognition algorithm.

The novel Randomized Decision Forests (RDF) based algorithm for hand gesture classification using Kinect proposed in [62], which labels each depth image pixel and classifies the images regions by voting. This system has been tested using an ASL gesture set consisting of 65,000 images, and the result shows a recognition rate of 97.8%. In addition, the author also used a subset of ChaLearn Gesture Dataset [63] to test the system for classifying multiple gestures (attaining a 87.88% precision rate) and single gestures (attaining a 94.35% precision rate).

Kirac et al. [64] proposed an improved feature extraction method for hand skeleton recovery from the depth images using real-time random regression forests. The algorithm overcomes the limitations due to the finger self-occlusion and low resolution of the depth camera, and can estimate the hand joints position even if it is out of the frame. Experiments show that it performs better than the random classification forests algorithm. Aiming to enhance the feature extraction, [65] introduces an Oriented Radial Distribution feature for locally detecting fingertips and globally representing the hand postures. The performance was evaluated using several datasets of ASL gestures.

The Leap Motion Controller (LMC) proved to offer accurate hand tracking and efficient solutions for real-time gesture recognition problems.

Chuan et al. [66] developed an ASL recognition system using the Leap Motion Controller. The system deployed k-Nearest Neighbour (k-NN) and SVM classifiers to recognize 24 handshapes of the ASL alphabet, and experimental results showed a success rate of 72.78% for the k-NN and 79.83% for the SVM classifier, respectively. Mohandes et al.

[67] utilized LMC to develop an Arabic Sign Language recognition system, which included a hand tracking pre-processing phase. This system uses a Multilayer Perceptron (MLP) algorithm to train the NN classifier. Experiments showed a 99% recognition rate. Funasaka et al. [68] proposed a Leap Motion-based finger-spelling language recognition system using 16 basic decisions. The recognition accuracy for the letters varies for different order of decisions. Marin et al. [69] presented a novel HGR method using a combination of Kinect and Leap Motion devices. Features consisting of fingertips position and orientation are extracted from both Kinect and Leap Motion data, and then fed into a multi-class SVM classifier. Experiments using a subset of American Manual Alphabet convincingly validated the real-time performance of the described system.

Van den Bergh et al. [70] proposed a real-time HCI system using a Time of Flight (ToF) camera. The ToF also helped to enhance the Haarlet-based gesture recognition performance. The system can recognize six basic gestures used to interact with a robot using the Robot Operating System (ROS) [71]. Experiments showed that the combination of the depth camera and ToF cameras could reach a success rate of 99.54%. However, this work also points that the accuracy of using the two cameras together does not lead to any significant improvement as the depth cameras are good enough to provide all the information that the gesture recognition system actually needs.

2.3.2. Recognition of Dynamic Gestures

The dynamic HGR remains a more challenging research topic than the static posture recognition due to the scene complexity, feature extraction difficulties, and flexibility of human-made gestures. This subsection reviews recent works on dynamic gesture recognition using depth cameras, Kinect, the Leap Motion Controller, and other depth sensors.

Kinect-based systems: Gallo et al. [3] proposed a contact-free interaction system using Kinect to manipulate the medical images data, like Magnetic Resonance Imaging (MRI). Hand and arm gestures, such as scaling, animation, rotation, translation, are defined and used for topological analysis. Lai et al. [72] developed a Kinect-based near distance hand gestures recognition system using two different nearest-neighbour algorithms. One is using the classic Euclidean distance metric while the other is using the log-Euclidean metric. Experiments conducted while recognizing eight hand gestures from a dataset of 20

classes in real-time showed a 99% success rate.

Lui [73] proposed a product manifold model to normalize the gesture data tensors based on a non-linear regression model. This regression framework is formulated as a composite function of location attributes. Experiments were conducted using three hand gesture datasets: ChaLearn, Cambridge hand gesture [74], and UMD Keck body-gesture [75], and showed a good performance. Lui [76] extended the least squares regression based algorithm to develop a gesture recognition system that provides a better solution for ChaLearn one-shot-learning challenge using two key features: least squares fitting and underlying geometry. Another one-shot-learning was presented in [77], which introduced a template matching scheme to represent the hand gestures. The gesture template allows the algorithm to classify gestures by matching the template with the best likelihood. Since this method used only training sample per action, the hand region segmentation and movement measurement are no longer needed. The classifier used a Fourier transform to calculate the correlation coefficient of gesture templates. Experiments used comprehensive dataset to evaluate the proposed method.

Wan et al. [78] proposed a one-shot-learning based recognition algorithm combined with Bag of Features (BoF) strategy to classify ChaLearn dynamic hand gestures using an RGB-D camera. A novel features extraction method, namely 3D Enhanced Motion Scale-Invariant Feature Transform (3D EMoSIFT), was used to fuse the depth data. The new features are less affected by the scale and orientation while containing more useful visual information compared to the traditional BoF features. A sparse representation of the input data (SOMP) is used to represent features using a linear combination of codewords. This approach showed a good performance in ChaLearn gesture competition [63].

Similar to the regular RGB-based HGR, the DTW algorithm still plays a key role in depth-based gesture recognition problems. Pisharady et al. [79] proposed a robust hand tracking and recognition algorithm based on DTW and probabilities estimation. The features are extracted by Kinect, and the size, velocity, and detection of the hand gestures are enhanced using DTW. A multiple probability estimation algorithm was used to classify 12 letter spelling gestures and the experimental results showed a 96.85% success rate.

Cheng et al. [80] presented a 3D HGR system based on an Image-To-Class DTW (I2C-DTW) method. The basic idea is to disassemble a 3D gesture sequence into a set of

finger combinations, namely “fingerlets”, which represents the different gestures and captures the discrepancies between gesture classes. The gesture classification is done using multiple I2C-DTW. The method showed significantly improved performance when tested using two ASL gesture datasets. Later, Cheng et al. [81] extended their work and proposed a Windowed DTW (WDTW) algorithm for multi-class 3D hand movement trajectory recognition. They proposed a novel concept, called parameterized searching window, to calculate the cost matrix to find the begin and end points of a gesture sequence. A hand gesture dataset of 16 classes recorded using Kinect was used to evaluate the performance of the proposed approach.

The LMC-based systems: Chen et al. [82] proposed a rapid HGR system using the LMC. A gesture database consisting of 3D movement trajectory of 36 gestures recorded by Leap Motion was used to test the accuracy of this system. The gesture classification is done using SVM classifier. Experiments comparing SVM’s and HMM’s performance proved that SVM has a better performance regarding the overall recognition rate and running time. Lu et al. [83] developed another HGR system based on Leap Motion. This work introduced a simplified gesture recognition procedure without the traditional pre-processing and hand detection steps, as the LMC significantly simplifies the hand extraction and tracking by providing accurate absolute positions of hand joints. The hand gestures are classified by Hidden Conditional Neural Field (HCNF) algorithm. Experimental results show an accuracy of 89.5% when using the LMC-gesture3D database and 95.0% for Handicraft-Gesture database.

Other depth sensor-based systems: Holte et al. [84] presented a view-invariant hand gestures recognition system using a depth-intensity camera which provides both depth information and intensity images. The algorithm aligned these two layers of data to improve the recognition accuracy. Different from the conventional movement trajectory-based approach, this work uses the shape context in the image to represent the motion primitives. The gestures are classified by the probability Edit Distance approach which finds the gesture class with the best description. Test data samples recorded from a different viewpoint then the training data are used to test the system accuracy, which returns a 92.9% success rate.

Erden et al. [85] used a Pyroelectric Infrared (PIR) camera and a regular RGB camera to develop a hand gesture remote control system. The PIR camera is used to recover the hand movements in the video sequence. If the sequences are recognized as valid hand movements, the captured information is directly sent to the classification step, otherwise a video analysis has to be performed on the RGB camera data. An improved Winner-Take-All (WTA) algorithm was used to classify the gestures in real-time. Experiment shows that this method has a better performance in multi-model gesture recognition than the Jaccard distance-based recognition method.

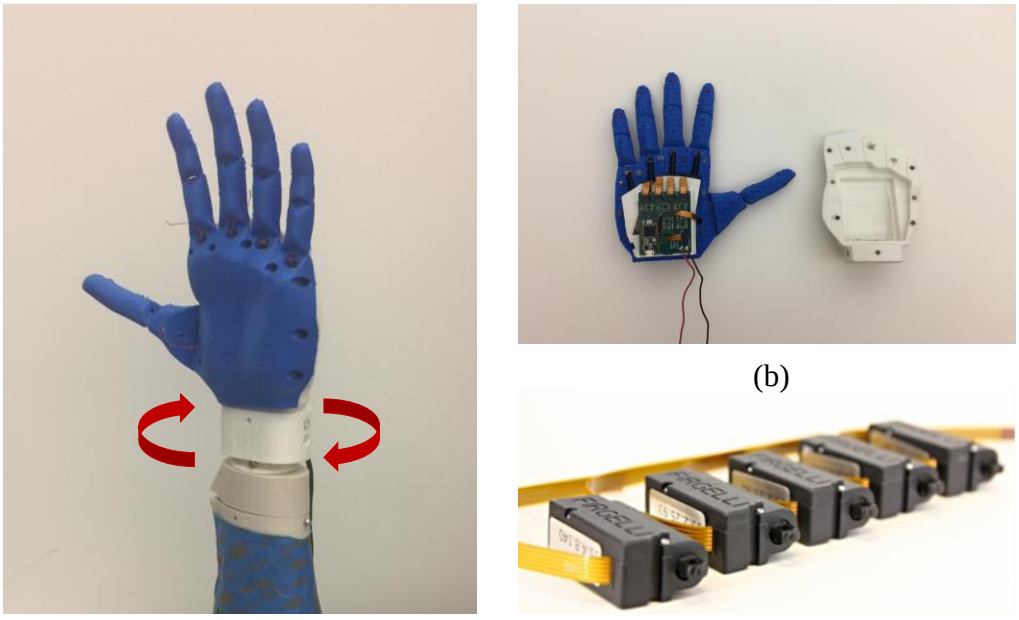
Chapter 3. System Overview

This chapter presents the hardware and software components of the developed system. The hardware architecture given in section 3.1, consists of three major ingredients, the robotic hand, the robotic arm, and the Leap Motion Controller. The software architecture, presented in section 3.2, consists of the Robot Operating System (ROS) framework, the software development kit for Leap Motion Controller, and a C++ library used to real-time hand gesture prediction. Section 3.3 will discuss the hardware limitations of the assembled robotic hand and the Leap Motion Controller. Section 3.4 presents the conclusions.

3.1. Hardware

The three hardware components are the robotic hand, the robotic arm, and the Leap Motion Controller. The five-finger robotic hand and the arm are assembled to build a human-like robot hand, as shown in Figure 3.1.

3.1.1. The Robotic Hand



(a) (b) (c)

Figure 3.1 Assembly of the robotic hand

The robotic hand, shown in Figure 3.1, is an open-source 3D printed hand [86]. It has five underactuated fingers actuated by tendons and linear motors shown in Fig. 4(c). Also, there is an extended wrist cover (white part in Figure 3.1(a)) at the base of palm which will be connected to the robotic arm to perform the wrist rotation movements. With this extended cover, our robotic hand can perform hand rotation movement around the z-axis from -45 degrees to +45 degrees.

3.1.2. The Robotic Arm

The robotic arm, as known as forearm [87], is an open-source fully 3D printed robotic arm designed for research and educational purposes, as shown in Figure 3.2. It mainly consists of arm cover, wrist cover and wrist motor, and all the parts are printed with PLA material. It should be noticed that the robotic arm part used in this thesis is not the standard fully assembled forearm with all servo motors because this work focuses on hand gestures and wrist movement. So, there are no motors inside the arm cover but an empty shell to support the robotic hand. A servo motor [88] was installed inside the wrist cover of the arm to control the hand rotation movements around the vertical axis with 90 degrees.

The wrist motor is connected to an electronic board with a microcontroller as shown in Figure 3.2(c), which powered by a regular power supply via the bottom right red and black wires. The other five wires are all connecting to the wrist motor as shown in Figure 3.2(b). The brown wire gets the feedback of the potentiometer of the wrist motor for the controller stabilization purposes. The other wires connected to the wrist motor gives the power and converts the voltage when we need to change the movement direction of the motor. Besides, there is a micro USB interface on the microcontroller which allows the PC to connect and control it. The primary duty of the microcontroller system is to control the gear rotation of the wrist servo motor when a movement command received from computer, by converting it into the direct voltage. From the software perspective, the incoming movement command is a continuously changing float value with a range from 0 to 1, where 0 means negative 45 degrees and 1 indicates positive 45 degrees.

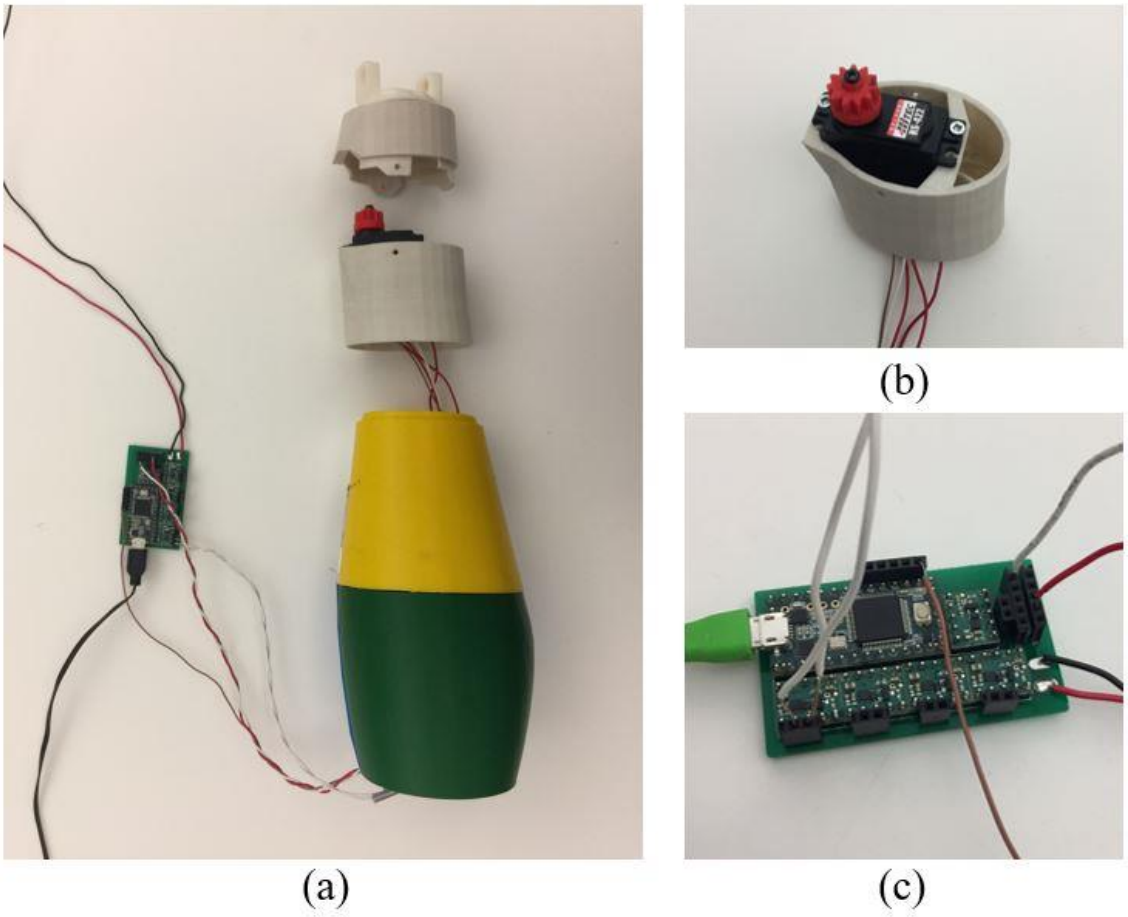


Figure 3.2 The Robotic Arm and its assembly

3.1.3. The Leap Motion Controller

The Leap Motion Controller [6] has quickly created a revolution in the field due to the new method of interacting. Unlike the popular depth camera, such as Microsoft Kinect, the Leap Motion Controller only concentrates on hand tracking and detection, providing high precision discrete motion and position information, and a well-maintained and comprehensive software development kit.

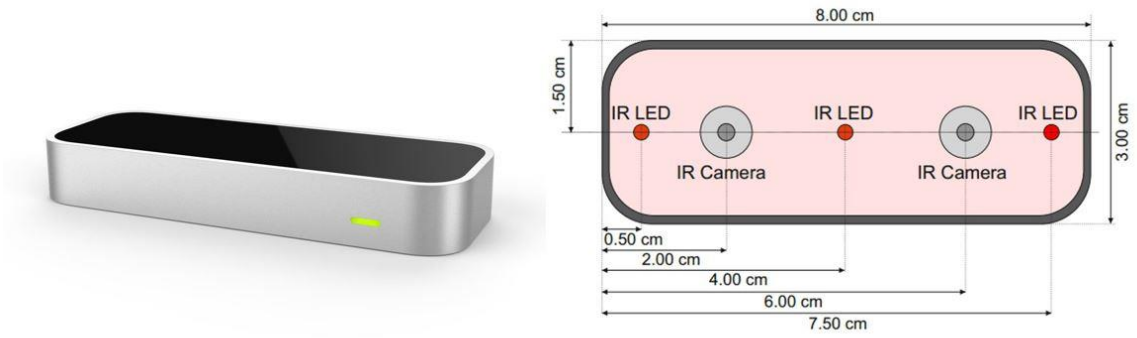


Figure 3.3 The Leap Motion Controller [89]

The Leap Motion Controller uses visual cameras and infrared light to provide comprehensive position information of the hands and finger by establishing a 3D space above the device's surface. In this coordinate system, the origin is located at the center of the surface of the Leap Motion Controller. The x- and z-axes occupy the horizontal plane paralleling to the device's surface, while the y-axis is vertical to the x-z plane, with the increasing positive values upward. Because the observed hands are always above the surface of the Leap Motion Controller, the discrete positions of hands would not contain any negative values of the y-axis. For instance, the output of palm position is (-25.812, 102.571, -63.436) means the center point of the palm is located at 102.571 millimeters above the surface of Leap Motion Controller in the current frame.

The working area of the Leap Motion Controller is very limited, only approximately from 25 millimeters to 600 millimeters with a field of view of 150 degrees above the device's surface in the air as shown in Figure 3.4. However, the localization precision of Leap Motion is higher than other depth sensors, which makes the position output of the finger joints, fingertips, and palm center can be accurate to three decimal places. The Leap Motion Controller not only provides the precise position of the hands, but it is also tracking each part of the hands and providing the speed of them in millimeters per second. The user can quickly determine which finger or joint is moving since the speed of each part is tracked separately.

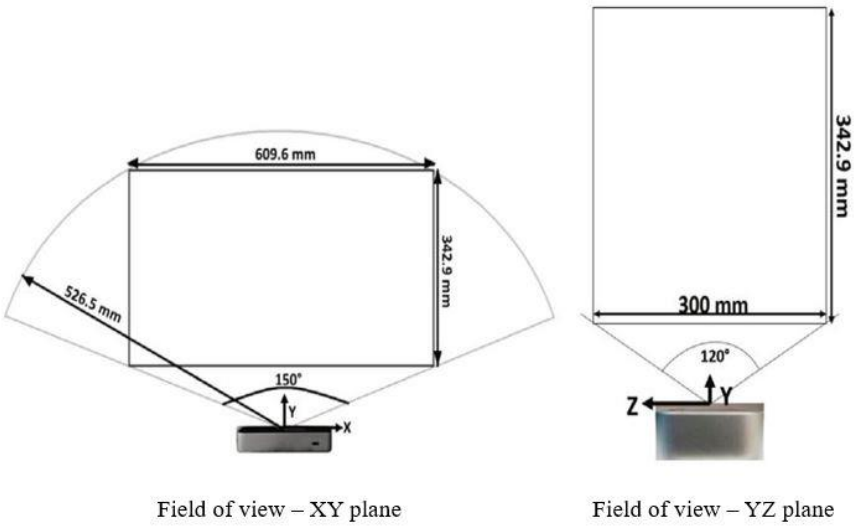


Figure 3.4 The working area of the Leap Motion Controller [90]

In addition, there are several advantages of Leap Motion Controller compare to other popular depth sensors: (1) Easy to connect to a computer via USB cable, (2) Portable. With width: 3.1-inch, Depth: 1.2-inch, Height: 0.5-inch, and Weight 1.6 ounce, (3) Affordable and inexpensive with around 100 dollars. These unique advantages make Leap Motion Controller has a perfect fit for hand gesture recognition research than other depth sensors.

3.2. Software

The diagram shown in Figure 3.5 specifies the relations between each component as well as indicates the inputs and outputs under the ROS (Robot Operating System) framework, as follow:

1. *Robot hand A, or human hand* performs the hand gestures representing the input of the interaction system. The hand gestures can be classified into static hand postures and dynamic hand gestures;
2. *The Leap Motion Controller* used to obtain the real-time hand gestures. The inputs of this sensor are the static hand postures or the dynamic hand gestures, and the outputs are the corresponding depth data containing hand tracking information;
3. *Classification Module* is the most important part of the proposed system for the training phase as for the gesture classification. It receives the raw depth data of

captured hand gestures as the input and outputs the predicted gesture label to the robotic hand B. In classification module, there is a training phase need to be implemented to perform real-time gestures classification. The more details will be explained in section 3.3.4;

4. *Robotic hand B* is the destination robot which will receive the predicted gesture labels from the classification module and transfer the predicted label into movement command send to robotic hand B via ROS system.

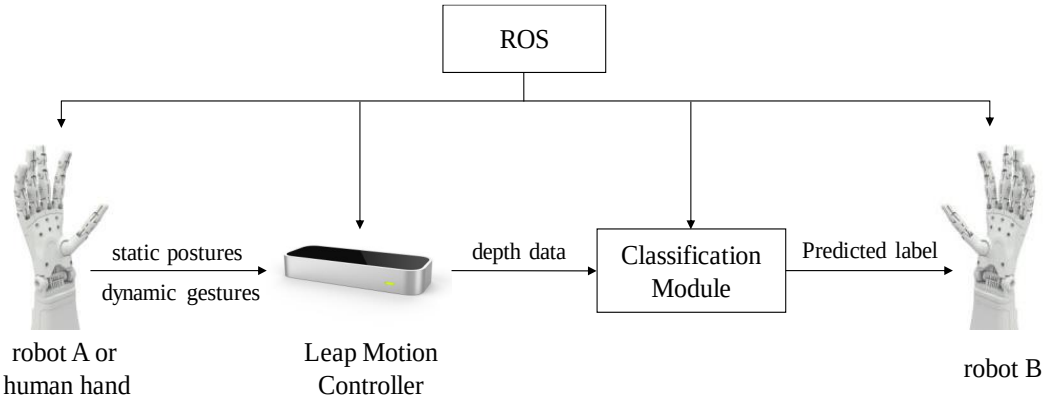


Figure 3.5 The Overview of Software Architecture

Based on the input prediction label, the ROS hand-node and wrist-node will send a size of 5 float value to the hand node and one float value to the wrist node. For example, if the prediction class is 1, then a float vector [0.0, 1.0, 0.0, 0.0, 0.0] (representing gesture 1) will be sent to five linear motors of the hand and a float value 0.5 (representing the rotation angle equals to zero) will be sent to wrist motor. All these float control values are within the range [0, 1], for the robot hand movement command, 0 means the fingers fully closed and 1 indicates the fingers fully opened. For the wrist part, value 0 represents the wrist rotates to the position of -45 degrees and 1 represents +45 degrees, respectively.

The presented interaction system was developed under ROS (Robot Operating System) framework [71] and integrated with Leap SDK [91] and GRT (Gesture Recognition Toolkits) library [92] to program. The ROS is the highest-level framework which controls and communicates all system components through the serial port using peer-to-peer technology. The GRT library is deployed in classification module and used to handle the real-

time hand gesture recognition. Moreover, the Leap Motion Controller provides it own software development kit to give users the access to the hand tracking data, which is a necessary software component in this system.

3.2.1. Robot Operating System (ROS)

The Robot Operating System (ROS) [71] is an open-source inter-robots communication software framework. It provides a well-organized communication architecture which allows a wild range of robotic hardware to communicate each other in the different layout. More importantly, ROS also has the capabilities to integrate large-scale of software, such as third-party libraries and device's SDK, since most of the current projects involve varying equipment and learning algorithms. Furthermore, it allows developers with different levels to work together under the framework using C++ or Python programming language.

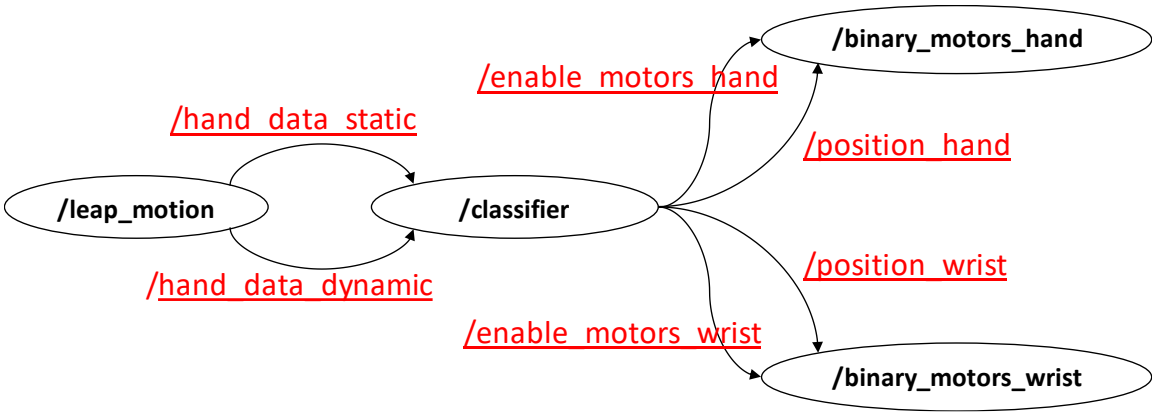


Figure 3.6 ROS framework

The ROS framework provides better performance and convenience for the robotic project development compared to other similar interfaces. To deal with any level of robot communication, ROS framework introduced some basic concepts of its implementation, and they are ROS packages, ROS nodes, ROS messages, ROS topics, and ROS services.

Packages are the primitive folders of independent projects which stored within the ROS workspace, and each project must create a unique ROS package to save everything related to it. The structure of each ROS package is restrictive, a *cmake* file must be included for telling the compiler how to build the code and listing all the dependencies, and an XML

file of describing and declaring some fundamental properties of this package. Besides, the ROS system has a pre-installed build solution instead of the default *make* or *cmake* command to compile and build the project in Linux, which is called *catkin*.

Nodes are the executable programs in a ROS package. The core of ROS nodes is the initialization of node class using the ROS library to make every executable program or components can communicate with other nodes using peer-to-peer links. After that, all the elements in ROS system are treated as ROS nodes (shown in Figure 3.6), and each node can send data to or receive data from other nodes. This peer-to-peer communication includes two parts, namely "**publishing**" and "**subscribing**".

- Publishing indicates the action of sending data from a ROS node to a ROS topic. The data is sent by a publisher in the node, which called "publisher", and a ROS node can contain many publishers. It means a ROS node can send many different types of data to the ROS topics, allowing other nodes get these data as needed.
- Subscribing denotes the action that a ROS node receives data from a ROS topic. To receive the data, a ROS node needs to create one or more "subscriber" inside the node and determine the name of the specific topic to subscribe. It should be noticed that a subscriber can only "subscribe" one topic at a time.

Figure 3.6 shows the foundational concept of ROS node in *"/leap_motion"*, *"/classifier"*, *"binary_motors_hand"*, and *"binary_motors_wrist"*. These four ROS nodes represent the corresponding components in the presented interaction system.

Messages indicate the way of how ROS nodes communicate with others. In the ROS framework, a ROS message composes of a specific type of data structure and will be published on a unique topic. This data type can be any standard data structure supported by most popular programming languages, such as integer, float, double, boolean, etc., or the arrays that consist of other messages.

Topics mean that a ROS node will send the message to a provided topic. In general, a ROS topic will be created with the initialization of a **publisher** by giving a unique name. For instance, ROS node *"/leap_motion"* is publishing the raw data of static hand posture and dynamic hand gesture from the publishers inside the node to the ROS topic *"/hand_data_static"* and *"/hand_data_dynamic"*, respectively. Simultaneously, Another

ROS node "/classifier" is listening to the two topics and getting its messages using **subscribers** in the node when there are messages in the topics.

- Publisher represents the class of letting the ROS nodes send messages to other nodes. It will notify the ROS master node to register a given topic name and keep this registration of publishing. Except for the topic name, a typical publisher will be given a size of the message queue to buffer up the published messages which have not been sent yet when the messages are published too quick.
- Subscriber denotes the class which allows the ROS nodes can receive messages from the other nodes by calling its subscribe function. This function usually has three parameters: (1) the first parameter is the name of the topic that a ROS node wants to subscribe, (2) the second parameter is the message queue same as the publisher which gives a buffer to store the received messages that have not been proceeded, (3) the third parameter is a callback function that will be triggered when a message has arrived.

A ROS topic usually contains only one type of ROS message and allows multiple subscribers to receive message from it. On the other hand, a ROS node may publish or subscribe various topics to send or receive messages. Furthermore, all the publishers and subscribers in concurrent ROS network are comparatively independent and they don't have an awareness of knowing each other.

Services indicate the alternative communication method between each ROS nodes in the concurrent framework. A ROS service can only be advertised by one ROS node with a specific name. This service usually contains many commands that can be attached and directly operate the nodes to send a request and receive a response.

In this thesis, the version of the ROS framework is **indigo** and installed on **ubuntu 14.04 LTE** Operation System.

3.2.2. Leap Motion Controller SDK

When the Leap Motion Controller device is connecting to the PC via a USB, the auxiliary software runs a background process to receive the hand tracking data from the device, which called "**Leap Service**" [91] in the system architecture of Leap Motion as shown in the Figure 3.7. This service processes the hand tracking information from the hardware and

sends it to the foreground application by default or the background application by request. Besides, there is a Leap Motion application works independently from the service process and allows the user to control the device by changing the basic settings or installation configuration of the Leap Motion Controller.

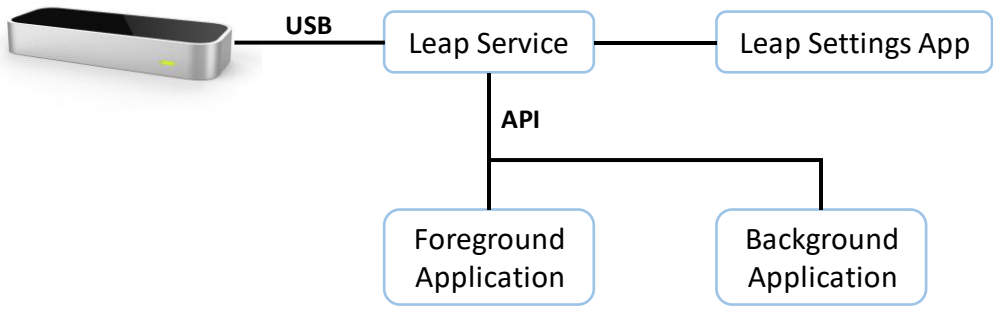


Figure 3.7 The architecture of Leap Motion software

The Leap Motion Controller software development kit, as known as *LeapSDK*, provides two different type of APIs (Application Programming Interfaces), a Native Application Interface (NAI) and a WebSocket Interface. However, the latter API would not be introduced in this section since the presented interaction system was developed using Native Application Interface. Same as all APIs, the NAI is a dynamic link library can be used in many popular programming languages, such as C++, Python, and Java. It also gives the application the access to the Leap Service to get the hand tracking data from the device so that we can develop our foreground or background application. In the system, the foreground application indicates the program continuously receiving hand movement tracking information from the device through NAI, while the background application pauses collecting data from the service process and runs in the background.

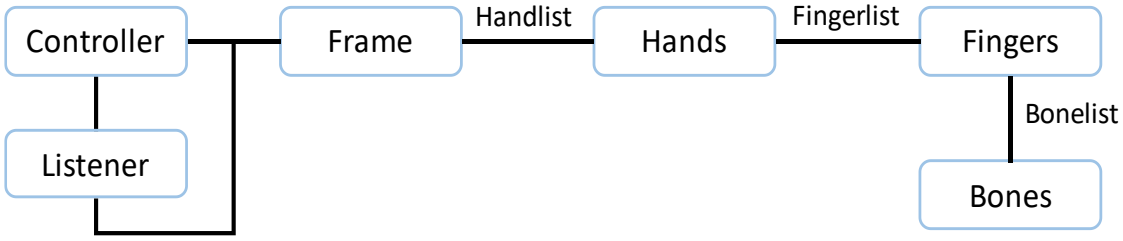


Figure 3.8 The layout of Leap Motion NAI

The NAI provides complete and absolute hand positions in 3D space created by Leap Motion as mentioned earlier in section 3.2.3. Through the NAI, all hand motion data can be obtained frame by frame, and each frame of tracking information consists of the absolute positions of each part of the hand, including fingers, bones, fingertips, etc., as shown in Figure 8. To get the desired information from the current frame, we need to trigger some callback functions in each layer. The main classes provided in NAI consists of Controller, Listener, Frame, Hand, Finger, and Bone, as follow:

- **Controller** class represents the Leap Motion Controller, and the created controller object can automatically make a connection to the service process to obtain the hand tracking data. This class contains several member functions that user can use to configure the device.
- **Listener** class includes many callback functions to be overridden by assigning a controller instance to listen to the events of the controller. For instance, the frame callback function from a listener object can be triggered to get the frame information when the latest frame obtained by the Leap Motion Controller.
- **Frame** class indicates a single frame captured by the device which contains a set of hand and finger tracking data, giving their orientations, positions, directions, and velocities. The frame object can be initialized either by implementing a listener class to get the latest frame when the callback has been triggered or creating a frame object as usual in a controller object. A frame object attached with a handlist which has all the tracking data as we needed.
- A **hand** class provides the physical motion information of detected hand. It contains a batch of hand tracking data, including velocity, position, and normal of palm, the direction of hand, and the list of attached fingers. Besides, a handlist consisting of hand objects can be extracted from the frame class, and this hand objects normally compose left hand and right hand.
- Similar to the hand class, we can get the **finger** objects from a finger list provided by hand class. In the list, all fingers are stored in the order of thumb, index, middle, ring, and pinky. Moreover, finger class also provides the position and direction of fingertips as well as the velocities, and per each finger, a bone list is contained to give the users the access to the bone object.

- The **bone** class is exactly the primitive class in the NAI architecture. It is the smallest class and only provides the tracking information of the detected bone. For example, in index finger, length: 52.9144mm, width: 17.1565 Proximal bone, start at (3.21084, 147.759, 64.3163), end with (11.4152, 156.976, 27.6814), direction: (-0.212235, -0.238427, 0.947686). Except to thumb finger, all fingers have four bones identified as Metacarpal, Proximal Phalanx, Intermediate Phalanx, and Distal Phalanx, in the order from hand base to tip, as shown in Figure 3.9.

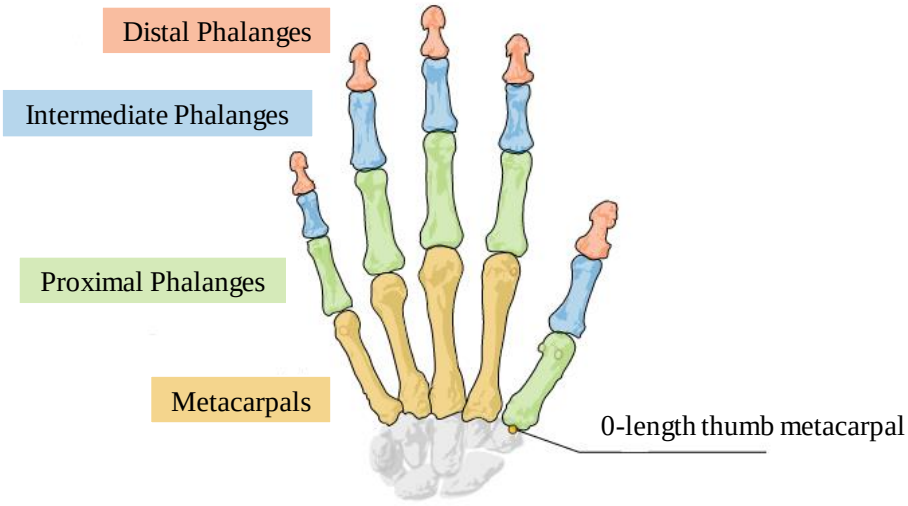


Figure 3.9 Illustration of bones class provided in Leap Motion API [93]

In addition to above classes, we can obtain the many other hand tracking information in Leap Motion API, such as the distances between fingertips or any two arbitrary joints, palm's rotation angle around each axis, and the velocities. These extra tracking data still useful in the presented interaction system will show in Chapter 4 and 5.

3.2.3. Gesture Recognition Toolkits

The Gesture Recognition Toolkits (GRT) [92] is an open-source C++ library developed for real-time gesture classification and recognition applications. It contains a set of popular and classic Machine Learning algorithms that can be used to integrate into user's C++ project. A GRT project can handle with a wild range of data types or sensor inputs, such as the voice data, motion data, and even depth data, which makes the library can be applied to various applications. Furthermore, users can change different learning algorithms deployed

in the project to get the best recognition performance according to the type and specific demand of the project. For instance, the SVMs algorithm might be used in a static postures recognition project if the posture inputs are linear separated; otherwise, a k-NN classifier would be better to handle with non-linear separable postures.

Recently, this real-time gesture recognition library has rapidly taken researcher's interests according to its several unique advantages. Firstly, GRT provides a quick and comprehensive data training phase. It allows the user to record own custom gestures through any sensor and save the recorded gestures as labelled training dataset for later use. Then, the labelled training will be sent to supervised learning method to train the gesture classifier. Secondly, GRT has additional signal processing algorithm can be used for filtering the gesture inputs. This large set of processing algorithms includes pre- and post- signal processing algorithms, and feature extraction methods, such as Moving Average Filter, Non-Movement Filter, and trajectory features, to refine the training dataset and increase the recognition accuracy of classifiers. Finally, this library is developed for real-time gesture recognition project including static postures recognition and dynamic gestures recognition, which means that the supported learning algorithms can achieve gestures spotting by default. With this unique feature, the trained classifier automatically computes the rejection thresholds used to validate continuous data stream.

3.2.4. Classification Module

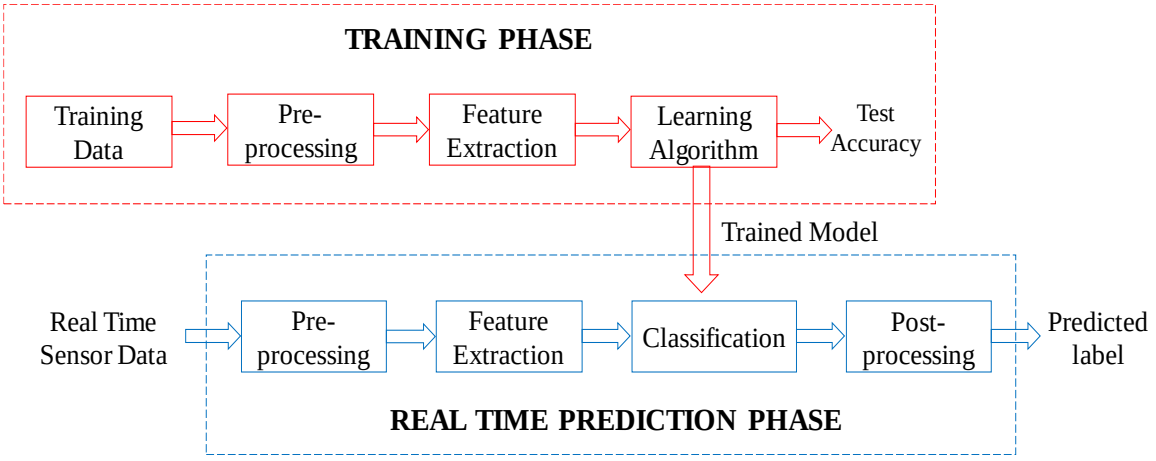


Figure 3.10 Flowchart of the Classification Module

The proposed classification module of software architecture plays a crucial role in the

whole interaction system, and it includes two major phases: data training phase and real-time prediction phase (as shown in Figure 3.10). The primary function of this module is to transfer the real-time gesture data from a sensor into the predicted gesture label send to the robotic hand. To achieve this purpose, a classifier should be trained during the training phase before we implement real-time hand gesture classification.

The training phase indicates the process of training classifier models, and there are some little differences between static postures training phase and dynamic gestures training phase, as follow:

- For **static postures training phase**, the recorded training data consists of static hand postures of ASL alphabet hand shapes. Then, a moving average filter was implemented to pre-processing these training data. After that, the processed training data will be used to perform feature extraction to extract the training feature of static hand postures. The extracted features contain distances and velocities of fingertips, see Chapter 4 for more details, and it will be fed into the static posture classifier. To be more specific, a multi-class SVM learning algorithm has been chosen as the classifier for static hand postures classification and trained model will be saved for further use in real-time prediction phase. Lastly, a test accuracy of the trained classifier will be given at the end of the training phase.
- For **dynamic gestures training phase**, some predefined simple hand movements have been recorded and saved as the dynamic gestures training dataset. Similar to static postures training phase, the dynamic training dataset needs to be pre-processed to get a precise trained model by trimming those non-movement frames, then, the refined training dataset will be used to extract dynamic features. The extracted dynamic features are more complex than static because of higher dimensions and may differ from Human-Computer interaction to Robot-Robot interaction, see Chapter 5. Besides, DTW algorithm has been selected as the main classifier for dynamic hand gestures classification compared to other algorithms. Same as static training phase, the trained model will be saved for real-time prediction, and the test accuracy will be given.

Although the training details are varying to static and dynamic, the final goal of training phase is to get a precise trained model used for real-time classification.

In real-time prediction phase, the real-time depth data of human-made or robot-made gestures are captured by Leap Motion Controller frame by frame. However, before we send the raw data to the feature extraction step, it must be pre-processed to determine the incoming real-time gesture belongs to static posture or dynamic gesture so that we add an extra velocity threshold of hand movement at pre-processing phase. Next, if the received raw data belongs to static postures, we will implement the static posture feature extraction; otherwise, the dynamic feature extraction method will be performed. Then, these extracted features will be sent to trained classifier model to do real-time prediction and output a predicted gesture label. Moreover, there is an extra post-processing step in prediction phase compared to training phase due to some special cases, see Chapter 4 and 5 for more details. Lastly, a predicted label will be presented and sent to the destination robot B, then the whole process of classification module has been completed.

3.3. Limitations

3.3.1. Limitation of the Robotic Hand

The robotic hand is the central unit of the presented system that interacts with human or another robot, and it is also the bridge to connect the user and computer. Thus, the limit of the robotic hand will determine the overall performance of the entire system because we will use this hand to repeat human's or the other robots' gestures as much as we can. This limitation mainly refers to the degree of freedom (DOF):

1. *Five DOF of the robotic hand*: As we can see in section 3.1.1, each finger is controlled by one linear motor inside the palm cover. It also uses a fishing line to attach all the tendons of the finger by connecting the fingertip and the linear motor so that the fingers can be actuated. However, this hardware structure is exactly the most significant limitation of the robotic hand because all the tendons or phalanges have to move together, result in only five degrees of freedom can be used. Also, the bending direction of each finger is fixed so that this hand cannot perform the complex handshapes, like the finger twists.
2. *One DOF of the wrist part*: Because the original robotic hand does not contain the wrist part, we assembled the with forearm together to make the robotic hand can perform rotation movement for some dynamic gestures. But due to the material and

time limit, there is only one servo motor installed inside the wrist part, also refers to one degree of freedom.

3. *Loss of the fishing line*: This limitation indicates that the fishing line which binds the tendon and motor will loose with the increasing experiment times. This defect is inevitable so that we need to fasten it or re-tie the connections to avoid the inaccurate effect of the handshape time to time.

The defects 1 and 2 of the robotic hand are the main limitations that decline the possibility of repeating postures and gestures, result in the training samples have to be selected in a limited range.

3.3.2. Limitation of the Leap Motion Controller

The Leap Motion Controller is depth camera used to capture hand gestures input and record the training dataset. The accuracy of the Leap Motion will directly influence the quality of training samples and the recognition rate. Even the Leap Motion Controller is one of the most accurate 3D depth sensors in the market regarding the hand tracking, and there are still several defects when we use it:

1. *Communication interruptions*: The communication interruption indicates that the PC loses signal from Leap Motion Controller sometimes, result in getting invalid hand tracking frame or frames, depends on the duration of the signal interruption. It mainly caused by weak USB contact or the short break of process *Leap Service*, and the probability of this phenomenon is relatively low and usually happens at the beginning or the end of a posture recording.
2. *Meaningless finger tracking data*: This disadvantage is mainly caused by the relative position and orientation of the hand about the camera. When some fingers are overlapping, the Leap Motion Controller may detect the forefingers as the back fingers and vice versa, result in the incorrect finger tracking data recorded in the current frame. If these false frames still exist in the training dataset, the accuracy of the trained model will be affected.
3. *Poor working area*: As we know in section 3.1.3, the working area of the device is above the surface 25 millimeters to 600 millimeters with a field of view of 150 degrees. However, this working area might be compressed in some cases.

Above three limitations are all caused by the camera itself and can be improved or effectively avoid by recalibrating the Leap Motion Controller using the default software.

3.4. Conclusions

In this Chapter, we introduced the architecture of the presented hand gesture interaction system which contains two major parts: hardware and software. Then, section 3.2 presented all the hardware components built in this project, including the robotic hand, the robotic arm (forearm), and the Leap Motion Controller. Especially, the robotic hand and arm are assembled to perform the hand gestures, while the Leap Motion Controller is the sensor receiving the gesture inputs. Meanwhile, section 3.3 described the corresponding software parts used to control the above hardware components. It consists of a high-level control system, ROS framework, and two libraries, GRT and LeapSDK, to take charge of classification module of Leap Motion device. Besides, a classification module presented and described in section 3.3.4. which plays a key role in in the system. The next chapters will explain how to classify static hand postures and dynamic hand gestures based on classification module flowchart.

Chapter 4. Classification Module for Static Hand Postures Recognition

This chapter presents the methodology used for the real-time static hand postures recognition, corresponding to the classification module shown in section 3.3.4 for both the human-robot interaction (HRI) and the robot-robot interaction (RRI) respectively. Section 4.1 describes the raw training dataset for static postures recognition. Then, section 4.2 shows how the raw static hand postures data are pre-processed to get a better training dataset, and section 4.3 explains what kind of features from static postures training data should be used for classification. Section 4.4 which gives a detailed description and comparison of three machine learning methods we used for static postures recognition. The experimental results discussed in section 4.5, explaining why we decided to select the multi-class SVM classifier for static posture recognition in our system.

4.1. Training Dataset for Static Hand Postures

The training dataset of static hand postures consists of ten digits and selected letters based on American Sign Language. It composes of hand tracking frames captured by Leap Motion Controller and contains all the information which allows us to extract the features as we need. As the handshapes of some ASL letters are very similar to digit handshapes, only those letters with obviously different handshapes have been used in the training dataset. For instance, as the posture for the letter ‘V’ is same as that for the digit 2 in terms of handshape, orientation, and location of each finger, the letter ‘V’ has been removed from the training dataset, even there is a little discrepancy in the position of the thumb finger.

The selected letters are varying to the different interaction modes because of the flexibility of human’s hand and robotic hand. Because the human’s hand is more flexible than any robotic hand, and it can perform the complicated hand shapes, like the fingers twist. This is a significant feature in ASL communication because some of the letters or gestures involve complex handshape, such as the letter ‘R’ (see Figure 2.1). Due to the limitation of the degree of freedom (DOF) and flexibility, the robotic hand can only repeat

those hand postures which do not require orientation changes of the wrist part. Thus, some letters have been removed from the robot-robot interaction mode.

In both the human-robot interaction mode and the robot-robot interaction mode, the ten digits (0 to 9) hand postures in ASL have been used as static posture training data. Because the handshape for the digit 0 is same as that for the letter ‘O’, we removed the duplicated letter ‘O’ from both modes. It should be noticed that all the handshapes for the nine digits except for digit 0 do not involve the wrist rotation, which means the palm always has to face the camera or the observer. In the case of the hand posture for the digit 0, the wrist will rotate 90 degrees around the arm direction. The set of all ten digits postures is shown in Figure 4.1.

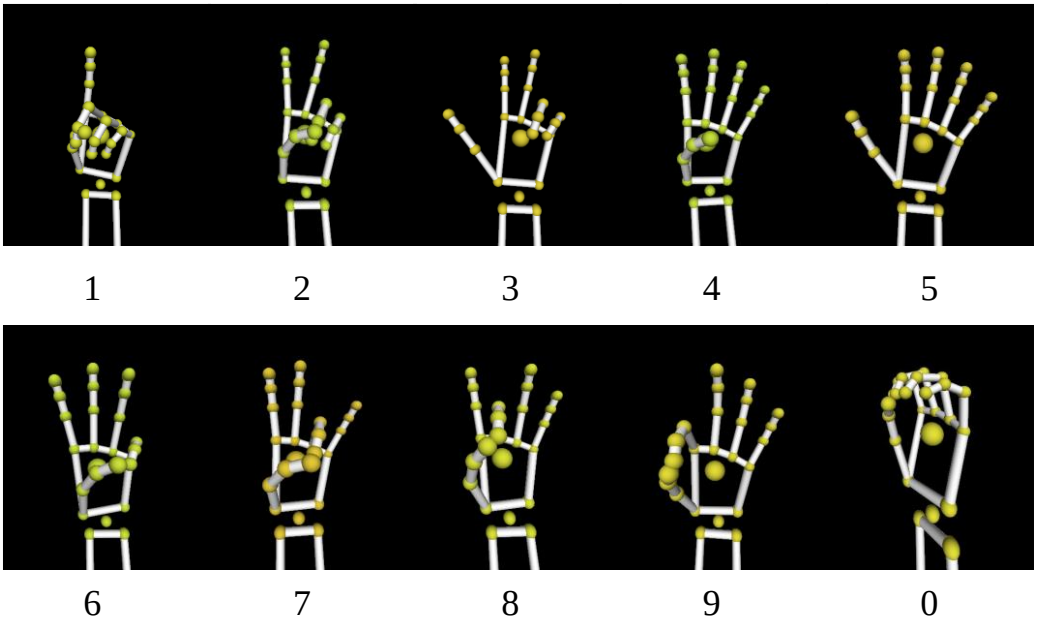


Figure 4.1 Ten digits recorded by the Leap Motion Controller

The *posture for the digit 1*: only the index finger fully extended, and the thumb finger is closed on the middle finger. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 2*: the index and middle fingers are fully extended, and the thumb finger is closed on the ring finger. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 3*: the index, middle, and thumb fingers are fully extended,

ring and pinky fingers are closed. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 4*: only the thumb finger is closed on the palm, other fingers are fully extended. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 5*: all fingers are fully extended. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 6*: the index, middle, and ring fingers are fully extended, the thumb and pinky fingers are closed together. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 7*: the index, middle, and pinky fingers are fully extended, the thumb and ring fingers are closed together. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 8*: the index, ring, and pinky fingers are fully extended, the thumb and middle fingers are closed together. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 9*: the middle, ring, and pinky fingers are fully extended, the thumb and index fingers are closed together. The orientation of the hand is up, and the normal of palm direct to the camera.

The *posture for the digit 0*: all fingers are half extended, and the thumb touches the middle finger to form a circle. The orientation of the hand is up, and the normal of palm has 45 to 90 degrees with the normal of the camera's surface.

There are several discrepancies of the ten digits for robot-robot interaction (RRI) mode and human-robot interaction (HRI) mode. Firstly, the recorded static postures training data for robot-robot interaction mode was performed by the robotic hand, while the human's hand performed training data used for human-robot interaction. Secondly, some digits played by the robotic hand cannot meet the standard handshape of ASL. For example, the posture for the digit 6 performed by the robot hand failed to achieve the connection between the thumb and pinky according to the constraints of a robotic hand, including the maximum bending angle and the bend direction of the fingers. Lastly, there are some position differences of the ring finger between the training datasets for two modes. Due to the

muscular structure and connections of middle, ring, and pinky fingers [94], the ring finger will bend when the pinky finger or the middle finger are bending. Thus, the fingertip's positions of the ring finger will be a little forward compared to the fully extended position, if the postures need to extend the ring finger and close the middle or pinky finger at the same time, like the *posture for the digit 6 and 8*. However, this deviation only happens in the training data for human-robot interaction mode when the poses contain the above situation, and it would not influence the accuracy of the trained classifier.

There are extra postures of letters have been included in training dataset for RRI and HRI, respectively. The following sections 4.1.1 and 4.1.2 will explain which and why the letters have been selected by listing and analyzing the pros and cons from the perspective of fitting this project.

4.1.1. Letter Postures for Robot-Robot Interaction Training

In addition to the ten digits, there were six letters 'A', 'C', 'I', 'L', 'X', 'Y' also included in the training dataset for robot-robot interaction (RRI) mode as the corresponding postures for these letters are distinguishable. There are more letters that were not included in the RRI training dataset because there is a high possibility of the confusion between robot handshapes, as follow:

1. The posture of letter 'B' and that of the digit 4;
2. The posture of letter 'D' and that of the digit 1;
3. The postures of letters 'E', 'S', and 'T' and that of the letter 'A';
4. The posture of letter 'F' and that of the digit 9;
5. The posture of letter 'G' and that of the digit 1;
6. The posture of letter 'H' and that of the digit 2;
7. The posture of letter 'K', 'U', and 'V' and that of the digit 2;
8. The postures of letters 'M', 'N', and 'O' and that of the digit 0;
9. The postures of letters 'P' and 'Q' cannot be performed by robotic hand due to complex handshapes and the direction of wrist rotation;
10. The posture of letter 'R' includes finger twist;
11. The postures of letters 'W' and that of the digit 6.

Moreover, the postures of letters 'J' and 'Z' are not static posture so that they would not be

considered in this training dataset.

The extra six letters included in the RRI training dataset are divided into two categories: (i) those requiring wrist rotation, and (ii) those which do not require wrist rotation, as shown in Table 4.1 and 4.2.

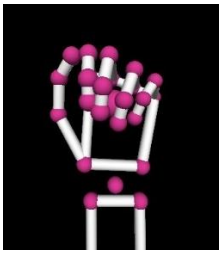
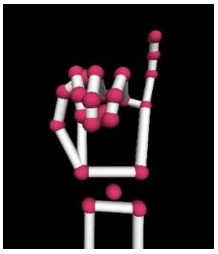
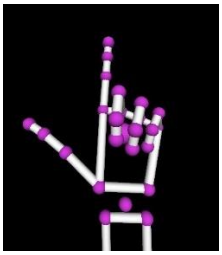
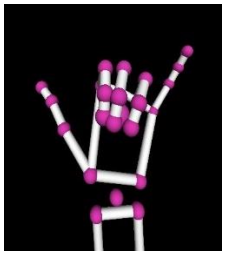
Letter	A	I	L	Y
Illustration (left-handed)				
Robotic hand (left-handed)				
Leap Motion image (left-handed for both human-made and robot)				
Description	Extended fingers: None; Palm's normal direction: camera	Extended fingers: Pinky; Palm's normal direction: camera	Extended fingers: Thumb, index; Palm's normal direction: camera	Extended fingers: Thumb, pinky; Palm's normal direction: camera

Table 4.1 The letters that do not require wrist rotation: 'A', 'I', 'L', and 'Y'

The postures for 'A', 'I', 'L', and 'Y' do not require the wrist rotation as shown in Table 4.1. Similar to the postures for the ten digits 1 to 9, the normal of the palm in the postures for these four letters is always facing the camera. The second raw of the table shows how the robotic hand performs the postures used for the training dataset. There is a little discrepancy in the position of the pinky and thumb fingers when the robotic hand plays the posture for letter 'Y' because it lacks the degree of freedom to wave. Lastly, the third raw displays the depth tracking data of hand skeleton provided by the LMC.

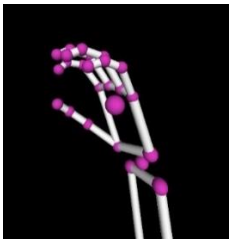
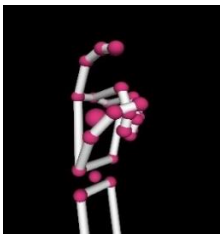
Letter	C	X
Illustration (left-handed)		
Robotic hand (left-handed)		
Leap motion image (left-handed for both human-made and robot)		
description	Extended fingers: All; Palm's normal direction: +45 degrees of camera	Extended fingers: index; Palm's normal direction: -45 degrees of camera

Table 4.2 Letters that require wrist rotation for RRI training: 'C', and 'X'

The hand postures for letters 'C' and 'X' require wrist rotation as shown in Table 4.2. The hand posture for letter 'C' requires a 45 degrees positive rotation around the wrist while the posture for letter 'X' requires a 45-degree negative wrist rotation. There are 100 frames recorded for each of these letters.

4.1.2. Extra Letter Postures for Human-Robot Interaction Training

In the human-robot interaction (HRI) mode, we used extra three letters 'B', 'F', and 'U', as shown as in Table 4.3, on top of the letter postures 'A', 'B', 'C', 'F', 'I', 'L', 'U', 'X', 'Y' used in the RRI static training dataset. While a human hand can easily perform all the postures of ASL alphabet, we added only these three extra letter postures because of the Leap Motion Controller limitations (see section 3.4.2). For instance, the thumb finger is hidden by other four fingers in the handshape of letters 'M' and 'N' so that the Leap Motion

Controller cannot detect the specific position of thumb finger. Another primary reason for rejecting some letters is about the slight and unnoticeable differences between postures as explained in section 4.1.1.

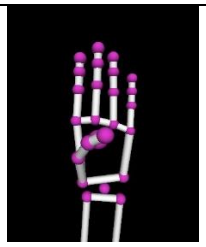
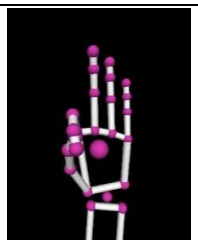
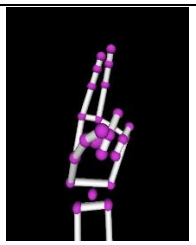
	B	F	U
Illustration (left-handed)			
Robotic hand (left-handed)			
Leap motion image (left-handed for human-made only)			
description	Extended fingers: All except thumb; Palm's normal direction: camera	Extended fingers: middle, ring, pinky; Palm's normal direction: camera	Extended fingers: index, middle; Palm's normal direction: camera

Table 4.3 Extra letters used in the HRI mode: "B", "F", and "U"

Because differently from the hand postures training dataset used for the RRI mode, the hand postures used in the HRI mode could be anywhere of the Leap Motion working area in the real-time prediction phase. We, therefore, recorded 10,000 frames per each letter to handle these position variations. These 10,000 frames for each letter were taken from 5 different relative positions in front of the camera (2,000 frames per each position), including *up*, *down*, *left*, *right*, and *centre* positions.

The hand position of *up* has an angle of approximately 45 degrees upward to the horizontal direction of the camera; the position of *down* indicates there is 45 degrees downward to the horizontal of the camera; the left means the relative position of hand is located

at the left side of the vertical direction of the camera, while *right* represents the hand has a rightward angle to the vertical direction of the camera; the hand position of centre is at the opposite of the camera's surface. In each position, the direct distance from the hand to the camera will be always the same and approximately 35 centimeters, moreover, the normal of the hand's palm always direct to the camera.

4.2. Pre-processing

Pre-processing of the static training dataset is a necessary step before feature extraction from data because these data could be affected by hardware communication interruption, distal phalanges vibration, and meaningless tracking data.

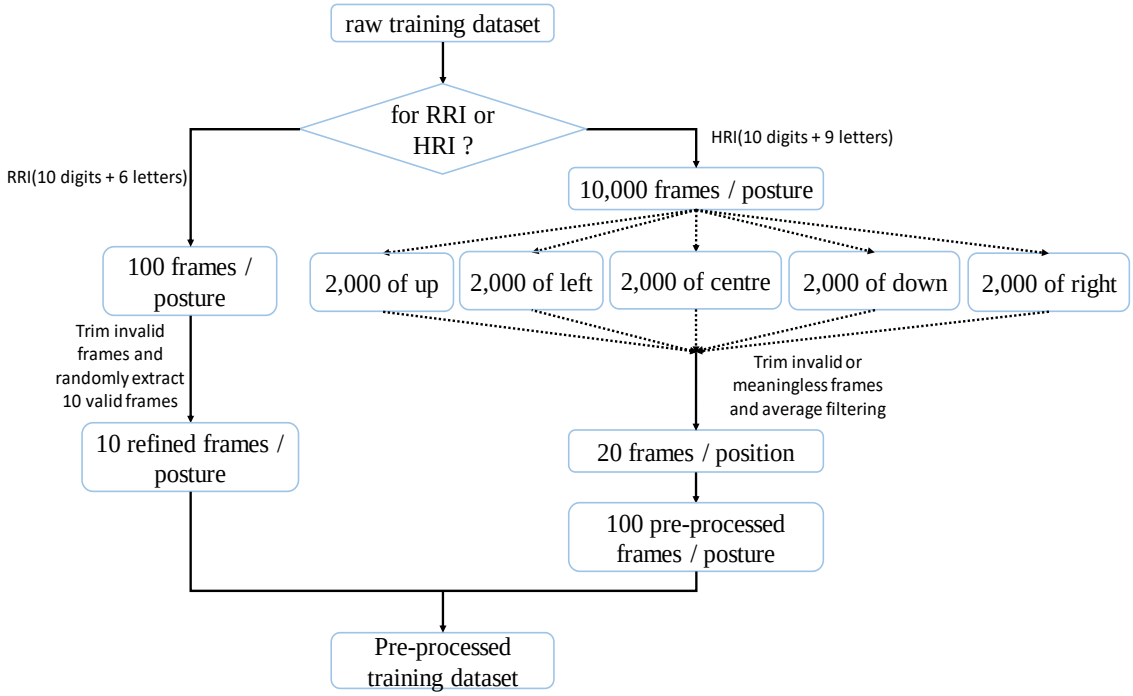


Figure 4.2 Flow chart of pre-processing for RRI and HRI

In both human-robot interaction and robot-robot interaction modes: the raw static posture training data contains invalid frames more caused by communication interruption, see section 3.3.2. To deal with this hardware issue, we will have to eliminate these invalid tracking frames from the 100 frames per posture in the HRI mode and the 10,000 frames per posture in the RRI mode, respectively.

In the human-robot interaction mode, there is another problem to deal with distal

phalanges vibration. This is due to the subtle jitter/vibration of the distal phalanges of each finger that may occur during the posture recording. A velocity threshold of 10 millimeters per second was set to eliminate the postures affected by massive amounts of vibration. Besides, we also implemented an average filter to correct the position of fingertips in those recorded frames by calculating the average positions of every 100 frames to extract 20 refined frames per posture, as shown in Figure 4.2.

In the human-robot interaction mode, it may happen that some meaningless frames recorded. This is due to another limitation of the camera (see section 3.3.2) that produces a mismatch of the finger positions when the forefingers are blocking the view of the rear fingers. To deal with this problem, we have not only to eliminate the incorrect frames, but also need to manually adjust the orientation of the hand postures to make all fingers could be separately detected.

After these pre-processing steps for both RRI and HRI, we obtained a clean static postures dataset consisting of 160 RRI frames and 1,900 HRI frames, which are used in the next step, the feature extraction.

4.3. Feature Extraction

Feature extraction is a crucial step before we implement the classifier training because it decides what kind of features will be fed into the static posture classifier. Furthermore, the selection of features from static training dataset will directly influence the trained model and the recognition accuracy in many cases.

The extracted features will be same for both HRI and RRI modes. The tracking data consists of a **9-dimensions vector** from each frame of the pre-processed training data (see section 4.2) obtained after repeating experiments and comparing the results, so it provides the best balance between the scale of data and the information it contains.

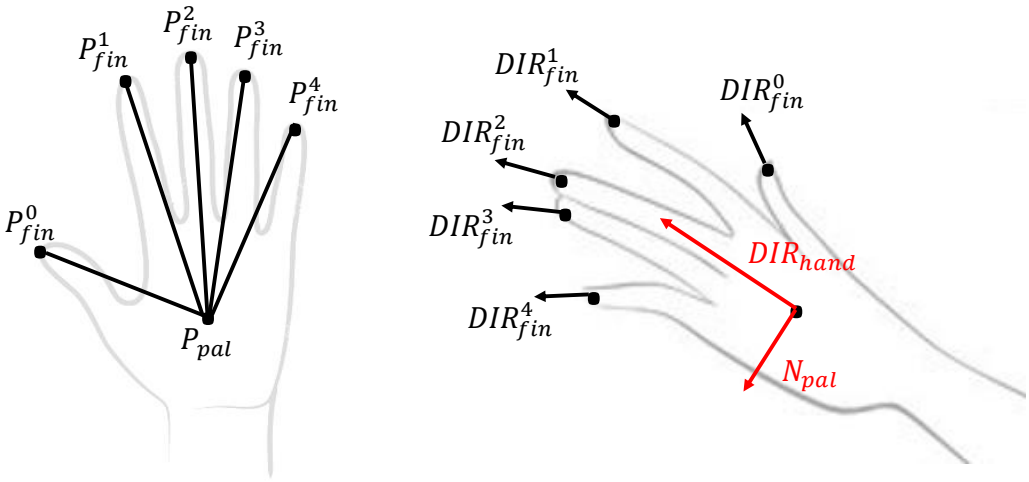


Figure 4.3 Illustrations of the hand tracking data for feature extraction

Figure 4.3 shows the 9-dimensions tracking data, and how this vector was calculated as described as following,

- Palm's position P_{pal}, normal N_{pal}, and velocity V_{pal};
- The fingertips position P^i_{fin}, velocity V^i_{fin}, and direction DIR^i_{fin} (where *i* representing the finger index: 0-thumb, 1-index, 2-middle, 3-ring, and 4-pinky);
- Hand direction DIR_{hand}.

The features for the static training dataset could mainly be divided into two parts. One part is associated with the distances between palm's position to each fingertip, represented by D^i_{p2f}, which occupied total 5 dimensions and represented by:

$$D^i_{p2f} = |P_{pal} - P^i_{fin}| \quad (1)$$

where $i = 0 \sim 4$ indicates five fingers in order from thumb to pinky. The second part is associated with the absolute distances of fingertips positions between two adjacent fingers, computed as:

$$D^{ij}_{f2f} = |P^i_{fin} - P^j_{fin}| \quad (2)$$

where *i* and *j* are the indices of the fingers, for example, D^{01}_{f2f} means the distance of fingertips between thumb and index fingers, and it consumes 4 dimensions of the vector. The resulting 9-dimensions feature vector representing the static training data that will be fed

into the learning algorithm to train the classifiers.

All the static features are stored in .grt file format [92] and labelled as a unique class for each posture. As there is a 9-dimensions feature vector for each frame, there are total 160 vectors for the RRI mode and 1,900 vectors for the HRI mode.

4.4. Learning Algorithm

Three popular learning methods for hand gesture classification have been selected and implemented in this section in order to find the most suitable one: The Adaptive Naïve Bayes Classifier (ANBC), the k-Nearest Neighbours (k-NN), and the multi-class Support Vector Machine (SVM). They are all supervised learning algorithms and need to be fed with training data with labelled gesture classes. A test accuracy will be given for each classifier using the training dataset itself to test the real-time prediction.

4.4.1. Static Hand Postures Recognition with ANBC

Adaptive Naïve Bayes Classifier (ANBC), presented in [95], is a basic statistical classifier, particularly suitable for gestures recognition. The Naïve Bayes classifiers have been used to deal with a wild range of classification problems, such as text classification [96] and gesture recognition [97], for which a mathematical model can be deducted from the basic Bayes' probability calculation. ANBC is a simple classifier that gives some strong assumptions of each training data input (the 9-dimensions vector in our case), most notably all the dimensional values in the vector are relatively independent. The Adaptive Naïve Bayes Classifier has an adaptive online training phase which provides the classifier with relatively high accuracy for recognizing the hand gestures, by refining the original trained model during the real-time prediction phase. This adaptive feature allows the algorithm to continuously train and perfect the model of each gesture by adding the latest real-time predicted result to the original model, and then re-compute the model using appended labelled data. Moreover, another significant feature of ANBC is the weighting factors which gives the user freedom to weight the essential dimensions of the training data and makes the trained model more precise.

The ANBC uses the Bayes' theorem to represent the posterior probability of an event A given the data of event B, by:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3}$$

where $P(A)$ represents the prior probability of the event A happening, and $P(B)$ represents the probability of the event B.

In the hand gesture classification problems, the calculated posterior probability $P(A|B)$ of event A using equation (3) is actually representing the probability of each dimension value of input vectors occurring given each class probability, where $P(A)$ is now the conditional probability of each input dimension given the gesture class and $P(B)$ is the probability of this gesture occurring. Based on this cogitation, the likelihood of gesture G_k ($k \in [1, K]$) can be calculated from the probability associated with the real-time sensor data d:

$$P(G_k|d) = \frac{P(d|G_k)P(G_k)}{\sum_{i=1}^{K} P(d|G_i)P(G_i)} \tag{4}$$

In equation (4), the probability of gesture class $P(B)$ is the sum of all the K gestures, $P(G_k|d)$, in the trained model happening when given the real-time sensor data. The prior probability of gesture k is equal to $1/K$ because we manually set the same number of training vectors for each gesture.

The equation (4) can be updated to calculate the class probability of gesture k when the given the real-time sensor dataset contains the N-dimensional vector D, since the strong assumptions for each dimension have been made independently, by:

$$P(G_k|D) = \frac{P(D|G_k)P(G_k)}{\sum_{i=1}^{K} P(D|G_i)P(G_i)} \tag{5}$$

where $D = \{d_1, d_2, \dots, d_N\}$, $N = 9$ in our case, and the conditional probability of N-dimensional vector $P(D|G_k)P(G_k)$ can be calculated by:

$$P(D|G_k)P(G_k) = \prod_{1}^{N} P(D_i|G_k)P(G_k) \tag{6}$$

Here, a Multivariate Normal Distribution (Multivariate Gaussian Distribution) has been used as the density function due to the advantage of handling the continuous-valued vector D of a specific gesture k.

Lastly, an adaptive re-training process can be implemented after the classification model has been trained. During this phase, the algorithm uses a fixed size buffer to store

the latest predicted gesture as the input training data and pop out the oldest data to update the trained model with the streaming data. Meanwhile, the algorithm allows adding weights for those dimensions of training data that the user considers necessary, which is helpful to distinguish the two similar hand gestures in our case.

After all, the final prediction of input data can be made by finding the maximum likelihood among with all labelled gestures given the current input gesture from sensor data, by equation (7):

$$\arg\max k \quad P(G_k|D) = \frac{P(D|G_k)P(G_k)}{\sum_{i=1}^{K} P(D|G_i)P(G_i)} \qquad 1 \le k \le K \tag{7}$$

4.4.2. Static Hand Postures Recognition with k-NN

K-nearest neighbour (k-NN) algorithm is a well-known supervised learning algorithm that can be used to solve classification and regression problems. Although it belongs to the supervised machine learning algorithm, in fact, k-NN classifier learns nothing from the training data and uses the entire training dataset to perform prediction, which is called *lazy learning*. Thus, the training phase is not a necessary step before the real-time prediction phase since the predicted label for the current input gesture will be computed during the prediction period.

The idea of the k-NN used for classification is to find the majority from the k-nearest instances which can be calculated based on geometric distance measurement between the input sample and the labelled training samples, and each instance among the k-nearest group represents the corresponding gesture class. The majority class will be the prediction of the k-NN classifier. In most real-world problems, Euclidean distance is the most popular geometric distance metric and a perfect choice in hand gesture recognition because the variables of the training data possess the same type, the absolute positions in the 3-D space. However, it can be replaced with other distance metrics according to the specific training data type, such as Hamming Distance [98], Manhattan Distance [99], and Minkowski Distance [100]. The Euclidean distance between $D(g_i, g_j)$ is defined as:

$$D(g_i, g_j) = \sqrt{(g_{i1} - g_{j1})^2 + (g_{i2} - g_{j2})^2 + \ \dots + (g_{iN} - g_{jN})^2} \tag{8}$$

where the $g_i, i = (1, 2, 3, \dots, K)$, is the input vector sample with N-dimensional features

$\{g_{i1}, g_{i2}, \ldots, g_{iN}\}$, and K equals to the total number of input gestures; g_j represents the training samples.

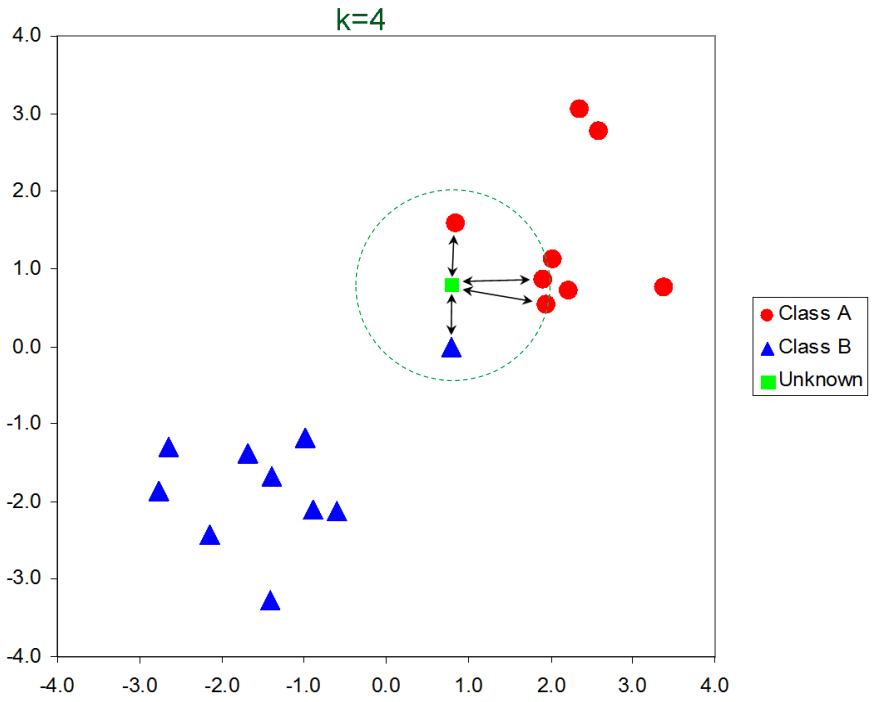


Figure 4.4 Illustrations of how k-NN algorithm predicts a 2-D class, when k = 4. [101]

Figure 4.4 shows the 4-nearest neighbour classifier that predicts an unknown 2-dimensional class by finding the majority of its four nearest instances. The prediction will be the class A.

Algorithm 1. Pseudocode of k-Nearest Neighbour classifier

Input: N-Dimensional training data, k-value
Output: the predicted label
begin
  Classify (M, N, x) // M: training data, N: class labels of M, x: sample to be predicted
  for i=1 **to** n **do**
    Calculate Euclidean distance $D(X_i, x)$;
  end for
  Calculate the set I which has the indices for the k smallest distances $D(X_i, x)$.
  return majority label for $\{Y_i$ where $i \in I\}$
end

Table 4.4 Pseudocode of k-Nearest Neighbour algorithm

In the case of our hand gesture classification problem, the dimensions of input vector will be increased from 2 to 9 dimensions. For each input gesture, we compute the Euclidean distances between this gesture and all samples in the training dataset and sort the distances with ascending order. Then the smallest K distances can be obtained and take place the majority label as the prediction label. Table 4.4 shows the pseudocode of k-NN classifier.

4.4.3. Static Hand Postures Recognition with Multi-Class SVM

The multi-class Support Vector Machine classifier also is a supervised learning algorithm based on Support Vector Machine algorithm, which has a good performance in solving high dimensions and non-linear separable problems, being able to deal with a large range of pattern recognition and classification problems. We used a front-end class of LIBSVM [102], a library for support vector machines (SVMs), providing gesture recognition toolkits (GRT) [92] to implement the SVMs model training and to act the real-time static posture prediction. Different from the previous two simple classifiers, the SVM algorithm is more complicated and has been extended to many sub-SVM algorithms applied to a wide number of real problems.

The basic idea of the SVM method for classification is to find the optimal hyperplane with the maximal margin which separates the training data samples, and then use this hyperplane in the prediction phase when dealing with the input sensor data. The classic SVM algorithm was proposed to handle the binary classification problems by satisfying the following requirements [22]:

$$\begin{aligned} \min_{\boldsymbol{w}\in H, b\in R, \xi i\in R} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{j}\xi i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi i, \\ & \xi i \geq 0, i = 1, 2, \dots, j \end{aligned} \tag{9}$$

where $\boldsymbol{x}_i \in R^d$, $i = 1, 2, \dots, j$ is the training data vector of the two independent classes, $y \in \{1, -1\}^j$ is the vector label, and function $\boldsymbol{\phi}(\boldsymbol{x}_i)$ represents the projection from training data to the high-dimensional vector $\boldsymbol{w}$ in the space H and parameter C is the regularization constant variable.

Using the regulation (9), the training data can be linearly separated by a line (when the dimensions = 2), or hyperplane (when the dimensions ≥ 3), and figure 4.5 shows the hyperplane separating the 2-classes training data with the optimal margin when the training samples are 2-D and 3-D, respectively.

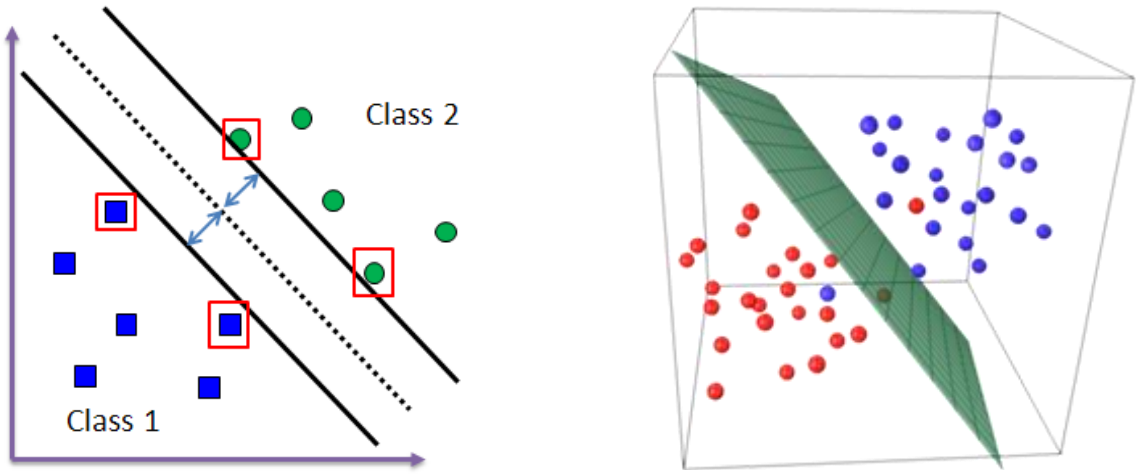


Figure 4.5 The line or hyperplane with maximal margin of binary classification.

Most real-life classification problems, including the hand posture classification, involve multi-class training data, that requires the SVM algorithm to generate more hyperplanes to separate the training samples. As it is not possible to solve the multi-class classification problems using the standard binary SVM classifier, a multi-class SVM classifier was proposed in [23].

The general idea of the multi-class SVM approach is based on "one-versus-one" strategy to construct $k(k-1)/2$ binary sub-classifiers that used to classify two classes using one training data, where k is the number of classes. Given the training samples from i to j classes, the multi-class classification problem can be represented by the following requirements:

$$\min_{\boldsymbol{w}^{ij},b^{ij},\xi^{ij}} \quad \frac{1}{2}(\boldsymbol{w}^{ij})^T\boldsymbol{w}^{ij} + C\sum_t(\xi^{ij})_t$$

$$\text{subject to} \quad (\boldsymbol{w}^{ij})^T\phi(\boldsymbol{x}_t) + b^{ij} \geq 1 - \xi_t^{ij}, \text{ if } \boldsymbol{x}_t \in i\text{th class},$$

$$(\boldsymbol{w}^{ij})^T\phi(\boldsymbol{x}_t) + b^{ij} \leq \xi_t^{ij} - 1, \text{ if } \boldsymbol{x}_t \in j\text{th class},$$

$$\xi_t^{ij} \geq 0 \qquad (10)$$

A majority voting approach is used to achieve the multi-class classification, which applies every binary classifier to perform prediction and vote to the winning class. Since all the classifiers vote for a class, the prediction label can be replaced by the class with the most votes. In addition to above strategy, there are other approaches to implement a multi-class SVM classifier, such as the "one-against-all" [103] strategy or the Weston and Watkins' [104] multi-class SVM.

As our system was designed for real-time prediction, we selected linear kernel as the classifier type since it dramatically boosts training and classification speed and significantly reduces the memory demands. There are several parameters need to be set to get the better result, as follow,

- Scaling training/prediction data to range of [-1, 1];
- A predicted class will be rejected if the classes' probability is below 0.8;
- Main parameters: $nu = 0.05$, $C = 1$;
- Type: C-Support Vector Classification (C-SVC).

where SVM parameters: (i) *nu* represents the maximum percentage of the misclassified training samples in the margin; (ii) penalty parameter C is any positive value which finds the balance between getting the largest minimum margin of the hyperplane and separating the training samples as many as possible. As our training samples are highly linear separable, we set $C = 1$.

The SVM parameters may vary according to the features of the static postures. In this case, we found above parameters are perfectly fit either the human-made postures or the robotic hand postures.

4.5. Real-time Prediction Phase

Using the gesture classifiers trained as described in section 4.4, we can actually implement the real-time prediction. During the prediction period, the process is same as we did in the training phase but there are several differences in the specific steps. The following paragraphs will explain the differences and how to implement real-time postures prediction based on the diagram shown in Figure 4.6.

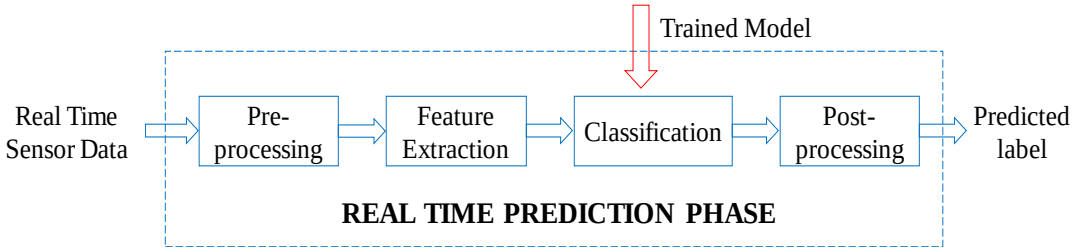


Figure 4.6 The flowchart of the real-time prediction phase

The raw real-time posture data captured by the camera are sent to the prediction module, where the invalid frames caused by communication interruption or finger self-occultation will be removed at pre-processing step and then sent to next step for feature extraction.

Different from the training phase, the velocity features of the five fingertips V_{fin}^{i} and hand's palm V_{pal} will be extracted in order to distinguish between the static hand postures and the dynamic hand gestures. Only those frames with all velocity features less than 10 millimeters per second will be sent to classification step. If there is one or more velocity failed to meet the threshold requirement, then this frame will be removed.

Then, the gesture prediction can be made in classification step using the trained model, where we will use the multi-class SVM as the primary classifier, and the reason will be shown in next section.

Finally, the predicted posture label of the current frame is generated. If the classifier is ANBC, then there is an extra post-processing step before the predicted label come out. Because the ANBC classifier performs an adaptive feature to update the original training samples with the newest labelled class so that the produced label will be stored with all features of this frame in a buffer. Then, the whole process of real-time prediction has been achieved.

4.6. Experimental Results and Analysis

Two experiments were conducted to validate the hand postures recognition in the robot-robot interaction and, respectively, the human-robot interaction mode. We used two pre-recorded test datasets: one test dataset consisting of 3,800 frames of hand postures performed by human's hand to test the HRI trained classifier, and another dataset of 3,200

frames of hand postures performed by the robotic hand to test the RRI classifier.

4.6.1. Results for Human-Robot Interaction Mode

Gesture	Accuracy (%) and Time (ms)					
	SVM		**KNN**		**ANBC**	
One	98	26.49	100	658.9	99	28.87
Two	100	31.21	98.5	493.6	99	24.34
Three	100	29.03	100	579.2	100	27.89
Four	100	28.08	100	581.3	100	26.80
Five	100	29.65	100	557.6	100	23.63
Six	94	21.39	98	538.9	90	25.96
Seven	90.5	29.23	93.5	475.0	87.5	27.81
Eight	100	23.50	100	451.0	95.5	25.27
Nine	100	25.91	100	438.2	97.5	31.01
Zero	100	31.65	100	440.8	100	23.06
Average	98.25	27.614	99.0	521.45	96.85	26.464

Table 4.5 Comparison of the recognition accuracy (%) and prediction time (ms) for 10 digits using multi-class SVM, k-NN, and ANBC in HRI mode.

		Prediction Class (Multi-class SVM)									
		1	2	3	4	5	6	7	8	9	0
Original Class (Multi-class SVM)	1	0.980	0.020								
	2		1.000								
	3			1.000							
	4				1.000						
	5					1.000					
	6		0.060				0.940				
	7							0.905	0.095		
	8								1.000		
	9									1.000	
	0										1.000

Table 4.6 Confusion matrix for 10 digits recognition using Multi-Class SVM classifier (HRI).

Original Class (k-NN) \ Prediction Class (k-NN)	1	2	3	4	5	6	7	8	9	0
1	1.000									
2		0.985					0.015			
3			1.000							
4				1.000						
5					1.000					
6		0.020				0.980				
7		0.010					0.935	0.055		
8								1.000		
9									1.000	
0										1.000

Table 4.7 Confusion matrix for 10 digits recognition using k-NN classifier (HRI).

Original Class (ANBC) \ Prediction Class (ANBC)	1	2	3	4	5	6	7	8	9	0
1	0.990	0.010								
2		0.990				0.010				
3			1.000							
4				1.000						
5					1.000					
6		0.060				0.900				
7						0.040	0.875	0.085		
8				0.045				0.955		
9						0.025			0.975	
0										1.000

Table 4.8 Confusion matrix for 10 digits recognition using ANBC classifier (HRI).

Letter	Accuracy (%) and Time (ms)					
	SVM		**KNN**		**ANBC**	
A	96	30.92	97.5	489.6	96	24.36
B	100	29.13	100	590.4	99	25.63
C	94.5	33.64	96	462.3	88.5	23.84
F	97	26.53	99	437.0	98	30.45
I	100	24.39	100	635.9	100	27.11
L	100	28.26	100	546.2	100	28.78
U	100	30.87	99	501.7	99	25.12
X	98.5	29.56	100	536.4	95	25.32
Y	100	28.25	100	525.0	100	26.70
Average	98.44	29.061	99.06	524.94	97.27	26.368

Table 4.9 Comparison of the recognition accuracy (%) and prediction time (ms) for 9 extra English letters using multi-class SVM, k-NN, and ANBC in HRI mode.

		Prediction Class (Multi-class SVM)								
		A	B	C	F	I	L	U	X	Y
Original Class (Multi-class SVM)	A	0.960		0.040						
	B		1.000							
	C	0.055		0.945						
	F		0.030		0.970					
	I					1.000				
	L						1.000			
	U							1.000		
	X			0.015					0.985	
	Y									1.000

Table 4.10 Confusion matrix for 9 letters recognition using multi-class SVM classifier (HRI).

Original Class (k-NN) \ Prediction Class (k-NN)	A	B	C	F	I	L	U	X	Y
A	0.975		0.025						
B		1.000							
C	0.040		0.960						
F		0.010		0.990					
I					1.000				
L						1.000			
U		0.010					0.990		
X								1.000	
Y									1.000

Table 4.11 Confusion matrix for 9 letters recognition using k-NN classifier (HRI).

Original Class (ANBC) \ Prediction Class (ANBC)	A	B	C	F	I	L	U	X	Y
A	0.960		0.040						
B		0.990		0.010					
C	0.115		0.885						
F		0.020		0.980					
I					1.000				
L						1.000			
U							0.990		
X			0.050					0.950	
Y									1.000

Table 4.12 Confusion matrix for 9 letters recognition using ANBC classifier (HRI).

We evaluated the performance of three classifiers for recognizing ten digits and nine letters by comparing their recognition accuracy using the test dataset and the running time of predicting every 200 test samples, as shown in Table 4.5 and 4.9.

In the ten digits recognition case, Table 4.5 shows that the KNN classifier has the

highest average recognition accuracy of 99.0%, following by 98.25% of multi-class SVM and 96.85% of ANBC. Although k-NN offers the best accuracy, the average running time of this classifier is comparatively slow as we have a lot of data samples as requested by the algorithm in order to calculate the distances and sort them for each prediction. Comparing the multi-class SVM and ANBC classifiers, they all have a good average running time. However, ANBC classifier cannot give an accurate prediction when the data is not linearly separable, or the newest predictions used to re-train are consecutive false. Thus, multi-class SVM classifier shows relatively good accuracy and less running time compared to the other two's.

Table 4.6, 4.7, and 4.8 shows the confusion matrices of the ten digits recognition using different classifiers. According to these matrices, we can see that the predictions of the hand postures for the digit 7 and 8 are severely confused: with a confusion rate of 9.5% for the multi-class SVM classifier, 6.5% for the k-NN classifier, and 12.5% for the ANBC classifier. This confusion mainly caused by the two postures have a similar handshape and the fact of human's muscular structure that the ring finger will bend with the bending of pinky or middle fingers. Another significant misclassification is in the case of the hand postures for the digit 6, where all three classifiers confused it with the digit 2 due to the same reason of ring finger's muscular connection.

For the nine letters recognition, Table 4.9 shows that the k-NN classifier still offers the highest recognition rate of 99.06% against the 98.44% of multi-class SVM and the 97.27% of ANBC. However, the average recognition rates of all classifiers have a slight increase due to the significant differences between the hand postures of these nine letters. The average running time for each classifier remains at the same level as that for the ten digits recognition: with average 26.368 milliseconds for the ANBC, 29.061 milliseconds for the multi-class SVM, and 524.94 milliseconds for the k-NN.

The confusion matrices for the recognition of the nine letters shown in Table 4.10, 4.11, and 4.12 show that a confusion between the postures for 'A' and for 'C' happens in all three classifiers, especially in the ANBC classifier with a confusion rate of 11.5% when predicting posture 'C', 5.5% in the multi-class SVM, and 4% in the k-NN. The ANBC classifier also misclassified the letters 'B' and 'F' with a confusion rate of 2% when predicting posture 'F'.

4.6.2. Results for Robot-Robot Interaction Mode

Gesture	Accuracy (%) and Time (ms)					
	SVM		**KNN**		**ANBC**	
One	100	25.78	100	605.2	100	27.78
Two	100	30.11	100	481.8	100	23.44
Three	100	27.56	100	512.9	100	26.91
Four	100	26.18	100	518.7	98	23.49
Five	100	28.19	100	536.0	100	22.36
Six	100	23.30	100	483.5	97	24.06
Seven	100	24.03	100	493.9	100	25.18
Eight	100	27.05	100	449.7	100	22.72
Nine	100	26.19	100	404.4	100	26.33
Zero	100	31.56	100	413.6	100	24.50
Average	100	26.995	100	489.97	99.50	24.677

Table 4.13 Comparison of the recognition accuracy (%) and prediction time (ms) for 10 digits using multi-class SVM, k-NN, and ANBC in RRI mode.

		Prediction Class (ANBC)									
		1	2	3	4	5	6	7	8	9	0
Original Class (ANBC)	1	1.000									
	2		1.000								
	3			1.000							
	4				0.980		0.020				
	5					1.000					
	6				0.030		0.970				
	7							1.000			
	8								1.000		
	9									1.000	
	0										1.000

Table 4.14 Confusion matrix for 10 digits recognition using ANBC classifier (RRI).

Letter	Accuracy (%) and Time (ms)					
	SVM		**KNN**		**ANBC**	
A	100	33.86	100	542.3	98.5	24.89
C	100	30.48	100	515.8	97	22.71
I	100	22.71	100	463.2	100	26.45
L	100	29.16	100	539.0	100	25.53
X	99	25.63	100	498.5	99	27.38
Y	100	28.79	100	531.7	100	25.23
Average	99.83	28.438	100	515.08	99.08	25.365

Table 4.15 Comparison of the recognition accuracy (%) and prediction time (ms) for 6 extra English letters using multi-class SVM, k-NN, and ANBC in RRI mode.

		Prediction Class (ANBC)					
		A	C	I	L	X	Y
Original Class (ANBC)	A	0.985	0.015				
	C	0.030	0.970				
	I			1.000			
	L				1.000		
	X	0.010				0.990	
	Y						1.000

Table 4.16 Confusion matrix for 6 letters recognition using ANBC classifier (RRI).

As shown in Tables 4.13 and 4.15, the average recognition rates for both digits and letters postures have a slight increase for all the classifiers in the RRI mode compared with those in the HRI mode. The k-NN classifier continued its good performance and dominated the field with 100% recognition rate for both digits and letters. However, the high average running time of k-NN is still the biggest barrier for using it in real-time prediction applications. Meanwhile, the multi-class SVM shows a perfect recognition rate of 100% when handling the ten digits prediction, while it also has a relatively good performance in recognizing the letters with 99.83%. Even the ANBC classifier got the lowest recognition rates of 99.5% and 99.08%, respectively, the level of accuracy is still satisfactory. From the time point of view, both the multi-class SVM and k-NN demonstrate a comparatively lower

average running time at the level of 10 milliseconds per 100 frames.

Table 4.14 and 4.16 are the confusion matrices that show the misclassified postures and the failure rates for the ANBC classifier. As shown in Table 4.14, the ANBC classifier confused the hand postures of the digit 4and 6. Table 4.16 shows a slight confusion for the hand postures of letters 'A' and 'C'. There are no confusion matrices of other two classifiers in the RRI mode because k-NN has 100% accuracy and multi-class SVM has 100% for digits and 99.83% for letters.

There are two reasons for getting a better recognition performance in the RRI mode than in the HRI: (i) the robot hand does not have muscular structure, and (ii) the fixed relative position of robotic hand and camera. The training dataset recorded from the robotic hand is more precise compared to those made by the human hand, as the robotic is not affected by the ring finger's muscular connection.

4.6.3. Comparisons and Analysis

In this section, we will discuss the strengths and weaknesses of each classifier for posture classification from their theoretical perspective by disregarding the negative influence of training data.

ANBC--advantages: (1) Based on Bayes' theorem, this classifier is simple but powerful. (2) ANBC would not suffer from the "curse of dimensionality" [105] because all the input variables, as well as the values with different dimensions, are treated independently so that the computational complexity will not grow exponentially with the increase of the input dimensions. (3) An adaptive feature has been implemented to allow the algorithm to re-train the classifier using the latest prediction samples, so that the original training data can be reduced to a small amount.

ANBC--disadvantages: In the worst case, if the re-training of the classifier starts with some misclassification labels, then the classifier will go "out of control" and get worse and worse with regard to newest prediction.

KNN--advantages: (1) The classifier training phase is not necessary, because k-NN performs prediction using the training dataset itself. (2) the k-NN classifier uses the Euclidean distance metric to measure the prediction, which is a good fit to hand gesture classification, and improves the accuracy by changing the parameter *k*.

KNN--disadvantages: (1) As a classifier, k-NN suffers from the "curse of dimensionality" because it computes the Euclidean distance between the new instance to the training samples and then sorts the distances to find the k-nearest samples. As the computational complexity is dramatically raising with the increasing dimensions of training data, the k-NN classifier cannot be used for real-time applications that need high-dimensions training data.

Multi-Class SVM--advantages: (1) It is a powerful classifier that works well for multi-class classification problems when the training dataset is high-dimensional or not linearly separable. (2) Since it is based on an "one-against-one" strategy, the multi-class classification problem can be downgraded to many binary classification level, which significantly decrease the computational demands.

Multi-Class SVM--disadvantages: (1) In order to get good classification results, there are several important parameters need to be set appropriately, such as *C*, *nu*, *degree*, etc. Finding the best parameters setting by repeating experiments could be quite a time-consuming exercise.

After comparing all the pros and cons, the k-NN method has been excluded due to the high average running time even it has the highest average recognition accuracy. The multi-class SVM has finally been selected as the classifier for the static posture recognition module, as it shows a better performance than ANBC in our case. The experimental results have shown that the multi-class SVM's accuracy is slightly higher than that offered by ANBC while the level of average running time remains the same.

4.7. Conclusions

This chapter discussed design considerations for the classification module of the real-time static recognition of the hand postures for the ten digits and six letters in the RRI mode, and for the ten digits and nine letters in the HRI mode. The experimental results of testing the prediction performance of the three classifiers are shown in section 4.5. A comprehensive comparison of the advantages and disadvantages of the three classifiers has been presented in section 4.6 to illustrate why we choose the multi-class SVM classifier for our system.

Chapter 5. Classification Module for Dynamic Hand Gesture Recognition

This chapter presents the research methodology used for the real-time recognition of dynamic hand gestures. Section 5.1 introduces the dynamic hand gesture training data, and section 5.2 proposes the pre-processing methods. The tracking features used for training dynamic gestures classifiers are presented in section 5.3. Section 5.4 gives the two learning methods, Discrete Hidden Markov Models (DHMM) and N-Dimensional Dynamic Time Warping (ND-DTW), selected for dynamic hand gesture recognition. The real-time prediction phase is presented in section 5.5, and the experimental results are shown in section 5.6. Lastly, a comparison and final analysis are presented in section 5.7 to explain why the ND-DTW is suitable for our problem.

5.1. Training Dataset for Dynamic Hand Gestures

The training dataset of dynamic hand gestures consists of ten dynamic hand gestures performed by human's hand, which can be repeated by a robotic hand. These ten dynamic gestures can be divided into two categories, the gestures without the rotation movement and the gesture with a wrist rotation around the arm direction. Each of the two categories contains five dynamic gestures as shown in Figure 5.1 and Figure 5.2. The non-rotation gestures are *"grab", "click", "1", "4", and "come"*; and the rotation gestures are *"no", "good", "ok", "L", and "love"*.

The non-rotation gesture *"grab"* was created by switching from the posture of the *digit 5* to the posture of the *letter 'A'* and then finally back to the *digit 5*. The orientation of the hand is up and the normal of palm always direct to the camera.

The non-rotation gesture "*click"* starts with posture *digit 5* where all fingers are fully extended, then, the index finger will close or bend a bit to achieve a single click movement, and finally go back to posture *digit 5* to complete the movement cycle. It should be mentioned that the gesture performer needs to reduce the movement of the middle finger caused by the muscular structure for this gesture.

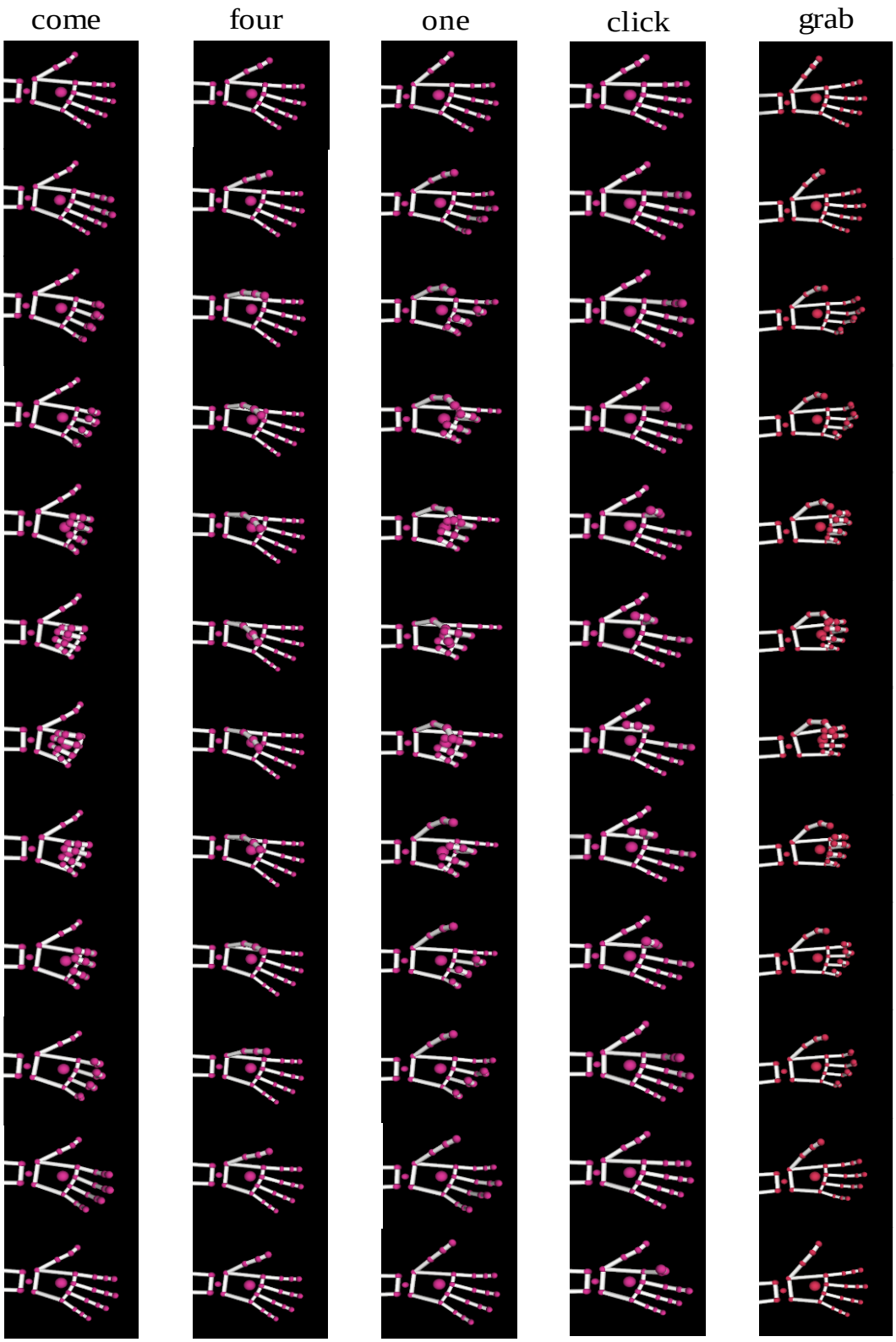


Figure 5.1 The five non-rotation hand gestures: “grab”, “click”, “1”, “4”, “come”.

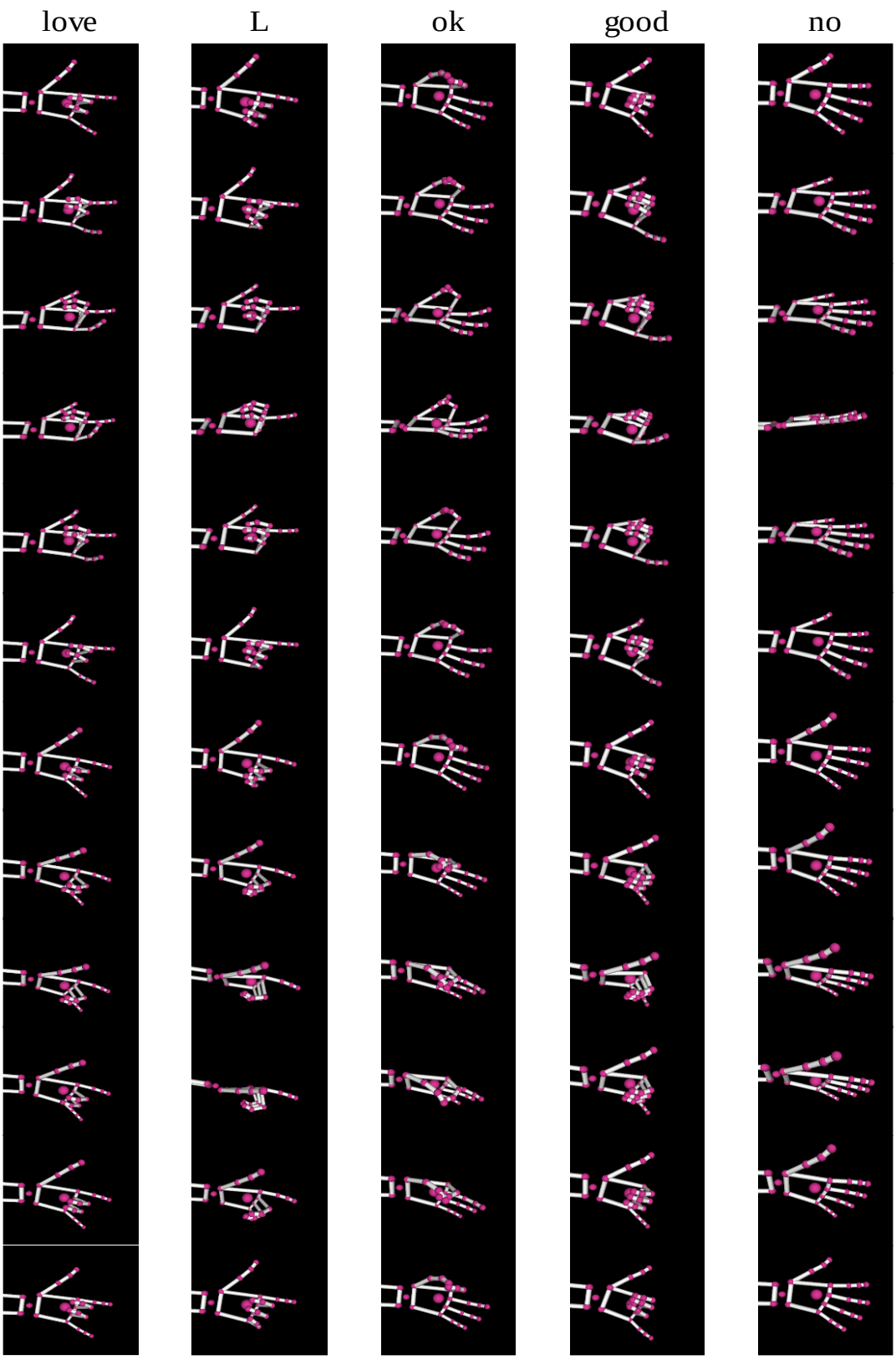


Figure 5.2 The five rotation hand gestures: “no”, “good”, “ok”, “L”, “love”.

The non-rotation gesture *"one"* is the reverse of the gesture *"click"*. It starts with the posture of *digit 5*, then all fingers close except the index which is exactly the posture of *digit 1* and ends with the posture of *digit 5*. This dynamic gesture would not suffer from the muscular effect.

The non-rotation gesture *"four"* was created by switching the initial status of the posture *digit 5* to the posture *digit 4*. During the transition, all fingers except the thumb will remain in the same position and the thumb will first close and then fully extend. Like the "one" gesture, this gesture would suffer from the muscular connection.

The non-rotation gesture *"come"* is like the reverse of the gesture *"four"* because the thumb finger will keep the position and other four fingers move from fully extended to the fully closed during the period. The orientation of the hand is up and the normal of palm direct to the camera.

The rotation gesture *"no"* consists of a wrist rotation movement while the hand-shape remains the same all the time. The initial normal direction of the palm is facing to the camera, then rotates to -45 degrees relative to the normal of the camera, and then rotates to a +45 degrees position. Finally, the wrist will move back to the initial position. The orientation of the hand is always up.

The rotation gesture *"good"* uses the handshape of *letter 'Y'* and the same wrist rotation movement as the gesture *"no"*. The orientation of the hand is up.

The rotation gesture *"ok"* uses the handshape of *digit 9*, which extends the fingers of middle, ring, and pinky, and closes the thumb and index. The wrist will rotate from 0 to -45 and to +45 degrees (the angle of the palm's normal and camera's normal). The orientation of the hand is always up.

The rotation gesture *"L"* uses the handshape of letter 'L' and a rotation movement as other rotation gestures. This dynamic gesture would not suffer from the influence of muscular structure.

The rotation gesture *"Love"* uses handshape with middle and ring fingers closed and other three extended. This gesture will slightly suffer from the effect of pinky muscular. About the trajectory of wrist rotation movement, the angle of the palm's normal and camera's normal will change from 0 to -45 and to +45 degrees. The orientation of the hand is always up.

Due to the hardware limitation, the robot-robot interaction (RRI) mode has been removed from the dynamic hand gestures recognition part, so that all the ten gestures training data were recorded using only the human's hand. Some of the gestures might still be affected by the muscular structure [94] so that the gesture performer needs to avoid quick movement and reduce the ring finger cooperation.

For this ten classes dynamic hand gesture training dataset, we recorded ten training samples per each class, and each training sample consists of 30 frames of data. As due to Leap Motion Controller's performance, approximately 120 frames per second are recorded when a hand is detected, we need to manually setup a time delay to record each frame of the hand tracking data. Because all the ten dynamic gestures are made in the range of 4 seconds to 6 seconds, we set a time delay threshold of 0.2 seconds, which means a frame will be captured every 0.2 second during the gesture recording. After finishing the raw gestures data collection, there are total 3000 frames of hand tracking data have been stored and waiting to be refined for next section.

5.2. Pre-processing

Same as in the case of the static training data, when pre-processing has to be done before training the classifier, the raw dynamic gestures training data will also need to be refined before training the dynamic gestures classifier. The major reasons for pre-processing are hardware communication interruption, meaningless tracking data, and accidental non-movement frames recording.

The causes of hardware communication interruption and the meaningless tracking data have been explained in section 4.2, but there are still some discrepancies in the dynamic gesture training data pre-processing. Firstly, if the invalid frame or meaningless frame occurs with only one adjacent frame (the begin frame or the end frame), then we remove it, otherwise, if this happens during a sequence, it will be replaced with a new frame. As the dynamic hand gestures are consistent and predictable, the new/replacement frame(s) can be calculated using moving average filtering (MA filter) because of the temporal coherence of the training sample frames. For instance (see Figure 5.3), if there are n continuous frames which have the meaningless hand tracking data starting with the ith frame, we will extract the hand tracking data A from the $i-1$ th frame and the hand tracking data B

from the $i + n$ th frame to calculate the distance metric $A - B$. Therefore, the hand tracking data of the ath frame out of the missing n frames can be represented by $a/n(A - B)$. It should be noticed that if the n is greater than 5 (which means we lost at least 1 second's hand movement tracking data), then the complete training sample will be dropped.

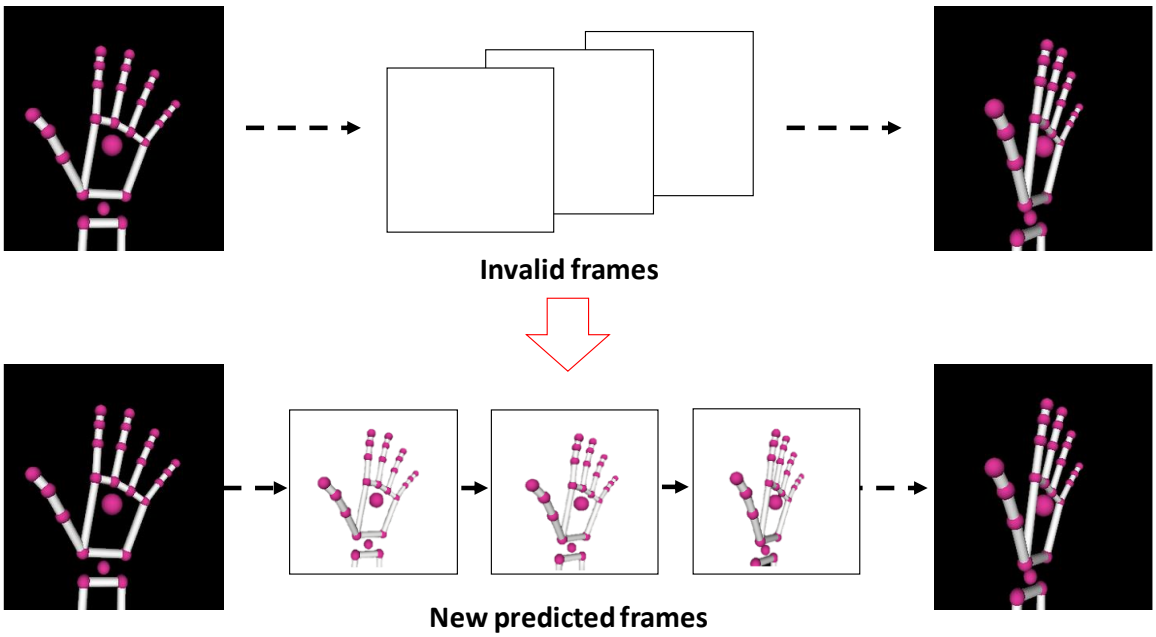


Figure 5.3 Dynamic gesture training sample pre-processing of invalid frames.

Another reason is that we need to preliminary process the non-movement frames of the training samples. Due to the method of recording (we manually start and pause the program to record each training sample), a gesture performing period cannot perfectly match the 30 frames. As a result, there are frames of non-movement or tiny movement also recorded at the beginning or the end of the period. Such a short non-movement period would not significantly influence the correctness of training data, but nevertheless, it will more seriously affect the classifier templates when we have a large number of frames. To fix this problem, we use the non-movement frames trim function of GRT library [92] to trim any sections that have none or very little movement at the start or end of a file. However, if the trimmed frames exceed 30% (equals to 9 frames) of the training sample, then this training sample will be dropped and need to be re-record.

If any of the training samples for one gesture class has been dropped (due to a large

amount of missing or invalid frames), then we need to go back to re-record the training data of this gesture class until it meets the requirements. Eventually, a refined dynamic hand gesture training dataset with ten training samples (for each of the 10 classes), consisting of 25 to 30 frames' hand tracking data per sample, can be obtained and stored for further use.

5.3. Feature Extraction

Different from the static feature case, the dynamic feature extraction is more complicated since a human's hand can perform sophisticated gestures which involve many joints movement in the period. Therefore, higher-dimensional hand tracking features are needed to describe the whole process of dynamic gesture movements. We use a *float matrix* to store the extracted features for each dynamic training sample that consists of continuous frames. For each frame of dynamic training sample, the extracted tracking data consist of the direction of all fingertips DIR_{fin}^{i}, hand palm's normal N_{pal}, as well as the direction of the hand DIR_{hand}, see Figure 4.3.

DIR_{fin}^{i} indicates the distal phalange's direction for each of the five fingers. As each fingertip's direction is a 3-dimensional vector, the DIR_{fin}^{i} contains total 15 dimensions float value, where $i = 0 \sim 4$ indicates five fingers in order from thumb to pinky. Instead of the relative position data of fingertips and palm's centre (the 9-dimensional vector), the direction-based data will be extracted from raw data for static training feature and used to train classifiers. It is mainly caused by the finger bending movements. For instance, the gesture "click" only has the index finger moving as described by the vectors D_{f2f}^{01} and D_{f2f}^{12}, and D_{p2f}^{0}. If the gesture involves more finger movements, then there will be more affected dimensions value, result in lower recognition accuracy of the classifier. However, the direction of fingertips would not be affected because the data will be changed independently, one finger's movement only affects three dimensions data. Thus, we extract the 15-dimensions fingertips direction value as a part of the dynamic feature even it consumes more memories to store.

DIR_{hand} is the hand direction, as well as the hand orientation, that used to make sure all the training samples have an up hand orientation during the movement period. This 3-

dimensional vector will also be used in real-time dynamic gesture prediction phase to refuse the gesture with other hand orientations.

N_{pal} is the most critical feature extracted from the dynamic training samples, which represents the current palm's normal as a 3-dimensional vector. This feature is advantageous when making a distinction between the rotation gestures and non-rotation gestures. If the 3-dimensional values of N_{pal} are not changing during the gesture period, then the classifier will consider it as a non-rotation gesture. Besides, N_{pal} usually perpendicular to DIR_{hand} or the plane of palm so that the relative orientation of the hand in front of the camera will be apparent by giving these two directions.

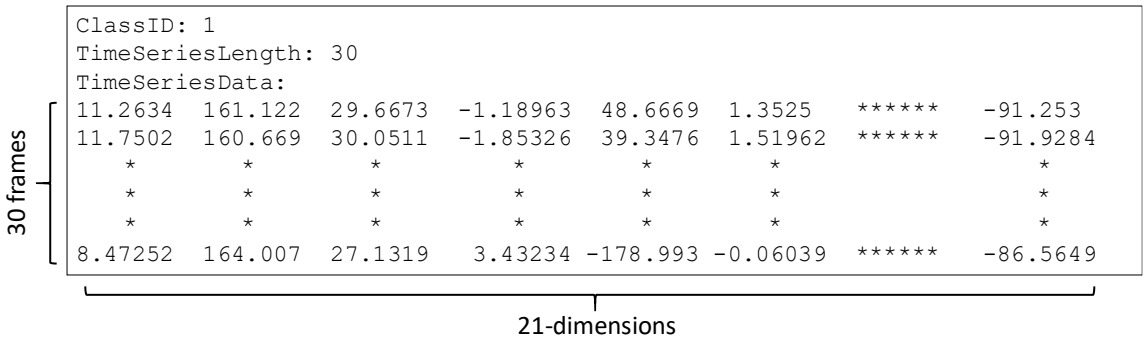


Figure 5.4 The float matrix used to store dynamic training features of one sample

Combining the three different types of feature, we obtain a *21-dimensional vector* for each frame of hand tracking data. And there will be 25 to 30 frames per training sample according to the pre-processing step so that we use a float matrix to store the whole features for the current training sample, as shown in Figure 5.4. Thus, there are total 100 training samples of 10 dynamic gestures stored in .grt file format [92] and labelled a unique number for each class. Finally, the training samples will be fed into the learning algorithm to train the dynamic gesture classifiers.

5.4. Learning Algorithm

There are two popular machine learning methods that work well on temporal classification problems as encountered in our dynamic gestures classifier, the Discrete Hidden Markov Models (DHMM) and the N-Dimensional Dynamic Time Warping (ND-DTW) algorithm.

5.4.1. Dynamic Hand Gesture Recognition with Discrete HMM

Hidden Markov Models (HMMs), is a very powerful and widely used statistical modeling algorithm for addressing three major problems [31]: evaluation, decoding, and training. HMMs are mainly divided into two major types, the Discrete HMM (DHMM) and the Continuous HMM (CHMM). The DHMM used in this section is classic HMM where the hidden states sequence and the input observation sequence are both discrete, different from the CHMM where the input observation sequence could be continuous, usually a Gaussian distribution. The former method is more logical when dealing with the problems that the sequence is not primarily relevant to time but use equally time interval to separate the states, such as speech recognition, gesture recognition, DNA sequence prediction, etc. Meanwhile, the latter approach concerns the events happening in continuous time and are observed by irregular time interval, such as the Poisson point process.

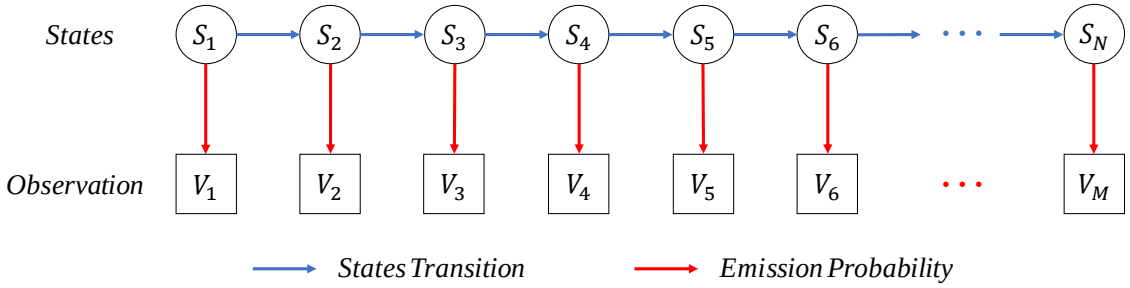


Figure 5.5 The two Markov Chains in a HMM

Typically, a Hidden Markov Model models the real problems by giving two Markov Chains as shown in Figure 5.5, the state sequence (hidden) and the observed sequence (visible). There is a probability distribution for each hidden state in the hidden Markov Chain that can influence the corresponding observation. Therefore, a standard discrete HMM notation [31] can be defined as follow:

1. *States sequence S*: $S = \{s_1, s_2, \dots, s_N\}$ -- set of states, where N represents the total number of states in the HMM, and q_t denotes the state at time t.
2. *Observation sequence V*: $V = \{v_1, v_2, \dots, v_M\}$ – set of observation, where M is the number of observations and given an O_t represents the observation according to the hidden states at time t.

3. *Transition probability matrix A*: $A = \{a_{ij}\}$, it is a matrix of hidden state transition probability distribution that reflects a transform probability a_{ij} from state s_i to state s_j, as follow,

$$a_{ij} = P(\, q_{t+1} = s_j \mid q_t = s_i \,), 1 \le i, j \le N \tag{11}$$

4. *Emission probability matrix B*: $B = \{b_j(k)\}$ is a matrix of the output probabilities distribution controlling the probability of observation symbol v_k given the specific hidden state b_j at time j, as follows,

$$b_j(k) = P(\, O_t = v_k \mid q_t = s_j \,), 1 \le j \le N, 1 \le k \le M \tag{12}$$

5. *Initial state distribution* π: $\pi = \{\pi_i\}$ represents the initial probability distribution at initial state s_i, where

$$\pi_i = P(q_1 = s_i), 1 \le i \le N. \tag{13}$$

Based on above definition, a standard Hidden Markov Model λ with complete parameters can be expressed as $\lambda = (A, B, \pi)$. With the full description and the parameters of HMM approach, we can model many real applications by answering the three fundamental problems of HMM as mentioned above, as follow,

Evaluation Problem: Given the observation sequence $O = \{O_1, O_2, \dots, O_T\}$ and an HMM model $\lambda = (A, B, \pi)$, how to compute the probability of the sequence occurring in given model $P(O|\lambda)$?

Decoding Problem: Given the observation sequence $O = \{O_1, O_2, \dots, O_T\}$ and an HMM model $\lambda = (A, B, \pi)$, how can we choose an optimal corresponding state sequence $Q = \{Q_1, Q_2, \dots, Q_N\}$ that can best describe the observation?

Training Problem: Given the observation sequence $O = \{O_1, O_2, \dots, O_T\}$, how can we modify the parameters in $\lambda = (A, B, \pi)$ to maximize the probability $P(O|\lambda)$?

We represent the training data of a given dynamic gesture as a frames sequence with the dynamic training features which we extracted in section 5.3 so that we can obtain an N-state HMM for each gesture of a G gestures set, where N represents the number of the frames in a gesture sample. We assume the frames of a training sample are independent and we have ten training (observation) sequences containing for each gesture. Therefore, in our case, the problem of dynamic gestures recognition can be done by answering the

three major problems in the order of training, decoding, and evaluation.

The first task is to solve the Training Problem, which needs building 10 individual gesture HMM models. To solve this, we use the Baum-Welch algorithm [31] to estimate the optimal parameters in an HMM $\lambda = (A, B, \pi)$, especially the transition matrix A and emission matrix B, for each gesture model. The second task is to find the solution for Decoding Problem, where we need to find an optimal corresponding hidden state sequence to perfectly explain the observations. Since all the training samples are labelled, then we can simply spot the states sequence with the gesture label and study the observed frames occurring in each gesture state. Besides, we can make an improvement of each gesture HMM by enlarging the modeling capacities using ten training samples. The final task is to recognize an unknown dynamic gesture performed by human's hand using the 10 designed gesture HMMs with the optimal parameters which we obtained from the first task. This goal can be achieved by solving the Evaluation Problem using Forward-Backward algorithm [31], which computes the likelihood of each gesture HMM-based upon the given input gesture sequence and then recognizes the input sequence as the gesture with the highest likelihood.

The HMM used in our system has a little difference between the classical HMM method in the training phase because the total number of frames for one training sample is not always equal to 30 due to the pre-processing step, usually 25 to 30 frames. However, the training sequence fed into an HMM must have equal length. To fix this problem, we use a data alignment algorithm [106] to guarantee that each training sample has the same sequence length in a gesture class.

5.4.2. Dynamic Hand Gesture Recognition with ND-DTW

This subsection will review the multivariate temporal gesture recognition using N-Dimensional Dynamic Time Warping [107]. The original DTW algorithm, which belongs to supervised learning, is a powerful classifier that can compute the likelihood of two identical timeseries even of different lengths. The basic theory of DTW can be described as following:

Given two one-dimensional time series, $a = \{a_1, a_2, \dots, a_M\}$, and $b = \{b_1, b_2, \dots, b_N\}$, where M is the length of timeseries a and N is the length of timeseries b. Then we create a

warping path $w = \{w_1, w_2, \dots, w_P\}$ of length P subject to the following constraints:

$$\max\{M, N\} \leq P \leq M + N \tag{14}$$

The kth element of w is,

$$w_k = (a_i, b_j), 1 \leq i \leq M, 1 \leq j \leq N \tag{15}$$

where a_i is the value of timeseries a at time i and b_j is the value of timeseries b at time j. It should be noted that this warping path must satisfy several constraints: (i) the initial position must at $w_1 = (1, 1)$; (ii) the end position must at $w_P = (M, N)$; (iii) this warping path is a forward path and cannot move back; (iv) it is a continuous path and the distance between two adjacent time must equal to 1. See more details in Figure 5.6.

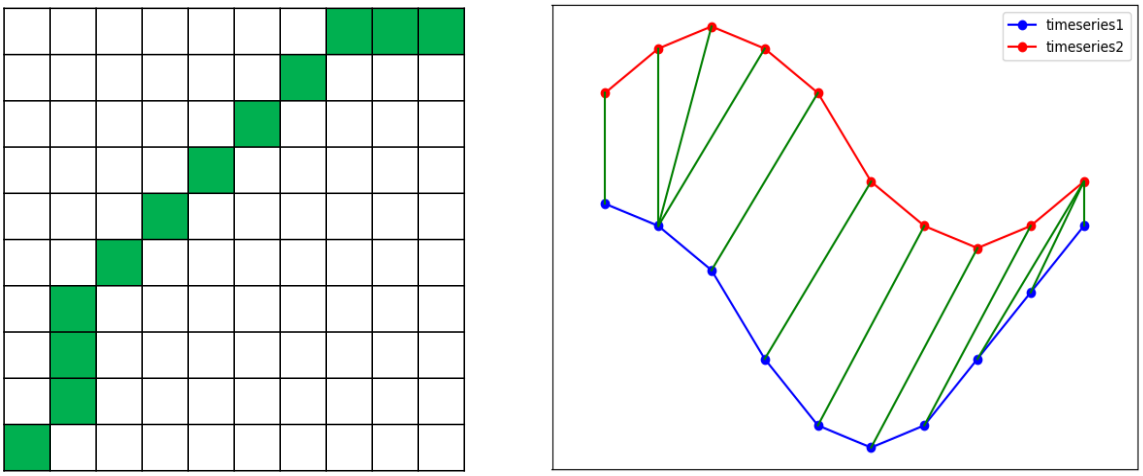


Figure 5.6 Illustration of the cost matrix and alignment between two timeseries.

With the above requirements, DTW overcomes the disadvantages of the unintuitive approach using Euclidean distances to measure the likelihood of the two timeseries. Instead, DTW will find an optimal global alignment, as well as the warping path, with the minimal overall cost C between two timeseries to exploit temporal distortions between them, which can be given by:

$$C = \min\left(\frac{1}{P} * \sum_{k=1}^{P} D(w_{k_i}, w_{k_j})\right) \tag{15}$$

where $D(w_{k_i}, w_{k_j})$ is the Euclidean distance between the points i and j in the two time-series. The corresponding warping path can be obtained by filling a two-dimensional cost

matrix using dynamic programming approach as shown in Table 5.1. After we found the warping path, we use Sakoe-Chiba band [108] as a global constraint to control the size of warping window and improve the computational efficiency of DTW.

Algorithm 2. Pseudocode of warping path optimization in DTW

Input: accumulated cost matrix C with two timeseries $a = \{a_1, a_2, \dots, a_M\}$, $b = \{b_1, b_2, \dots, b_N\}$, where M and N are the number of points.
Output: cost matrix C with optimal warping path w.
begin
  $C = [0, 0] := 0$; //Initialize the cost matrix
  for i = 1 **to** M **do** $C = [\mathrm{i}, 0] := \infty$;
  end for
  for i = 1 **to** N **do** $C = [0, \mathrm{i}] := \infty$;
  end for
  //Start filling the cost matrix using dynamic programming strategy
  for i = 1 **to** M **do**
    for j = 1 **to** N **do**
      cost := $D(a[\mathrm{i}],\ b[\mathrm{j}])$
      $C[\mathrm{i}, \mathrm{j}] := \mathrm{cost} + \min(C[\mathrm{i}-1, \mathrm{j}],\ C[\mathrm{i}, \mathrm{j}-1],\ C[\mathrm{i}-1, \mathrm{j}-1])$
    end for
  end for
  return $C = [M, N]$
end

□□□ **Table 5.1** Pseudocode of warping path alignment algorithm

In our case, the timeseries corresponding to the training frames are all 21-dimensional, so that the measurement of the Euclidean distance between two time-series needs to be re-computed for N-dimensional templates using the approach given in [109] by:

$$D(i,j) = \sqrt{\sum_{n=1}^{N}(i_n - j_n)^2}. \tag{16}$$

We can use this new N-dimensional distance to find the best warping path between two N-dimensional timeseries by finding the minimal global cost,

$$ND - C = \min\left(\frac{1}{P} * \sum_{\mathrm{k}=1}^{P} D\left(w_{k_i}, w_{k_j}\right)\right),$$

$$\text{where } D(i,j) = \sqrt{\sum_{n=1}^{N}(i_n - j_n)^2} \tag{17}$$

After the 10 N-dimensional DTW templates of dynamic gestures have been trained, we can classify a new gesture by calculating the warping paths between the input timeseries

with each of the 10 gesture ND-DTW template. Then the gesture template with the minimal cost warping path will be chosen as the predicted gesture label. However, the above method has a limitation when the new gesture timeseries input does not belong to any of the ten gesture templates, resulting in a false prediction. So, we set an additional global threshold of classification to reject the prediction label if the minimal cost still higher than a specific value, thus, the gesture input has no match.

5.5. Real-time Prediction Phase

Similar to section 4.5, we can implement the real-time prediction phase using the trained model after the dynamic gesture classifiers have been trained. During the prediction period, a post-processing step is added based on the steps in training phase.

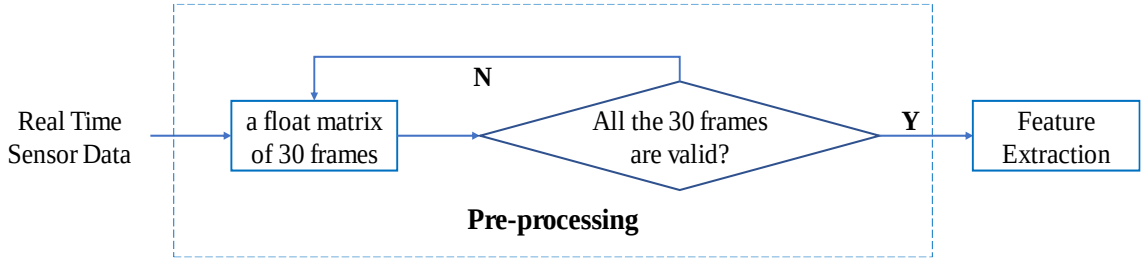


Figure 5.7 The pre-processing step for real-time dynamic gesture prediction

The incoming real-time dynamic gestures raw data captured by the camera and sent to the prediction module is buffered as a 30 * 21 float matrix. The invalid frames caused by communication interruption or finger self-occultation will be removed at pre-processing step. Therefore, if the 30 frames currently stored in the matrix contain any invalid frames, then the whole set of 30 frames will be discarded, and the buffer matrix will store the next 30 frames until all the frames are valid.

Secondly, we extract the velocity features of five fingertips V_{fin}^{i} and hand's palm V_{pal} from the pre-processed 30 frames for real-time prediction to distinguish the dynamic hand gestures from the static hand postures, and only those frames with all velocity feature greater than or equal to 10 mm/s will be sent to classification step. If one or more velocity vectors fail to meet the threshold requirement, the whole float matrix is removed, and the process waits for the next matrix from pre-processing step. Then, the gesture classification

step is carried out using the trained model from section 5.4, and the N-dimensional DTW as the primary classifier.

Finally, a post-processing step will provide the recognized gesture label for both DHMM and ND-DTW approaches. If we use DHMM classifier, the predicted label will be produced by computing the likelihood of the given gesture sequence with each HMM of 10 gestures and find the gesture with maximal likelihood as the prediction. However, if the given is not belong to any of the 10 gestures, the algorithm still has a predicted result with a low likelihood which is clearly incorrect. In order to reduce the chance of this false prediction, we set a low threshold for refusing the prediction likelihood lower than this percentage. A high threshold has been set for the ND-DTW classifier to reject the too high minimal global cost.

5.6. Experimental Results and Analysis

A test dataset has been recorded for testing the recognition rate of two classifiers, which consists of ten dynamic gestures, “grab”, “click”, “1”, “4”, “come”, “no”, “good”, “ok”, “L”, and “love”. Since the RRI mode has been removed for dynamic hand gestures recognition, there are 20 test samples with 30 frames recorded by human’s hand for each of the 10 gestures in Human-Robot Interaction mode. The main parameters used for implementing the DHMM classifier are: type = discrete, model = left-right, delta = 1, downsampling factor = 5, sigma = 20.0, rejection threshold = 0.2; while the parameters of ND-DTW are radius = 0.25, smoothing factor = 5, and rejection threshold = 0.25.

5.6.1. Experimental Results

We evaluated the performance of two dynamic classifiers for recognizing ten gestures by comparing their recognition accuracy when classifying dynamic gestures and the running time of predicting every 20 test samples, as shown in Table 5.2.

Table 5.2 shows that the DHMM method has a higher average recognition rate of 97.0% when compared to the 95.0% of the ND-DTW classifier. Both the DHMM and the ND-DTW methods have the same recognition accuracy of 100% when recognizing the six gestures “grab”, “click”, “4”, “come”, “no”, and “L”, while the major misclassifications occurred in the other four gestures “1”, “good”, “ok”, and “love”.

While the DHMM has a better recognition accuracy, its average running time of 1667.74 ms when predicting every 20 test samples is slower compared to the 189.3 ms of the ND-DTW. Keeping in mind the real-time prediction purpose, we believe the ND-DTW method is more suitable for our system as the primary gesture classifier.

Gesture	Accuracy (%) and Time (ms)			
	DHMM		**ND-DTW**	
"grab"	100	1937.9	100	114.72
"click"	100	1269.8	100	101.12
"one"	95	1584.3	90	200.60
"four"	100	1298.7	100	161.28
"come"	100	1825.8	100	137.66
"no"	100	1695.5	100	325.80
"good"	90	1735.9	85	261.31
"ok"	95	1627.8	90	292.40
"L"	100	1747.5	100	172.42
"love"	90	1954.2	85	125.75
Average	97.00	1667.74	95.00	189.301

Table 5.2 Comparison of Recognition Accuracy (%) and Prediction Time (ms) for 10 Dynamic Gestures using ND-DTW and DHMM

		Prediction Class (DHMM)									
		"grab"	"click"	"one"	"four"	"come"	"no"	"good"	"ok"	"L"	"love"
Original Class (DHMM)	"grab"	1.000									
	"click"		1.000								
	"one"	0.050		0.950							
	"four"				1.000						
	"come"					1.000					
	"no"						1.000				
	"good"							0.900			0.100
	"ok"								0.950		
	"L"									1.000	
	"love"							0.100			0.900

Table 5.3 Confusion matrix for 10 gestures recognition using DHMM classifier.

Original Class (ND-DTW) \ Prediction Class (ND-DTW)	"grab"	"click"	"one"	"four"	"come"	"no"	"good"	"ok"	"L"	"love"
"grab"	1.000									
"click"		1.000								
"one"	0.100		0.900							
"four"				1.000						
"come"					1.000					
"no"						1.000				
"good"							0.850			0.150
"ok"						0.100		0.900		
"L"									1.000	
"love"							0.100			0.850

Table 5.4 Confusion matrix for 10 gestures recognition using ND-DTW classifier.

Table 5.3 shows the confusion matrix of the ten dynamic gestures recognition using DHMM, where we can see that the predictions of gesture "good" confused with the gesture "love". This confusion mainly caused by the two gesture sequences have a similar hand-shape, the only difference is the extended index finger in gesture "love", and same movement trajectory during the period. Another misclassification occurred when predicting the gesture "1", which could be caused by either the similar hand movement or limitations of the Leap Motion Controller.

Table 5.4 displays the confusion matrix of ten dynamic gestures recognition of ND-DTW. Similar to the DHMM classifier, the gesture "love" and "good" still confused each other with a rate of 10% and 15%, respectively. One test sample of gesture "love" mismatched with any of the 10 templates because of the calculated minimal global cost still higher than the rejection threshold. Also, two test samples of gesture "1" were confused with the gesture "grab" due to the similar finger movement. Simultaneously, two samples of the gesture "ok" were confused with the gesture "no".

5.6.2. Comparisons and Analysis

Using DHMM for dynamic gesture recognition offers two significant advantages. Firstly,

the Discrete Hidden Markov Models (DHMM) approach works well on temporal recognition problems when the training dataset is large enough, which means the accuracy of an HMM $\lambda = (A, B, \pi)$ can be improved by finding the optimal parameters if we feed enough training data. Secondly, DHMM is a powerful statistical framework that provides flexibility to solve many classification issues. Also, as the decision making of HMMs is based only on the sequence information itself, not other metrics such as distance measurements, the classification results are more reliable.

The weaknesses of DHMM in our system are also obvious. One major limitation is that the recognition is highly dependent on the parameters in $\lambda = (A, B, \pi)$, which means that a small difference in the parameters leads to a great discrepancy in the result so that we need a longer training phase to optimize the parameters. Another disadvantage is about the quantization of the N-dimensional vector into a one-dimensional discrete value. The selection of a proper algorithm, such as the K-means algorithm [110], will impact on the classifier's accuracy. Also, as the number of the states is increasing, the computational complexity is getting larger and it will dramatically affect the real-time prediction performance.

The advantages of using the ND-DTW approach for dynamic gesture recognition: (i) it requires comparatively less training data than other mathematical methods because the gesture prediction is done by comparing the input timeseries with a set of trained gesture templates to find the perfect match; (ii) the key of this algorithm is to find the warping path with the minimal global cost which achieved by dynamic programming strategy, which means the complexity of the algorithm is up to $O(M * N)$, where M and N represent the lengths of two timeseries. This feature guarantees that ND-DTW is a quick classifier and suitable for real-time prediction applications.

The weakness of ND-DTW is mainly related to the rejection threshold. Because the calculation of the warping path is based on Euclidean distance measurement, the selection of the rejection threshold has to find the best balance between rejecting the none *G*-gestures inputs as much as possible and not higher than the minimal cost of the real prediction. A low threshold might affect the recognition accuracy.

5.7. Conclusions

This chapter discusses design details of the classification module for the real-time prediction of ten simple dynamic hand gestures in the human-robot interaction mode. The highlight of this chapter has been explained in section 5.4 discussing the two different learning methods, Discrete HMM and N-Dimensional DTW, which could be used for dynamic gestures recognition. The experimental results shown in section 5.5 give the specific recognition rate and confusion matrices for different classifiers. Lastly, the comprehensive comparison of the pros and cons of the two methods shown in section 5.6 is convincingly illustrating why we choose the ND-DTW classifier for our system.

Chapter 6. Conclusions and Future Work

6.1. Conclusions

This thesis presents a novel real-time hand gesture recognition system for training a human-like robot hand to interact with humans or other robots through sign language.

We defined a training/testing dataset of 3D hand gestures, consisting of static postures and dynamic gestures, which is tailored to accommodate the kinematic constructive limitations of the human-like robot hand. The static postures consist of the ten digits (numeric symbols) and a selected set of Latin letters based on the ASL alphabet. The dynamic gestures involve hand rotation movements as well as non-rotation movements.

The raw 3D hand tracking data are converted into high-level and meaningful training features that fed to a real-time hand gesture recognition flowchart.

The classification module uses a multiclass Support Vector Machine (SVM) method for the recognition of the static hand postures, and the N-Dimensional Dynamic Time Warping (ND-DTW) method for the dynamic hand gestures recognition. The static hand posture recognition has two interaction modes: human-robot interaction and robot-robot interaction.

Experiments conducted while using our testing dataset show that the multiclass SVM method offers a good performance for the static posture classification, with an overall 98.34% recognition rate in the HRI mode, and 99.94% in the RRI mode. It also allows for a lower running time comparing with the k-NN and ANBC approaches. Results also show that the ND-DTW method has a better overall performance compared to Discrete-HMM with an average 97% recognition rate and significantly less run time.

6.2. Future Work

Future work should focus on improving the constructive limitations of the human-like robot hand. One potential task would be to build a more flexible robot hand with more degrees of freedom that will allow it to more efficiently perform sign languages and hand gestures. Another task will be to study new interaction systems able to recover hand gestures from different points of view.

References

[1] M. L. Knapp, J. A. Hall, and T. G. Horgan, *Nonverbal Communication in Human Interaction*. 2013.

[2] World Health Organization (WHO), "Deafness and hearing loss," 2017. [Online]. Available: http://www.who.int/mediacentre/factsheets/fs300/en/. [Accessed: 12-Apr-2018].

[3] L. Gallo, A. P. Placitelli, and M. Ciampi, "Controller-free exploration of medical image data: Experiencing the Kinect," *IEEE Symposium on Computer-Based Medical Systems*, 2011.

[4] B. Lange, C. Y. Chang, E. Suma, B. Newman, A. S. Rizzo, and M. Bolas, "Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor," *2011 International Conference of the IEEE Engineering in Medicine and Biology Society*, 2011, vol. 2011, pp. 1831–1834.

[5] Z. Zhang, "Microsoft Kinect Sensor and its Effect," *IEEE Multimedia*, vol. 19, no. 2. pp. 4–10, 2012.

[6] J. Guna, G. Jakus, M. Pogačnik, S. Tomažič, and J. Sodnik, "An analysis of the precision and reliability of the Leap Motion sensor and its suitability for static and dynamic tracking," *Sensors (Switzerland)*, vol. 14, no. 2, pp. 3702–3720, 2014.

[7] A. Kendon, *Gesture: Visible Action as Utterance*, Cambridge University Press. 2005.

[8] Z. Salzmann, R. Sutton-Spence, and B. Woll, "The Linguistics of British Sign Language: An Introduction," *Language*, vol. 76, no. 3, p. 753, 2000.

[9] T. Johnston and A. Schembri, *Australian sign language (Auslan): An introduction to sign language linguistics*. 2007.

[10] S. K. Liddell and R. E. Johnson, "American Sign Language: The Phonological Base," *Sign Language Studies*, vol. 1064, no. 1, pp. 195–277, 1989.

[11] S. E. Ohna, "Open your eyes: deaf studies talking," *Scandinavian Journal of Disability Research*, vol. 12, no. 2, pp. 141–146, 2010.

[12] U. Bellugi and S. Fischer, "A comparison of sign language and spoken language," *Cognition*, vol. 1, no. 2–3, pp. 173–200, 1972.

[13] M. J. Cheok, Z. Omar, and M. H. Jaward, "A review of hand gesture and sign

language recognition techniques," *International Journal of Machine Learning and Cybernetics*, vol. 0, no. 0, p. 0, 2017.

[14] H. Liu and L. Wang, "Gesture recognition for human-robot collaboration: A review," *International Journal of Industrial Ergonomics*, pp. 1–13, 2016.

[15] M. Lades *et al.*, "Distortion invariant object recognition in the dynamic link architecture," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 300–311, 1993.

[16] J. Triesch and C. von der Malsburg, "Robust classification of hand postures against complex backgrounds," in *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, 1996, pp. 170–175.

[17] J. Triesch and C. Von Der Malsburg, "A gesture interface for human-robot-interaction," in *Proceedings - 3rd IEEE International Conference on Automatic Face and Gesture Recognition, FG 1998*, 1998, pp. 546–551.

[18] J. J. . Triesch, C. Von Der Malsburg, and C. von der Malsburg, "A system for person-independent hand posture recognition against complex backgrounds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 12, pp. 1449–1453, 2001.

[19] Y. T. Li and J. P. Wachs, "HEGM: A hierarchical elastic graph matching for hand gesture recognition," *Pattern Recognition*, vol. 47, no. 1, pp. 80–88, 2014.

[20] L. Wiskott, J. M. Fellous, N. Krüger, and C. Von der Malsburg, "Face recognition by elastic bunch graph matching," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1997, vol. 1296, pp. 456–463.

[21] M. Zhao, F. K. H. Quek, and X. Wu, "RIEVL: Recursive induction learning in hand gesture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1174–1185, 1998.

[22] V. Vapnik, "Support vector machine," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[23] H. G. K. Ulrich, "Pairwise classification and support vector machines," in *Advances in kernel methods*, 1999, pp. 255–268.

[24] Y. T. Chen and K. T. Tseng, "Multiple-angle hand gesture recognition by fusing SVM classifiers," *Proceedings of the 3rd IEEE International Conference on*

Automation Science and Engineering, IEEE CASE 2007, pp. 527–530, 2007.

[25] D. Y. Huang, W. C. Hu, and S. H. Chang, "Vision-based hand gesture recognition using PCA+Gabor filters and SVM," *IIH-MSP 2009 - 2009 5th International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 1–4, 2009.

[26] D. Y. Huang, W. C. Hu, and S. H. Chang, "Gabor filter-based hand-pose angle estimation for hand gesture recognition under varying illumination," *Expert Systems with Applications*, vol. 38, no. 5, pp. 6031–6042, 2011.

[27] N. H. Dardas and N. D. Georganas, "Real-time Hand Gesture Detection and Recognition Using Bag-of-Features and Support Vector Machine Techniques," *IEEE Transaction on Instrumentation and Measurement*, vol. 60, no. 11, pp. 3592–3607, 2011.

[28] X. Teng, B. Wu, W. Yu, and C. Liu, "A hand gesture recognition system based on local linear embedding," *Journal of Visual Languages and Computing*, vol. 16, no. 5 SPEC. ISS., pp. 442–454, 2005.

[29] S. S. Ge, Y. Yang, and T. H. Lee, "Hand gesture recognition and tracking based on distributed locally linear embedding," *Image and Vision Computing*, vol. 26, no. 12, pp. 1607–1620, 2008.

[30] T. P. Baum, Leonard E., "Statistical Inference for Probabilistic Functions of Finite State Markov Chains," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[31] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[32] F.-S. Chen, C.-M. Fu, and C.-L. Huang, "Hand gesture recognition using a real-time tracking method and hidden Markov models," *Image and Vision Computing*, vol. 21, no. 8, pp. 745–758, 2003.

[33] H.-K. L. H.-K. Lee and J. H. Kim, "An HMM-based threshold model approach for gesture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 10, pp. 961–973, 1999.

[34] H. S. Yoon, J. Soh, Y. J. Bae, and H. Seung Yang, "Hand gesture recognition using combined features of location, angle and velocity," *Pattern Recognition*, vol. 34, no.

7, pp. 1491–1501, 2001.

[35] M. Elmezain, A. Al-Hamadi, J. Appenrodt, and B. Michaelis, “A Hidden Markov Model-Based Isolated and Meaningful Hand Gesture Recognition,” *Proceedings of the World Academy of Science Engineering Technology*, vol. 43, no. July, p. pages 394-401, 2009.

[36] S. Marcel, O. Bernier, J. E. Viallet, and D. Collobert, “Hand gesture recognition using input-output hidden Markov models,” *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, no. Figure 1, pp. 456–461, 2000.

[37] A. Just and S. Marcel, “A comparative study of two state-of-the-art sequence processing techniques for hand gesture recognition,” *Computer Vision and Image Understanding*, vol. 113, no. 4, pp. 532–543, 2009.

[38] A. Ramamoorthy, N. Vaswani, S. Chaudhury, and S. Banerjee, “Recognition of dynamic hand gestures,” *Pattern Recognition*, vol. 36, no. 9, pp. 2069–2081, 2003.

[39] D. Kim, J. Song, and D. Kim, “Simultaneous gesture segmentation and recognition based on forward spotting accumulative HMMs,” *Pattern Recognition*, vol. 40, no. 11, pp. 3012–3026, 2007.

[40] A. D. Wilson and A. F. Bobick, “Parametric hidden Markov models for gesture recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 884–900, 1999.

[41] M. Brand, N. Oliver, and a. Pentland, “Coupled hidden Markov models for complex action recognition,” *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 994–999, 1997.

[42] C. Vogler and D. ~N. Metaxas, “Parallel Hidden Markov Models for American Sign Language Recognition,” pp. 116–122, 1999.

[43] Q. Chen, N. D. Georganas, and E. M. Petriu, “Hand gesture recognition using Haar-like features and a stochastic context-free grammar,” *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 8, pp. 1562–1571, 2008.

[44] J. Davis and S. Mubarak, “Recognizing hand gestures,” in *European Conference on Computer Vision*, 1994, pp. 331–340.

[45] H. Il Suk, B. K. Sin, and S. W. Lee, “Hand gesture recognition based on dynamic

Bayesian network framework," *Pattern Recognition*, vol. 43, no. 9, pp. 3059–3072, 2010.

[46] M. H. Yang, N. Ahuja, and M. Tabb, "Extraction of 2D motion trajectories and its application to hand gesture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 8, pp. 1061–1074, 2002.

[47] C. W. Ng and S. Ranganath, "Real-time gesture recognition system and application," *Image and Vision Computing*, vol. 20, no. 13–14, pp. 993–1007, 2002.

[48] M. R. Ahsan, M. I. Ibrahimy, and O. O. Khalifa, "Electromygraphy (EMG) signal based hand gesture recognition using artificial neural network (ANN)," in *2011 4th International Conference on Mechatronics (ICOM)*, 2011, pp. 1–6.

[49] K. Tohyama and K. Fukushima, "Neural network model for extracting optic flow," in *Neural Networks*, 2005, vol. 18, no. 5–6, pp. 549–556.

[50] T. Kuremoto, Y. Kinoshita, L. bing Feng, S. Watanabe, K. Kobayashi, and M. Obayashi, "A gesture recognition system with retina-V1 model and one-pass dynamic programming," *Neurocomputing*, vol. 116, pp. 291–300, 2013.

[51] A. Corradini, "Dynamic time warping for off-line recognition of a small gesture vocabulary," in *Proceedings of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, 2001, vol. 2001–Janua, no. January, pp. 82–89.

[52] J. F. Lichtenauer, E. A. Hendriks, and M. J. T. Reinders, "Sign language recognition by combining statistical DTW and independent classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 2040–2046, 2008.

[53] X. Shen, G. Hua, L. Williams, and Y. Wu, "Dynamic hand gesture recognition: An exemplar-based approach from motion divergence fields," *Image and Vision Computing*, vol. 30, no. 3, pp. 227–235, 2012.

[54] K. S. Patwardhan and S. Dutta Roy, "Hand gesture modelling and recognition involving changing shapes and trajectories, using a Predictive EigenTracker," *Pattern Recognition Letters*, vol. 28, no. 3, pp. 329–334, 2007.

[55] M. C. Shin, L. V. Tsap, and D. B. Goldgof, "Gesture recognition using Bezier curves for visualization navigation from registered 3-D data," *Pattern Recognition*, vol. 37, no. 5, pp. 1011–1024, 2004.

[56] Y. Li, “Hand gesture recognition using Kinect,” *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on*, pp. 196–199, 2012.

[57] PrimeSense, “OpenNI User Guide,” *OpenNI User Guide*, vol. 01, no. June, p. 44, 2011.

[58] Y. Yao and Y. Fu, “Contour model-based hand-gesture recognition using the kinect sensor,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 11, pp. 1935–1944, 2014.

[59] Z. Ren, J. Yuan, and Z. Zhang, “Robust hand gesture recognition based on finger-earth mover’s distance with a commodity depth camera,” in *Proceedings of the 19th ACM international conference on Multimedia - MM ’11*, 2011, p. 1093.

[60] Z. Ren, J. Yuan, J. Meng, and Z. Zhang, “Robust part-based hand gesture recognition using kinect sensor,” *IEEE Transactions on Multimedia*, vol. 15, no. 5, pp. 1110–1120, 2013.

[61] Z. Ren, J. Meng, and J. Yuan, “Depth Camera Based Hand Gesture Recognition and its Applications in Human-Computer-Interaction,” *IEEE International Conference on Information Communication and Signal Processing*, vol. 1, pp. 3–7, 2011.

[62] C. Keskin, F. Kiraç, Y. E. Kara, and L. Akarun, “Randomized decision forests for static and dynamic hand shape classification,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012, pp. 31–36.

[63] I. Guyon, V. Athitsos, P. Jangyodsuk, and H. J. Escalante, “The ChaLearn gesture dataset (CGD 2011),” *Machine Vision and Applications*, vol. 25, no. 8, pp. 1929–1951, 2014.

[64] F. Kirac, Y. E. Kara, and L. Akarun, “Hierarchically constrained 3D hand pose estimation using regression forests from single frame depth data,” *Pattern Recognition Letters*, vol. 50, pp. 91–100, 2014.

[65] X. Suau, M. Alcoverro, A. López-Méndez, J. Ruiz-Hidalgo, and J. R. Casas, “Real-time fingertip localization conditioned on hand gesture classification,” *Image and Vision Computing*, vol. 32, no. 8, pp. 522–532, 2014.

[66] C.-H. Chuan, E. Regina, and C. Guardino, “American Sign Language Recognition Using Leap Motion Sensor,” *2014 13th International Conference on Machine*

Learning and Applications, pp. 541–544, 2014.

[67] M. Mohandes, S. Aliyu, and M. Deriche, "Arabic sign language recognition using the leap motion controller," *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, pp. 960–965, 2014.

[68] M. Funasaka, I. Yu, T. Masami, and J. Kazuki, "Sign language recognition using leap motion controller," in *the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2015, p. 263.

[69] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices," *2014 IEEE International Conference on Image Processing, ICIP 2014*, pp. 1565–1569, 2014.

[70] M. Van Den Bergh *et al.*, "Real-time 3D hand gesture interaction with a robot for understanding directions from humans," in *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, pp. 357–362, 2011.

[71] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," *Workshop on Open Source Software of the International Conference on Robotics and Automation (ICRA)*, 2009.

[72] K. Lai, J. Konrad, and P. Ishwar, "A gesture-driven computer interface using Kinect," *2012 IEEE Southwest Symposium on Image Analysis and Interpretation*, pp. 185–188, 2012.

[73] Y. M. Lui, "Human gesture recognition on product manifolds," *Journal of Machine Learning Research*, vol. 13, pp. 3297–3321, 2012.

[74] T. K. Kim and R. Cipolla, "Canonical correlation analysis of video volume tensors for action categorization and detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 8, pp. 1415–1428, 2009.

[75] Z. L. Z. Lin, Z. J. Z. Jiang, and L. S. Davis, "Recognizing actions by shape-motion prototype trees," *12th IEEE International Conference on Computer Vision*, pp. 444–451, 2009.

[76] Y. M. Lui, "A least squares regression framework on manifolds and its application to gesture recognition," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 13–18, 2012.

[77] U. Mahbub, H. Imtiaz, T. Roy, M. S. Rahman, and M. A. R. Ahad, "A template

matching approach of one-shot-learning gesture recognition," *Pattern Recognition Letters*, vol. 34, no. 15, pp. 1780–1788, 2013.

[78] J. Wan, Q. Ruan, W. Li, and S. Deng, "One-shot learning gesture recognition from RGB-D data using bag of features," *The Journal of Machine Learning …*, vol. 14, pp. 2549–2582, 2013.

[79] P. K. Pisharady and M. Saerbeck, "Robust gesture detection and recognition using dynamic time warping and multi-class probability estimates," in *Proceedings of the 2013 IEEE Symposium on Computational Intelligence for Multimedia, Signal and Vision Processing, CIMSIVP 2013 - 2013 IEEE Symposium Series on Computational Intelligence, SSCI 2013*, 2013, pp. 30–36.

[80] H. Cheng, Z. Dai, Z. Liu, and Y. Zhao, "An image-to-class dynamic time warping approach for both 3D static and trajectory hand gesture recognition," *Pattern Recognition*, 2015.

[81] H. Cheng, J. Luo, and X. W. Chen, "A Windowed Dynamic Time Warping Approach for 3d Continuous Hand Gesture Recognition," *2014 IEEE International Conference on Multimedia and Expo (ICMEs)*, 2014.

[82] Y. Chen, Z. Ding, Y. L. Chen, and X. Wu, "Rapid recognition of dynamic hand gestures using leap motion," *2015 IEEE International Conference on Information and Automation, ICIA 2015 - In conjunction with 2015 IEEE International Conference on Automation and Logistics*, pp. 1419–1424, 2015.

[83] W. Lu, Z. Tong, and J. Chu, "Dynamic hand gesture recognition with leap motion controller," *IEEE Signal Processing Letters*, vol. 23, no. 9, pp. 1188–1192, 2016.

[84] M. B. Holte, T. B. Moeslund, and P. Fihl, "Fusion of range and intensity information for view invariant gesture recognition," *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, 2008.

[85] F. Erden and A. Çetin, "Hand gesture based remote control system using infrared sensors and a camera," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 675–680, 2014.

[86] O. Biomics, "ADA V 1.1 Assembly Instructions," 2018. [Online]. Available: https://www.openbionics.com/obtutorials/ada-v1-assembly. [Accessed: 30-Jan-2018].

[87] InMoov, “Hand and Forarm assembly 3D views.” [Online]. Available: http://inmoov.fr/build-yours/hand-and-forarm-assembly-3d-views/. [Accessed: 26-Feb-2018].

[88] I. HITEC RCD USA, “HS-422 Deluxe Standard Servo - Product Description,” 2018. [Online]. Available: http://hitecrcd.com/products/servos/sport-servos/analog-sport-servos/hs-422-deluxe-standard-servo/product. [Accessed: 27-Feb-2018].

[89] D. Bassily, C. Georgoulas, J. Güttler, T. Linner, T. Bock, and T. U. München, “Intuitive and adaptive robotic arm manipulation using the leap motion controller,” *ISR/Robotik 2014*, pp. 78–84, 2014.

[90] Y. I. Curiel-Razo, O. Icasio-Hernández, G. Sepúlveda-Cervantes, J. B. Hurtado-Ramos, and J. J. González-Barbosa, “Leap motion controller three dimensional verification and polynomial correction,” *Measurement: Journal of the International Measurement Confederation*, vol. 93, pp. 258–264, 2016.

[91] Leap Motion, “C++ SDK Documentation,” 2018. [Online]. Available: https://developer.leapmotion.com/documentation/cpp/index.html.

[92] N. Gillian and J. a Paradiso, “The Gesture Recognition Toolkit,” *Journal of Machine Learning Research*, vol. 15, pp. 3483–3487, 2014.

[93] Leap Motion, “API Overview.” [Online]. Available: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Overview.html. [Accessed: 27-Jan-2018].

[94] J. Lin, Y. Wu, and T. S. Huang, “Modeling the constraints of human hand motion,” in *Proceedings - Workshop on Human Motion, HUMO 2000*, 2000, pp. 121–126.

[95] N. Gillian, R. B. Knapp, and E. Al, “An adaptive classification algorithm for semiotic musical gestures,” *the 8th International Conference on Sound and Music Computing (SCM2011)*, 2011.

[96] S. H. Lu, D. A. Chiang, H. C. Keh, and H. H. Huang, “Chinese text classification by the Naïve Bayes Classifier and the associative classifier with multiple confidence threshold values,” *Knowledge-Based Systems*, vol. 23, no. 6, pp. 598–604, 2010.

[97] P. Ziaie, T. Müller, M. E. Foster, and A. Knoll, “A naïve Bayes classifier with distance weighting for hand-gesture recognition,” in *Communications in Computer and Information Science*, 2008, vol. 6 CCIS, pp. 308–315.

[98] R. W. Hamming, “Error Detecting and Error Correcting Codes,” *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[99] E. F. Krause, “Taxicab Geometry,” *National Council of Teachers of Mathematics*, vol. 66, no. 8, pp. 695–706, 1973.

[100] J. B. Kruskal, “Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis,” *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[101] L. E. Peterson, “K-nearest neighbor,” *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[102] C. Chang and C. Lin, “LIBSVM : A Library for Support Vector Machines,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, pp. 1–39, 2011.

[103] J. Manikandan and B. Venkataramani, “Design of a modified one-against-all SVM classifier,” in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 1869–1874.

[104] J. Weston and C. Watkins, “Support Vector Machines for Multi-Class Pattern Recognition,” *Proceedings of the 7th European Symposium on Artificial Neural Networks (ESANN-99)*, no. April, pp. 219–224, 1999.

[105] M. Köppen, “The curse of dimensionality,” *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*, vol. 1, pp. 4–8, 2000.

[106] Z. Yang, Y. Li, W. Chen, and Y. Zheng, “Dynamic hand gesture recognition using hidden Markov models,” *2012 7th International Conference on Computer Science & Education (ICCSE)*, no. Iccse, pp. 360–365, 2012.

[107] N. Gillian, R. B. Knapp, and S. O ’modhrain, “Recognition Of Multivariate Temporal Musical Gestures Using N-Dimensional Dynamic Time Warping,” *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME'11)*, pp. 337–342, 2011.

[108] H. Sakoe and S. Chiba, “Dynamic Programming Algorithm Optimization for Spoken Word Recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[109] G. A. Ten Holt, M. J. T. Reinders, and E. A. Hendricks, “Multi-Dimensional Dynamic Time Warping for Gesture Recognition,” *Thirteenth annual conference of the Advanced School for Computing and Imaging*, 2007.

[110] D. Lee, S. Back, and K. Sung, “Modified K-means algorithm for vector quantizer

design," *IEEE Signal Processing Letters*, vol. 4, no. 1, pp. 2–4, 1997.