Technische Universität Berlin

Gesture Recognition for Human-Robot Interaction: An approach based on skeletal points tracking using depth camera

Masterarbeit
am Fachgebiet Agententechnologien in betrieblichen Anwendungen und der Telekommunikation (AOT)
Prof. Dr.-Ing. habil. Sahin Albayrak
Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

vorgelegt von
Sivalingam Panchadcharam Aravinth

Betreuer: Prof. Dr.-Ing. habil. Sahin Albayrak,
Dr. Yuan Xu

Sivalingam Panchadcharam Aravinth
Matrikelnummer: 342899
Sparrstr. 9
13353 Berlin

Statement of Authorship

Abstract

Human-robot interaction (HRI) has been a topic of both science fiction and academic speculation even before any robots existed. HRI research is focusing to build an intuitive and easy communication with the robot through speech, gestures and facial expressions. The use of hand gestures provides a better solution than conventional human-machine interfaces. Furthermore, translations of hand gestures can help in accomplishing the ease and naturalness desired for HRI. This has motivated a very active research concerned with computer vision-based analysis and interpretation of hand gestures.

In this thesis, we aim to implement the hand gesture recognition for robots with modeling, training, classifying and recognizing gestures based on computer vision algorithms and machine learning techniques. Gestures are modeled based on skeletal points and the features are extracted using NiTE framework using a depth camera.

In order to recognize gestures, we propose to learn and classify hand gestures with the help of Adaptive Naive Bayes Classifier using Gesture Recognition Toolkit. Furthermore, we aim to build a dashboard that can visualize the interaction between all essential parts of the system. Finally, we attempt to integrate all these functionalities into a system that interacts with a humanoid robot NAO.

As a result, on one hand, gestures will be used command the robot to execute certain actions and on the other hand, gestures will be translated and spoken out by the robot.

Keywords

Human-Robot Interaction, HRI, Hand Gesture Recognition, Humanoid Robot, NAO, Skeletal Points Tracking, NiTE, Depth Camera, Asus Xtion Pro Live, Machine Learning, Adaptive Naive Bayes Classifier, ANBC, Gesture Recognition Toolkit, GRT

Acknowledgments

It is a pleasure to thank all those who made this thesis possible. First of all, I would like to deeply thank my thesis advisor Dr. Yuan Xu for his continued support throughout the months that i worked on this Master Thesis at DAI Labor.

The biggest thanks have to go to my parents and my wife Natalia. Without their support, encouragement and patience, this thesis would not have been completed. Furthermore, I would like to thank my friend Mladen Miljic at Yetu AG for the hardware support for this work.

I am extremely grateful for the members of the examining committee, especially Prof. Sahin Albayrak, Dr. Stefan Fricke, Martin Berger for the constructive comments on this manuscript. Additionally, i would like to thank my friends at DAI Labor who have been helping me in various tasks during my thesis.

Finally, i would like to thank my employer Alessio Avellan Borgmeyer of The Jodel Venture GmbH who managed to support my autonomy, and provided me enough financial support and time off from my official working hours, so that i could concentrate on my thesis.

Contents

Chapter 1

Introduction

Huge influence of computers in society has made smart devices an important part of our lives. Availability, affordability and functionality of such devices motivated us to use them in our day-to-day living. The list of smart devices includes personal automatic and semi-automatic robots that are also playing a major role in our household. For an instance, Roomba is an autonomous robotic vacuum cleaners that automatically cleans the floor and goes to its charging station without human interaction [**?**].

Interaction with smart devices is still being mostly through displays, keyboards, mouse and touch interfaces. These devices have grown to be familiar but inherently limit the speed and naturalness with which we can interact with the machine. Usage of robots for domestic and industrial purposes has been continuously increasing [**?**]. Thus in recent years, there has been a tremendous push in the research towards an intuitive and easy communication with the robot through speech, gestures and facial expressions.

Tremendous progress have been made in speech recognition and several commercially successful speech interfaces are available [**?**]. However, speech recognition systems have certain limitations such as misinterpretation due to various accents and background noise interference.

Furthermore, there has been an increased interest in recent years in trying to introduce other human-to-human communication modalities into HRI. This includes a class of techniques based on the movement of the human arm or hand. The use of hand gestures provides an attractive alternative for HRI than the conventional cumbersome devices.

1.1 Goal

As described earlier, HRI research is focusing to build an intuitive and easy communication with the robot through speech, gestures and facial expressions. The use of hand gestures provides the ease and naturalness with which the user can interact with robots [?]. Our goal in this thesis to implement a system that should be integrated into NAO to recognize hand gestures.

Existing cameras of NAO are greatly limited by the quality of the input image. Variations in lighting and background clutters would only worsen the problem [?]. On the other hand, depth-based approaches are able to provide satisfactory results for hand gesture recognition even with poor indoor lighting and cluttered background condition [?]. Therefore, we have chosen Asus Xtion which has sensors that capture both RGB and depth data. Asus Xtion is an OpenNI compatible device, thus, we have chosen a NiTE middleware for the purpose of tracking the human skeletal points.

We have chosen Gesture Recognition Toolkit [?] to train and predict the 3D skeletal modeled gestures with feature based statistical learning algorithm. Adaptive Naive Bayes Classifier is the supervised machine learning algorithm which is chosen for the purpose of classifying and predicting the hand gestures in real time. Furthermore, all these interactions must be displayed to visually understand the status of the system. Finally, recognized hand gestures must be translated to robotic actions as following :

- **Gesture-to-Speech** should translate the recognized gestures and it should be spoken out loud using the integrated loudspeaker.
- **Gesture-to-Motion** should move the robot from one position to another in the 2 dimensional space. Therefore, each gesture should be assigned to a locomotion task.
- **Gesture-to-Gesture** should translate the human hand gesture to a robotic hand gesture by imitating hand gestures of the user.

The goal should be reached by studying the various solution to this problem and an appropriate design must be chosen. The main challenge is to find a solution that can integrate all these components into a robust system. Furthermore, this system must be tested and results must be presented clearly. Evaluations must be carried out to demonstrate the effectiveness of the classifier and to validate its potential for real time gesture recognition.

1.2 Outline

Background To begin with, we have studied key concepts, state of the art solutions and tools available to accomplish this goal. Chapter 2 thoroughly discusses about the humanoid robot and stages of hand gesture recognition with the help of computer vision and machine learning algorithms. In details, this chapter explains the concepts of gesture modeling, feature extraction from depth image by NiTE framework, learning and classification of hand gestures using Adaptive Naive Bayes Classifier with the help of GRT.

Hand Gesture Recognition for Human-Robot Interaction In addition to that, Chapter 3 talks about the functional blocks of system and how they are implemented to provide an efficient solution. Furthermore, it illustrates the functionalities of Human-Robot Interaction (HRI) module, Brain module, Control Center (CC) module and Command module, and how they all are integrated into a robust system.

Results and Evaluation Finally, Chapter 4 shows promising results in recognizing five static hand gestures which are trained in lab condition. It shows not only the performances of gesture prediction, but also the human-robot interactions where the robot has executed predefined tasks, when the hand gestures are recognized. Evaluation demonstrates the effectiveness of the system and its potential for real time recognition application. Finally, metrics such as mean, standard deviation, confusion matrix, precision, recall, F-Measure are presented.

Conclusion In a conclusion, this manuscript demonstrates how the goal of building a hand gesture recognition based on skeletal points tracking using depth camera is accomplished, as well as, how this system can be improved in several ways by proposing various alternatives.

Chapter 2

Background

Areas of artificial intelligence deal with autonomous planning or deliberation for robotic systems to navigate through an environment. A detailed understanding of these environments is required to navigate through them. High-level information about the environment could be provided by a computer vision system that is acting as a vision sensor.

In this thesis, we will focus on building hand gesture recognition system for a humanoid robot name as NAO, with the help of computer vision algorithms to track the skeletal points of human body using depth camera. Following sections clearly studies the key concepts of this thesis.

2.1 NAO - The Humanoid Robot

NAO is an autonomous programmable humanoid robot invented by Aldebaran Robotics. NAO Academics Edition is developed for universities and laboratories for research and educational purposes. Follow subsections briefly discuss the specifications of NAO as described by Aldebaran Robotics [**?**].

2.1.1 Body

NAO has a body with 25 degrees of freedom (DOF) whose key elements are electric motors and actuators as show in the figure 2.1. It has 48.6-watt-hour battery that provides 1.5 or more hours of autonomy, depending on the usage. Additional specifications of robot are shown in the table 2.1.

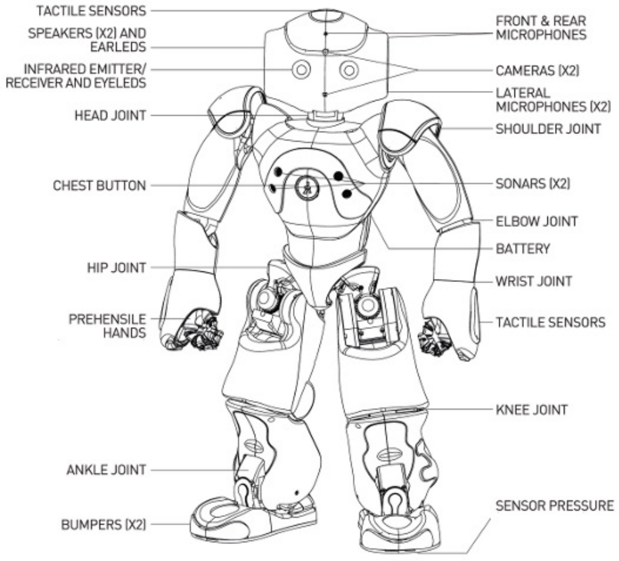


Figure 2.1: The body construction of NAO V5. [?]

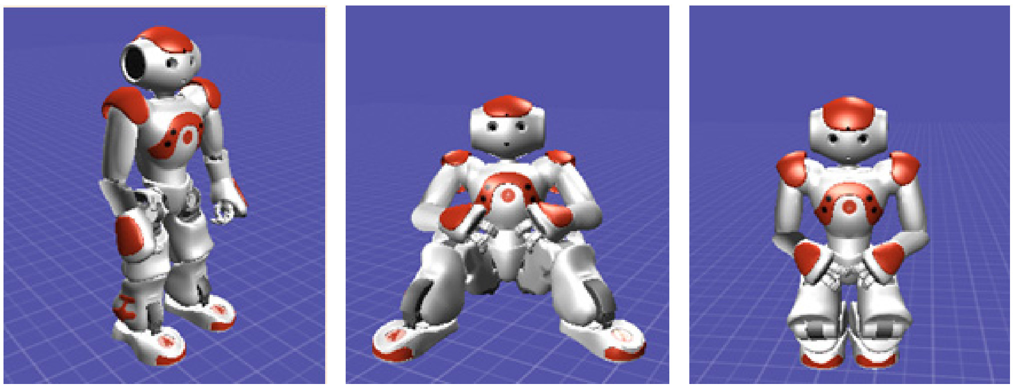


Figure 2.2: Standing, Sitting and Crouching postures of virtual NAO using ALRobot-Posture module. [?]

2.1.2 Motion

NAOs motion module is based on generalized inverse kinematics, which handles locomotion, joint control, balance, redundancy, and task priority. This means that when asking it to extend its arm, it bends over because its arms and leg joints are taken into account.

In this thesis, we attempt to use the locomotion and stiffness control of Motion API to move NAO to a position in the two dimensional space. Robot Posture API will also be used to make the robot go to the predefined posture such as Stand, Sit and Crouch as shown in the figure 2.2.

Table 2.1: NAO V5 specification. [?]

Height	58 centimeters (23 in)
Weight	4.3 kilograms (9.5 lb)
Battery autonomy	60 minutes (active use), 90 minutes (normal use)
Degrees of freedom	25
CPU	Intel Atom @ 1.6 GHz
Built-in OS	Linux
SDK compatibility	Windows, Mac OS, Linux
Programming languages	C++, Python, Java, MATLAB, Urbi, C, .Net
Vision	2 x HD 1280x960 cameras
Connectivity	Ethernet, Wi-Fi
Sensors	4 x directional microphones 1 x sonar range finder 2 x IR emitters and receivers 1 x inertial board 9 x tactile sensors 8 x pressure sensors

2.1.3 Audio

NAO uses four directional microphones to detect sounds and equipped with a stereo broadcast system made up of 2 loudspeakers in its ears as shown in the figure 2.3. NAOs voice recognition and text-to-speech capabilities allow it to communicate in 19 languages.

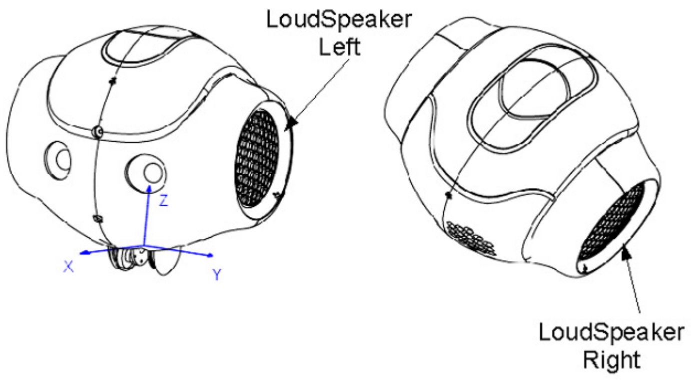


Figure 2.3: Stereo broadcast system of NAO using 2 loudspeakers. [?]

In this thesis, we aim to use Text-To-Speech API of NAO to say the detected gesture out loud.

2.1.4 Depth Sensor

Skeletal points based gesture recognition needs three dimensional data of the human skeleton. However, sensors integrated with NAO could not provide precise three dimensional data to the sophisticated algorithms to track human skeletal joints [?]. 3D cameras such as Microsoft Kinect and Asus Xtion are used not only for gaming but also for analyzing 3D data, including algorithms for feature selection, scene analysis, motion tracking, skeletal tracking and gesture recognition [?] [?]. Therefore, we seek to utilize Asus Xtion PRO LIVE as an external camera to support the skeletal points tracking system of NAO.

Figure 2.4: OpenNI compatible Asus Xtion PRO LIVE is a commercial depth camera that can capture both RGB and depth image. [?]

Table 2.2: Asus Xtion PRO LIVE specification. [?]

Distance of Use	Between 0.8m and 3.5m
Field of View	58 deg Horizontal, 45 deg Vertical, 70 deg Diagonal
Frame rate	VGA (640x480) : 30 fps QVGA (320x240): 60 fps
Resolution	SXGA (1280x1024)
Dimensions	18 x 3.5 x 5 cm
Sensors	1 x RGB Video camera 2 x IR Depth sensors 2 x Microphones

Asus Xtion Figure 2.4 shows Asus Xtion PRO LIVE that uses infrared sensors, adaptive depth detection technology, color image sensing and audio stream to capture a 3D image of the user in real-time. It uses infrared emitters to project speckle patterns on the object and uses a structured light technique to compute the depth of the image. Once the depth image is computed, it is mapped onto the RGB image as shown in the figure 2.5. Lighter color denote that a pixel is closer to the camera and darker color denotes that a pixel is far from the camera. Table 2.2 indicates the specification of Asus Xtion PRO LIVE.

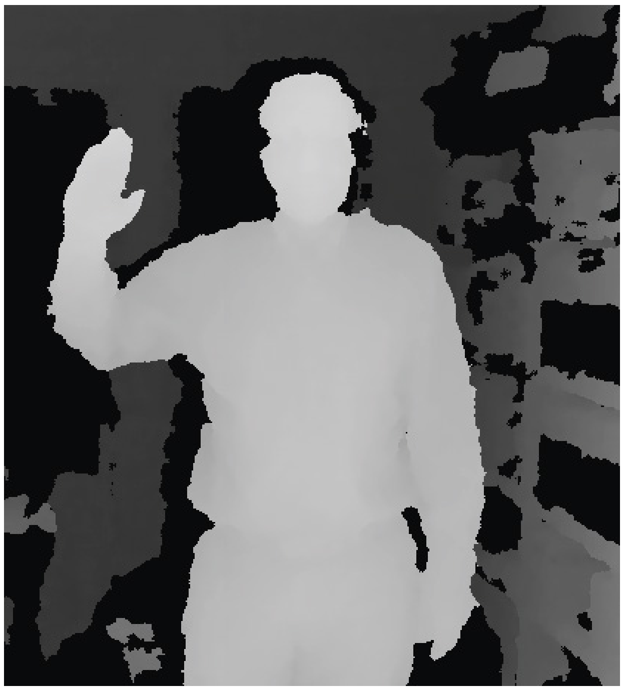


Figure 2.5: Depth Image captured by Asus Xtion PRO LIVE using OpenNI. The darker the color of a pixel, the farther it is from the sensor. [?]

2.1.5 Computing

NAO is equipped with Intel ATOM 1.6 GHz CPU in the head that runs a 32 bit Gentoo Linux to support Aldebarans proprietary middleware (NAOqi). NAOqi SDK is the programming framework used to program Aldebaran robots [?]. This framework allows homogeneous communication between different modules such as motion, audio and video. NAOqi executable that runs on the robot is a broker. The broker provides lookup services so that any module in the tree or across the network can find any method that has been advertised as shown in the figure 2.6.

Computational limitations [?] of NAO CPU hinders us to build a real time gesture recognition based on human skeletal joints. Thus, we aim to use an off-board computer to execute the gesture recognition program and communicated with NAO via NAOqi proxies.

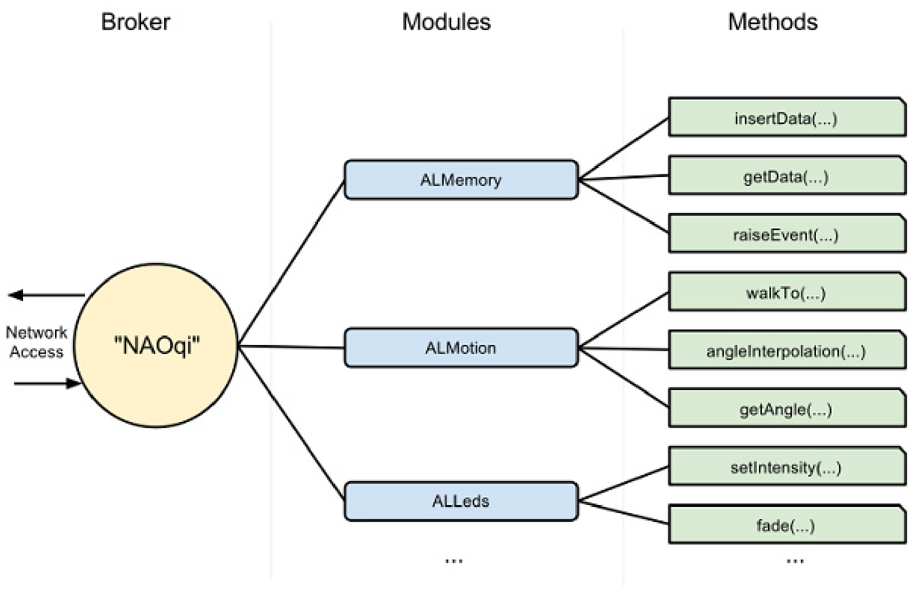


Figure 2.6: NAOqi modules form a tree of methods attached to modules, and modules attached to a broker. Thus, NAOqi Broker (proxy) can be used to remotely invoke any attached methods. [?]

2.2 Hand Gesture Recognition

Human hand gestures are means of nonverbal interaction among people. They range from simple actions of using our hand to point at, to the more complex ones that express our feelings and allow us to communicate with others. To exploit the use of gestures in HRI, it is necessary to provide the means by which they can be interpreted by robots. The HRI interpretation of gestures requires that dynamic and/or static configurations of the human hand, arm and even other parts of the human body, be measurable by the machine [?].

Initial attempts to recognize hand gestures resulted in electro-mechanical devices that directly measure hand and/or arm joint angles and spatial position using sensors. Glove-based gestural interfaces require the user to wear such a complex device that hinders the ease and naturalness with which the user can interact with the computer controlled environment [?].

Even though such hand gloves are used in highly specialized domain such as simulation of medical surgery or even in the real surgery, the everyday user will be certainly deterred by such sophisticated interfacing devices. As an active result of the motivated research in HRI, computer vision based techniques are innovated to augment the naturalness of interaction.

2.2.1 Gesture Modeling

Figure 2.7 shows various types of modeling techniques used to model a gesture [?]. Selection of an appropriate gesture modeling depends primarily on the intended application. For an application that needs just hand gesture to go up and down or left and light, a very simple model may be sufficient. However, if the purpose is a natural-like interaction, a model has to be sophisticated enough to interpret all the possible gesture. The following section discusses various gesture modeling techniques which are being used by the existing hand gesture recognition applications.

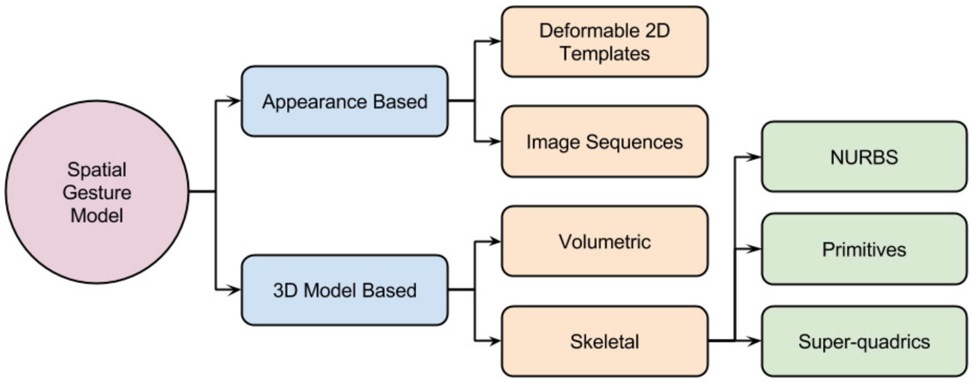


Figure 2.7: Types of Spatial Gesture Modeling. [?]

Appearance based models do not use the spatial representation of the body, because they derive the parameters directly from the images or videos using a template database [?]. Volumetric approaches have been heavily used in computer animation industry and for computer vision purposes. The models are generally created of complicated 3D surfaces. The drawback of this method is that is very computational intensive.

Instead of using intensive processing of 3D hand models and dealing with a lot of parameters, one can just use a simplified version that analyses the joint angle parameters along with segment length. This is known as a skeletal representation of the body, where a virtual skeleton of the person is computed and parts of the body are mapped to certain segments [?]. The analysis here is done using the position and orientation of these segments or the relation between each one of them.

In this thesis, we focus on skeletal based modeling, that is faster because the classifier has to focus only on the significant parts of the body.

2.2.2 Gestural Taxonomy

Several taxonomies have been suggested that deal with psychological aspects of gestures [?]. All hand/arm movements are first classified into two major classes Gestures and Unintentional movements as shown in the figure 2.8.

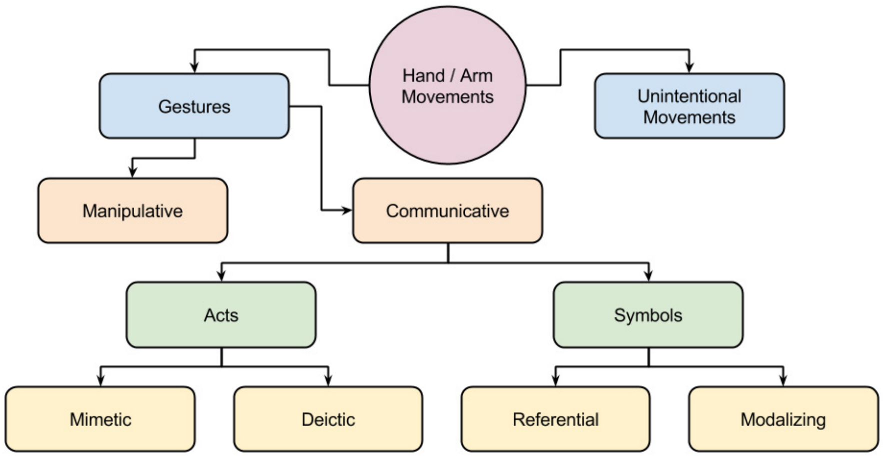


Figure 2.8: The taxonomy of Hand Gestures. [?]

Manipulative gestures are the ones used to act on objects. For example, moving a chair from one location to another. Manipulative gestures in the context of HRI are mainly developed for medical surgery. Communicative gestures, on the other hand, have purely communicational purpose. In a natural environment they are usually accompanied by speech or spoken as a sign language. In HRI context these gesture are one of the commonly used gestures, since they can often be represented by static as well as dynamic hand postures.

In this thesis, we focus on communicative gestures in the form of symbols. They symbolize some referential action. For instance, circular motion of hand may be referred as a command to turn in a circular motion.

2.2.3 Feature Extraction

Feature extraction stage is concerned with the detection of features which will be used for the estimation of parameters of the chosen gestural model. In the detection process it is first necessary to localize the user. Following sections talk about algorithms which are use to locate and extract features from the depth image.

2.2.3.1 OpenNI 2

OpenNI or Open Natural Interaction [?] is a framework by the company PrimeSense and open source software project focused on improving interoperability of natural user interfaces for Natural Interaction (NI) devices, applications which use those devices and middleware that facilitates access and use of such devices. Microsoft Kinect and Asus Xtion are commercially available depth cameras which are compatible with OpenNI.

The OpenNI 2.0 API provides access to PrimeSense compatible depth sensors. It allows an application to initialize a sensor and receive depth, RGB and video streams from the device. OpenNI also provides a uniform interface that third party middleware developers can use to interact with depth sensors. Applications are then able to make use of both the third party middleware, as well as underlying basic depth and video data provided directly by OpenNI.

2.2.3.2 NiTE 2

PrimeSense Natural Interaction Technology for End-user (NiTE) [?] is the middleware that perceives the world in 3D, based on the PrimeSensor depth images, and translates these perceptions into meaningful data in the same way as people do. NiTE middleware includes computer vision algorithms that enable identifying users and tracking their movements. Figure shows the architecture of NiTE, how it is working together with OpenNI, depth sensors and applications.

Figure 2.9 displays a layered view of producing, acquiring and processing depth data, up to the level of the application that utilizes it to form a natural- interaction based module.

- The lower layer is the PrimeSensor device, that is the physical acquisition layer, resulting in raw sensory data from a stream of depth images.
- The next C shaped layer is executed on the host PC represents OpenNI. OpenNI provides communication interfaces that interact with both the sensors driver and the middleware components, which analyze the data from the sensor.
- The sensor data acquisition is a simple acquisition API, enabling the host to operate the sensor. This module is OpenNI compliant interfaces that conforms to OpenNI API standard.
- The NiTE Algorithms layer is the computer vision middleware and is also plugged into OpenNI. It processes the depth images produced by the PrimeSensor.

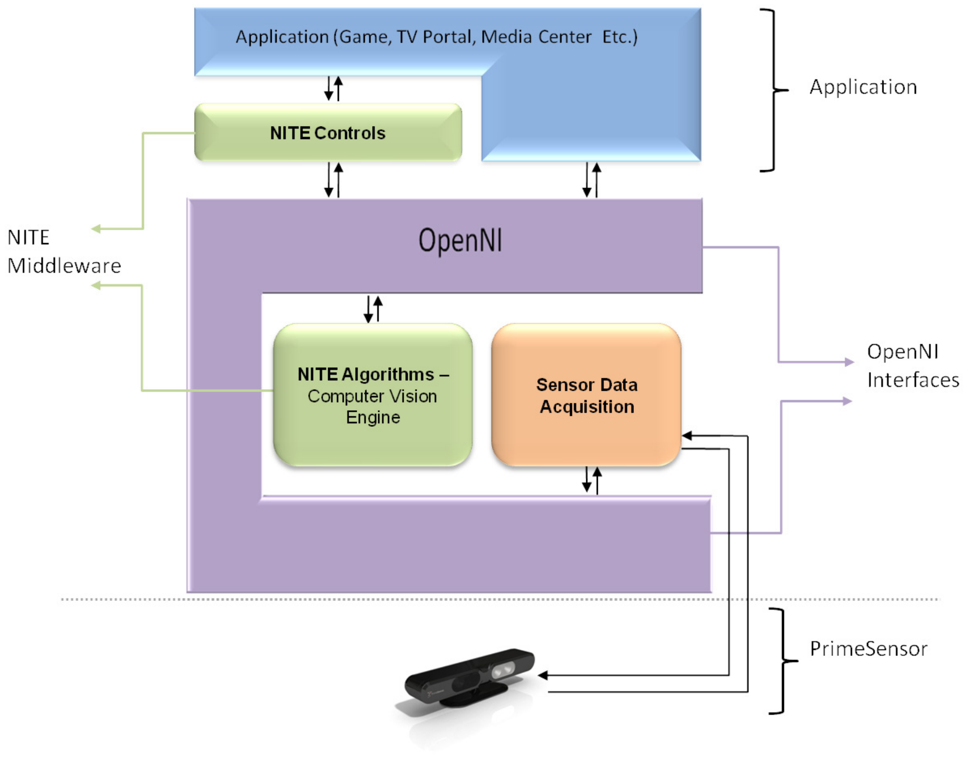


Figure 2.9: PrimeSense architecture shows that OpenNI processes depth data from the depth camera in thee lower layer, computer vision algorithm NiTE as middleware and in the higher layer end-user applications. [**?**]

- The NiTE Controls layer is an application layer that provides application framework for gesture identification and gesture-based UI controls, on top of the data that is processed by NiTE Algorithms.

2.2.3.3 Skeletal Points Tracking Algorithm

The lower layer of NiTE middleware that performs the groundwork of processing the stream of raw depth images. This layer utilizes computer vision algorithms to perform the following:

- Scene segmentation is a process in which individual users and objects are separated from the background and tagged accordingly.
- Hand point detection and tracking.
- Full body tracking based on the scene segmentation output. Users bodies are tracked to output the current user pose with a set of locations of body joints.

NiTE uses machine learning algorithms to recognize anatomical landmarks and pose of human body. Figure 2.10 shows how skeleton tracking algorithm works from a single input depth image and a per-pixel body part distribution is derived [**?**]. Colors indicate the most likely part labels at each pixel and correspond to the joint proposals. Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

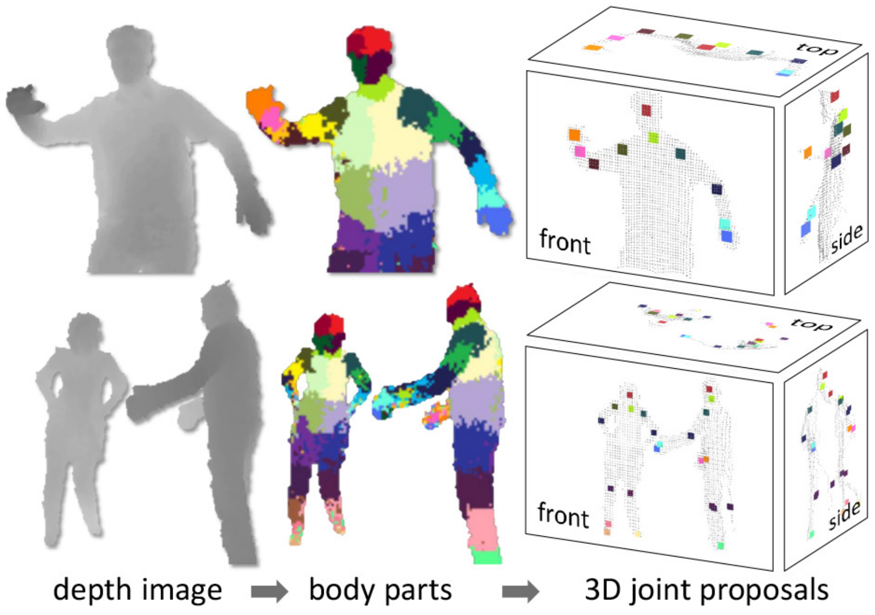


Figure 2.10: Skeleton Tracking algorithm processes a depth image and a per-pixel body part distribution is inferred, and finally, 3D joints proposals are made for 15 points in human skeleton. [**?**]

Training In order to train the system, large collection of synthetic and real representations of human body are recorded and labeled. Each body representation is covered with several localized body part labels. Some of these parts are defined to directly localize particular skeletal joints of interest, while others fill the gaps or could be used in combination to predict other joints.

Feature Labeling Features are located in depth image and labeled as shown in the figure 2.11. Feature extraction uses simple depth comparison between pixels in the depth image. For example, a feature θ_1 is to find the top of the body by comparing the depth difference of an offset pixel that is located above and feature θ_2 is to find thinner vertical body part such as arm. The yellow crosses indicate the pixel is being classified. The red circles indicate the offset pixel with which the depth comparison will be computed.

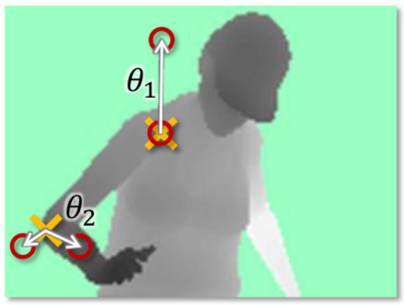


Figure 2.11: Features are located in depth image and labeled. [**?**]

Classification Randomized decision forest is the classification algorithm to predict the probability of a pixel belonging to a body part. Randomized decision trees and forests have been proven as fast and effective multi-class classifiers for many tasks [**?**]. Figure 2.12 shows the branching trees of Randomized Decision Forests algorithm and the red arrows indicate the different paths that might be taken by different trees for a particular input. A forest is an ensemble T decision trees. Each tree consists of split nodes (blue) and leaf nodes (green). Each split node consists of a feature f_θ and a threshold τ.

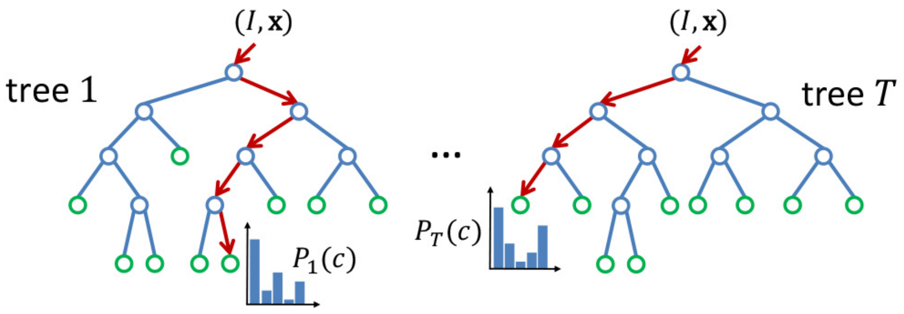


Figure 2.12: Randomized decision forest algorithm showing the branching trees with blue as split node and green as leaf node. [**?**]

Prediction To classify a pixel x in image I using Randomized decision tree, one starts at the root and repeatedly evaluates equation 2.1, branching left or right according to the comparison of threshold τ. When the lead node in the decision tree t is reached, a learned distribution $P_t(c|I,x)$ over body part labels c is stored. The distributions are averaged together for all trees in the forest to give the final classification.

$$P_t(c|I,x) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|I,x) \tag{2.1}$$

Each tree is trained on a different set of randomly synthesized images. A random subset of 2000 example pixels from each image is chosen to ensure an even distribution across body parts. Training phase is conducted in distributed manner by training 3 trees from 1 million images on 1000 core clusters [**?**].

After predicting the probability of a pixel belonging to a body part, the body parts are recognized and reliable proposals for the positions of 3D skeletal joints are generated. These proposals are the final output of the algorithm and used by a tracking algorithm to self initialize and recover from failure.

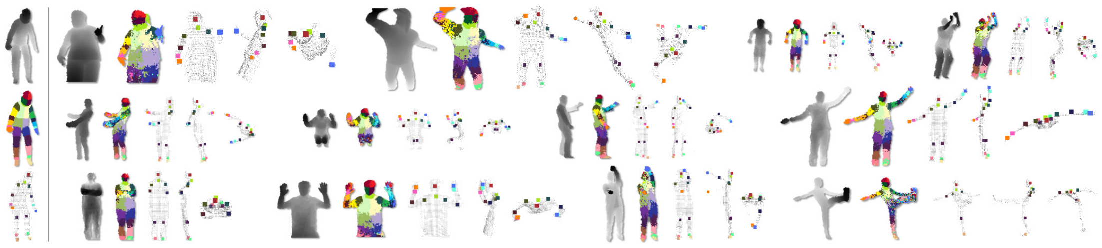


Figure 2.13: Joint proposals are derived for various poses. Results of synthetic training data is shown on the top row, real training data shown in the middle row and failure modes at bottom. Left column shows a neutral pose as a reference. [**?**]

Joints Proposal Figure 2.13 shows example results of synthetic and real datasets. In each example we see the depth image, the derived label of most likely body part, and the front, right, and top views of the joint proposals overlaid on a depth point cloud.

Skeletal points Finally, the API returns positions and orientations of the skeleton joints as shown in the figure 2.14. As well as, it returns the lengths of the body segments such as the distance between elbow and shoulder. Joint positions and orientations are given in the real world coordinate system. The origin of the system is at the sensor. +X points to the right, +Y points upward, and +Z points in the direction of increasing depth.

Hand Tracker Even though NiTE framework can recognize full human body, in this thesis we attempt to use only hand recognition and tracking due to the computational limitation of NAO. To start tracking a hand, a focus gesture must be gesticulated. There are two supported focus gestures: CLICK and WAVE. In the CLICK gesture, the user

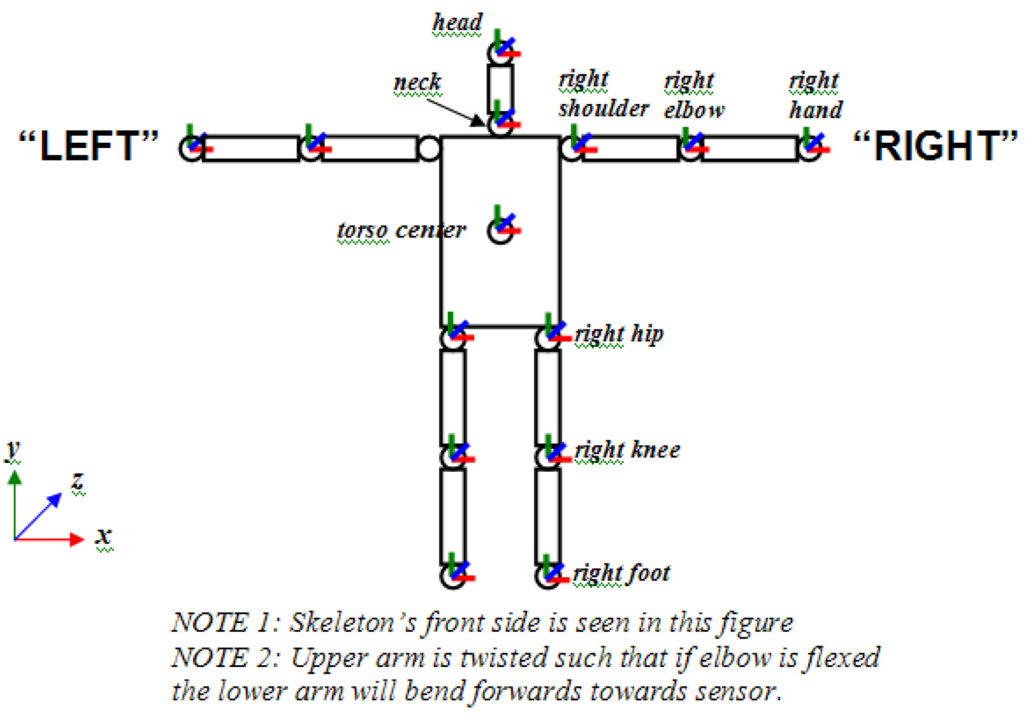

head
neck
right shoulder
right elbow
right hand
"LEFT"
"RIGHT"
torso center
right hip
right knee
right foot
y
z
x
NOTE 1: Skeleton's front side is seen in this figure
NOTE 2: Upper arm is twisted such that if elbow is flexed the lower arm will bend forwards towards sensor.

Figure 2.14: Positions and orientations of the tracked skeleton return by NiTE API. [**?**]

should hold the hand up and push it towards the sensor, then immediately pull the hand backwards. In the WAVE gesture, the user should hold the hand up and move it several times from left to right and back. Once hand is been found and it will be tracked till the hand leaves the field of view of the camera or hand point is lost due to various factors such as hand is touching another object or closer to another body part. Figure 2.15 shows how hand points are tracked using NiTE and trail of the hand positions in real world coordinates are mapped on to the depth image.

Focus gestures Focus gestures of NiTE can be detected even during hand tracking session. NiTE gestures are derived from a stream of hand points thats records how a hand moves through space over time. Each hand point is at the center of the hand in real-world 3D coordinate measured in millimeters. Gesture detectors are sometimes called point listeners (or point controls) since they analyze the points stream looking for a gesture. NiTE recommends users to follow these suggestions to gain maximum efficiency from its API for Hand tracking.

- The hand that performs the gesture must be kept away from the body of the user.
- Palm should be open, fingers pointing up and face the sensor.

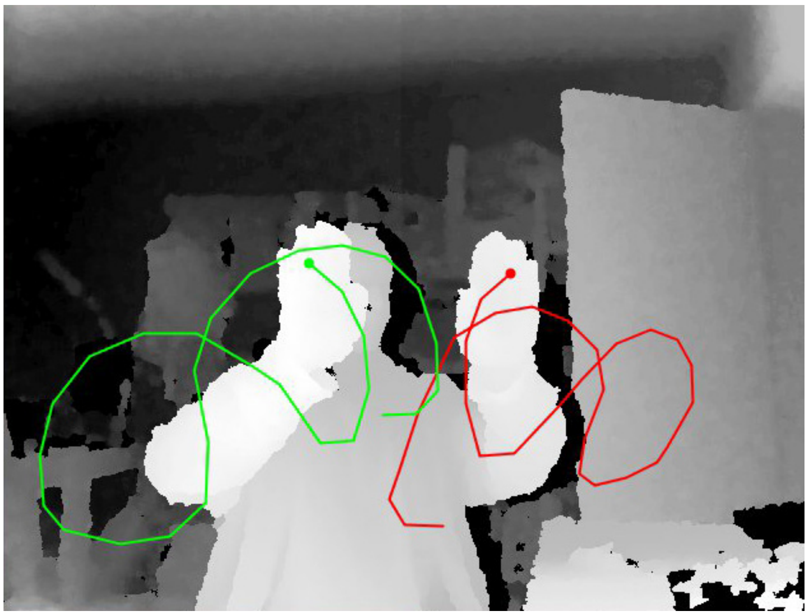


Figure 2.15: NiTE Hand Tracking application shows the trail of the tracked hand in different colors. [?]

- The movement should not be too slow or too fast.
- WAVE should consist of at least 5 horizontal movements left-right or right-left.
- CLICK should be at least 20 cm long enough and performed towards the sensor.
- If there is a difficulty in gaining focus, the user must stand closer to the sensor around at 2m.

2.2.4 Gesture Classification and Prediction

Like most other recognition systems such as speech recognition and biometrics, the tasks of hand gesture recognition involve modeling, feature extraction, training, classification and prediction. Though the alternatives such as Dynamic Programming (DP) matching algorithms have been attempted, the most successful solutions involve feature-based statistical learning algorithms [?]. Previous sections explained how a hand gesture is modeled, features are extracted from raw depth images, and the following sections discuss how extracted features are trained, classified and predicted.

In this thesis, we intend to employ a stochastic machine learning technique based on Adaptive Naive Bayes Classifier with the help of Gesture Recognition Toolkit. ANBC is an extension to the well-known Naive Bayes, one of the most commonly used supervised learning algorithms that works very well on both basic and more complex recognition problems.

2.2.4.1 Adaptive Naive Bayes Classifier

ANBC [**?**] is a supervised learning algorithm that can be used to classify any type of N-dimensional signal. It is based on simple probabilistic classifier called Naive Bayes classifier. It fundamentally works by fitting an N-dimensional Gaussian distribution to each class during the training phase. New gestures can then be recognized in the prediction phase by finding the gesture that results.

ANBC like Naive Bayes classifier makes a number of basic assumptions with input data that all the variables in the data are independent. However, despite these naive assumptions, Naive Bayes Classifiers have proved successful in many real-world classification problems [**?**]. It has also been shown in a study that the Naive Bayes Classifier not only performs well with completely independent features, but also with functionally dependent features.

ANBC algorithm is based on Bayes theory and gives the likelihood of event A occurring, given the observation of event B. In the equation 2.2, $P(A)$ represents the prior probability of event A occurring and $P(B)$ is a normalizing factor to ensure that all the posterior probabilities sum to 1.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.2}$$

Training The weighting coefficient adds an important feature for the ANBC algorithm as it enables one general classifier to be trained with multidimensional inputs, even if a number of inputs are only relevant for one particular gesture. For example, if it is used to recognize hand gestures, the weighting coefficients would enable the classifier to recognize both left and right hand gestures independently, without the position of the left hand affecting the classification of a right-handed gesture. For example, hand gesture recognition using x,y,z position of palms of Left and Right hand will have 6 dimensional sample. In this case left hand gestures will have weights 1,1,1,0,0,0, right hand gestures will have weights 0,0,0,1,1,1 and both hand gestures will have weights 1,1,1,1,1,1.

Using the weighted Gaussian model, the ANBC algorithm requires $G(3N)$ parameters, assuming that each of the G gestures require specific values for the N-dimensional μ_k, σ_k^2 and ϕ_k vectors where $\mu_k, \sigma_k^2, \phi_k$ are mean, variance and weighting coefficients.

Assuming that ϕ_k is set by the user, μ_k and σ_k^2 values can easily be calculated in a supervised learning scenario by grouping the input training data X into a matrix containing M training examples each with N dimensions, into their corresponding classes. The values for μ and $\sigma^2$of each dimension n for each class k can then be estimated by computing the mean and variance of the grouped training data for each of the respective classes [**?**].

$$P(g_k|x) = \frac{P(x|g_k)P(g_k)}{\sum_{i=1}^{G} P(x|g_i)P(g_i)} \quad 1 \leq k \leq G \tag{2.3}$$

After the Gaussian models have been trained for each of the G classes, an unknown N-dimensional vector x can be classified as one of the G classes using the maximum a posterior probability estimate (MAP). MAP estimate classifies x as the k-th class that results in the maximum a posterior probability given by the equation 2.3

$$lnN(x|\Phi_k) \quad 1 \leq k \leq G \tag{2.4}$$

Rejection Threshold Using equation 2.4, an unknown N-dimensional vector x can be classified as one of the G classes from a trained ANBC model. If x actually comes from an unknown distribution that has not been modeled by one of the trained classes then, it will be incorrectly classified against the k th gesture that gives the maximum likelihood value. A rejection threshold must therefore be calculated for each of the G gestures to enable the algorithm to classify any of the G gestures from a continuous stream of data that also contains non-gestural data [**?**].

Online Training One key element of ANBC is that it can easily be made adaptive. Adding an adaptive online training phase to the common two-phase (training and prediction) provides some significant advantages for the recognition gestures. During online training phase the algorithm will not only perform real-time predictions on the continuous stream of input data, but it will also continue to train and refine the models for each gesture. This enables the user to initially train the algorithm with a low number of training samples and during the adaptive online training phase, the algorithm can continue to train and refine the initial models, creating a more robust model as the number of training samples increases.

Pros / Cons ANBC works well for the classification of static gestures and non-temporal pattern recognition. However, the main limitation of the ANBC is that, it does not work well when the data you want to classify, is not linearly separable because it uses a Gaussian distribution to represent each class. Also when ANBC is working with online training enabled, a small number of incorrectly labeled training examples can create a loose model that becomes less effective at each update step and ultimately lead to a poor performance and accuracy.

2.2.4.2 Gesture Recognition Toolkit (GRT)

GRT is a cross-platform open-source C++ library designed and developed mainly by Nicholas Gillian at MIT Media Lab to make real-time machine learning and gesture recognition [**?**]. Emphasis is placed on the ease of use with a consistent, minimalist design that promotes accessibility while supporting flexibility and customization for advanced users. The toolkit features a broad range of classification and regression algorithms, and has extensive support for building real-time systems. GRT includes algorithms for signal processing, feature extraction and automatic gesture spotting.

In this thesis, we attempt to take advantage of GRT as framework to carry out most of the tasks involved in hand gesture recognition. Figure 2.16 shows that GRT provides the full fledge pipeline to build a real-time gesture recognition system.

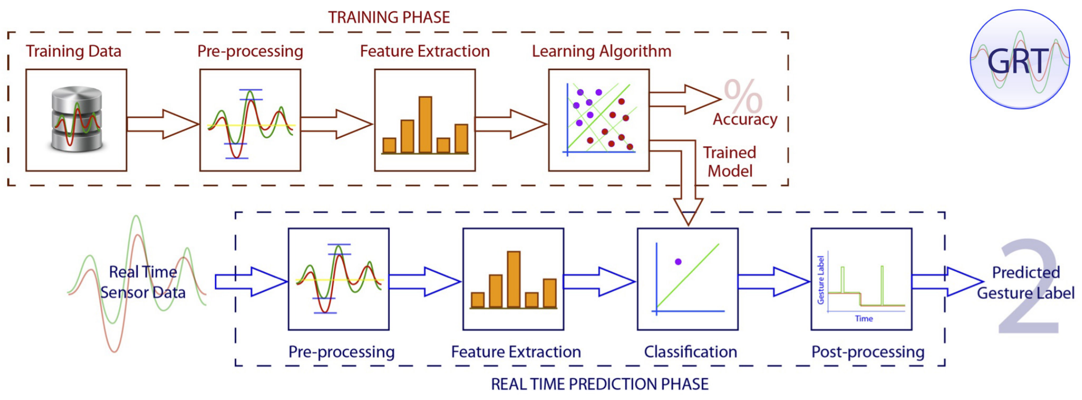


Figure 2.16: Stages of Gesture Recognition which are supported by GRT Recognition Pipeline. [**?**]

Pipeline GRT provides an API to reduce the need for boilerplate code to perform common functionality, such as passing data between algorithms or to per-process data sets. GRT uses an object-oriented modular architecture and it is built around a set of core modules and a gesture-recognition pipeline. The input to both the modules and pipeline consists of an N-dimensional double-precision vector, making the toolkit flexible to any type of input signal. The algorithms can be used as stand-alone classes; alternatively a gesture recognition pipeline can be used to chain modules together to create a more sophisticated gesture recognition system. Modularity of GRT pipeline offers developers opportunities to work on each stages of gesture recognition independently. Additionally, pipeline can be stored and loaded dynamically so that an compiled application can work in many different configurations.

ClassificationData Accurate labeling of dataset is very critical for machine learning problems. The toolkit thus contains an extensive support for recording, labeling and managing supervised and unsupervised datasets for classification, regression and time series analysis. *ClassificationData* is the data structure used for supervised learning problems and for most of the non temporal classification algorithms.

GRT allows us to store and load the training data in GRT format or Comma Separated Values (CSV). Since the training datasets are stored in human readable format, it enables us to add more samples which are collected separately or remove false data from the training dataset.

TrainingDataRecordingTimer Important part of the training phase is recording positive samples of modeled hand gestures. Hence, GRT provides a feature called *TrainingDataRecordingTimer* that sets recording and preparation time in milliseconds. Once it is started by calling *startRecording(prepationTime, recordTime)* method, it waits for given preparation time before it actually starts to store the data. This feature helps the trainer get into the right pose before samples are added to the training data and as well as train all the gestures for the same time duration.

Algorithms GRT features a broad range of machine-learning algorithms such as AdaBoost, Decision Trees, Dynamic Time Warping (DTW), Hidden Markov Models (HMM), K-Nearest Neighbor (KNN), Linear and Logistic Regression, Adaptive Naive Bayes (ANBC), Multilayer Perceptrons (MLP), Random Forests and Support Vector Machines (SVM) [**?**].

Null Rejection Another important feature of GRT is Null Rejections threshold. It means that algorithms can automatically spot the difference between trained gestures and unintended gestures that can happen when the user moves the hand in freely. It can be enabled by the method *enableNullRejection(true)* and the range of the null rejection region can be set by this method *setNullRejectionCoeff(double nullRejectionCoeff)* of the classifier. Algorithm such as the ANBC and N-Dimensional DTW, learn rejection thresholds from the training data, which are then used to automatically recognize valid gestures from a continuous stream of real-time data.

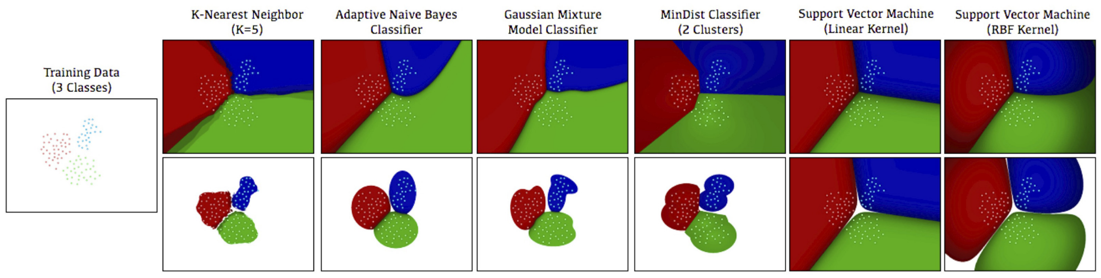


Figure 2.17: This shows the decision boundaries computed by training six of classification algorithms on an example dataset with 3 classes. The top row shows the predictions of each classifier with null rejection disabled. The bottom row shows the predictions of each classifier with null rejection enabled with null rejection coefficient of 3.0. [**?**]

Figure 2.17 shows that the decision boundaries computed by training six of classification algorithms on an example dataset with 3 classes. After training each classifier, each point in the two-dimensional feature space is colored by the likelihood of the predicted class label (red for class 1, green for class 2, blue for class 3). The top row shows the predictions of each classifier with null rejection disabled. The bottom row shows the predictions of each classifier with null rejection enabled with a coefficient of 3.0. Rejected points are colored white. Note that both the decision boundaries and null-rejection regions are different for each of the classifiers. This results from the several learning and prediction algorithms used by each classifier.

Scaling Normalization Real-time classification faces normalization problems when the range of training data differ from prediction input. To solve this problems, there are few solutions such as Z-score Standardization and Feature Scaling. GRT presents a simple solution called as Minimum-Maximum scaling.

Min-Max scaling rescales the range in [0, 1] or [-1, 1]. Selecting the target range depends on the nature of the data. Classifiers *enableScaling(true)* method scales input vector between the default min-max range that is from 0 to 1. The cost of having this bounded range is that model will end up with smaller standard deviations, which can suppress the effect of outliers. Equation 2.5 shows how Min-Max scaling is done.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{2.5}$$

Pre/Post Processing Modules In many real-world scenarios, the input to a classification algorithm must be preprocessed and have salient features extracted. GRT therefore supports a wide range of pre/post-processing modules such as Moving Average Filter, Class Label Filter and Class Label Change Filter, embedded feature extraction algorithms such as AdaBoost, dimensionality reduction techniques such as Principal Component Analysis (PCA) and unsupervised quantizers such as K-Means Quantizer, Self-Organizing Map Quantizer.

There will not be any need of preprocessing modules in this project since raw data received from depth sensor is processed by NiTE framework. However, post-processing modules such as Class Label Filter and Class Label Change Filter may be needed for a reasons that depth sensor samples 30 frames per second, therefore 30 input samples per second are supplied to the classifier for prediction and the output must be triggered once for every gesture.

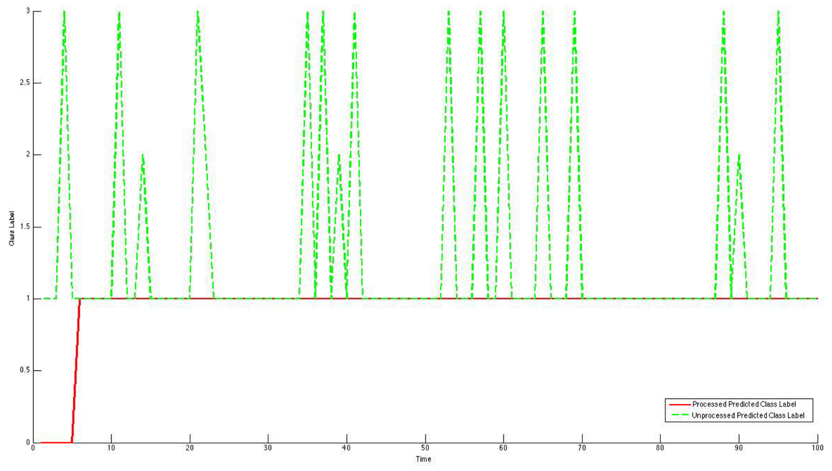


Figure 2.18: GRT Class Label Filter removes the sporadic prediction values and outputs buffered class label. [**?**]

Class Label Filter It is a useful post-processing module which can remove erroneous or sporadic prediction spikes that may be made by a classifier on a continuous input stream of data. Figure 2.18 that the classifier correctly outputs the predicted class label of 1 for a large majority of the time that a user is performing gesture 1. However, may be due to sensor noise or false samples in the training data, the classifier outputs the class label of 2. In this instance the class label filter can be used to remove these sporadic prediction values with the output of the class label filter in this instance being 1.

Class Label Filter module is controlled through two parameters: the minimum count value and buffer size value. The minimum count sets the minimum number of label values that must be present in the buffer to be output by the Class Label Filter. The size of the class labels buffer is set by the buffer size parameter. If there is more than one type of class label in the buffer then the class label with the maximum number of instances will be output. If the maximum number of instances for any class label in the buffer is less than the minimum count parameter then the Class Label Filter will output the default null rejection class label of 0.

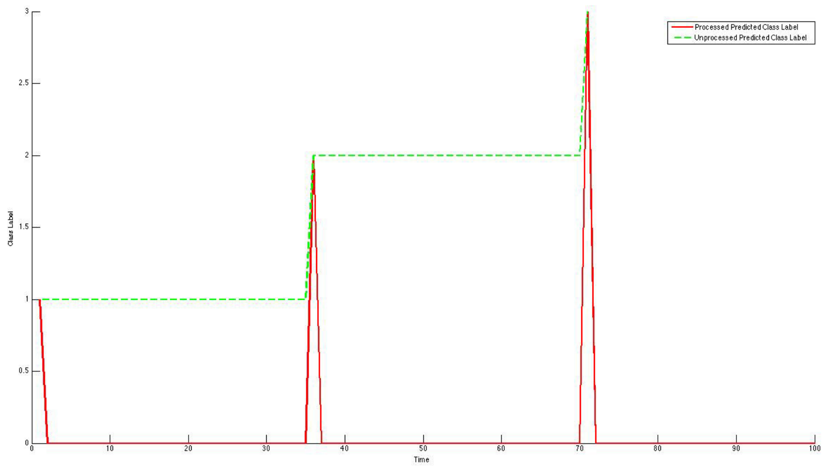
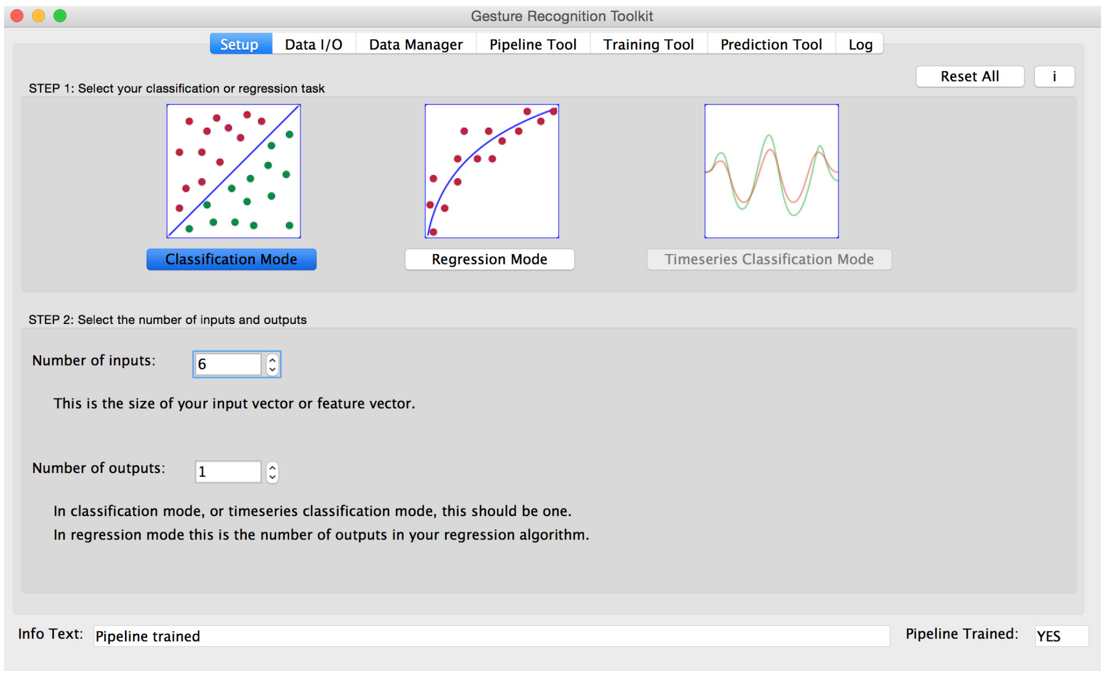


Figure 2.19: GRT Label Change Filter outputs only when there is change in the prediction. [?]

Class Label Change Filter It is one of the useful post-processing module that triggers when the predicted output of a classifier changes. Figure 2.19shows that, if the output stream of a classifier is 1,1,1,1,2,2,2,2,3,3, then the output of the filter would be 1,0,0,0,2,0,0,0,3,0. This module is useful to trigger a gesture once, if the user is gesticulating the same gesture for longer time duration. If the user intends to trigger the same gesture again, then hand position must be changed to another such as pointing the hand towards the ground, and gesticulate the gesture again.

GUI Figure 2.20 shows GRT-GUI which is an application that provides an easy-to-use graphical interface developed in C++ to setup and configure a gesture recognition pipeline that can be used for classification, regression, or time-series analysis. Data and control commands are streamed in and out of this application as Open Sound Control (OSC) packets via UDP . Therefore, it acts as a standalone application to record, label,



STEP 1: Select your classification or regression task

STEP 2: Select the number of inputs and outputs

Number of inputs: 6

This is the size of your input vector or feature vector.

Number of outputs: 1

In classification mode, or timeseries classification mode, this should be one.
In regression mode this is the number of outputs in your regression algorithm.

Info Text: Pipeline trained

Pipeline Trained: YES

Figure 2.20: GRT GUI is an standalone application to quick prototype by recording, labeling, saving, loading, testing the training data and to perform a real-time prediction. [?]

save, load and test the training data and performs a real-time prediction for the incoming data, send output to another application.

2.3 Summary

This chapter has discussed the concepts of hand gesture recognition using skeletal points tracking with the help of depth camera. It has also talked about the specifications of Aldebaran NAO. Furthermore, It has discussed the features of the machine learning tool GRT that helps us to carry out the classification and prediction of hand gestures in real time.

Chapter 3

Hand Gesture Recognition for Human-Robot Interaction

To build an effective and easy to use hand gesture recognition system for NAO, various tools and technologies are studied during this thesis. The main challenge is to find a solution that can integrate all essential components into a robust system. However, due to the computational and compatibility limitations of NAO [**?**], we have faced problems in implementing few contemplated solutions. Finally, the successful solution in achieving the goal will be discussed in the following section.

3.1 Implementation

After analyzing the disadvantages of other experimental designs, the final design is chosen to build an efficient real-time hand gesture recognition for human-robot interaction using skeletal points. Figure 3.1 shows the architecture of the solution that is implemented during this thesis by grouping many components into 4 different modules which serve several purposes. Each module is implemented in different environment as shown in the figure and they communicate with one another to complete the data flow. All these modules use a common configuration file named as *hri.json* that contains information such as port number, host name and log path.

3.1.1 Human-Robot Interaction (HRI) Module

HRI module is implemented first to get the raw data from the depth sensor and process it to track the skeletal joint positions in real world coordinates. It is developed in C++ using a core library called Boost and NiTE 2 framework is used for the purpose of

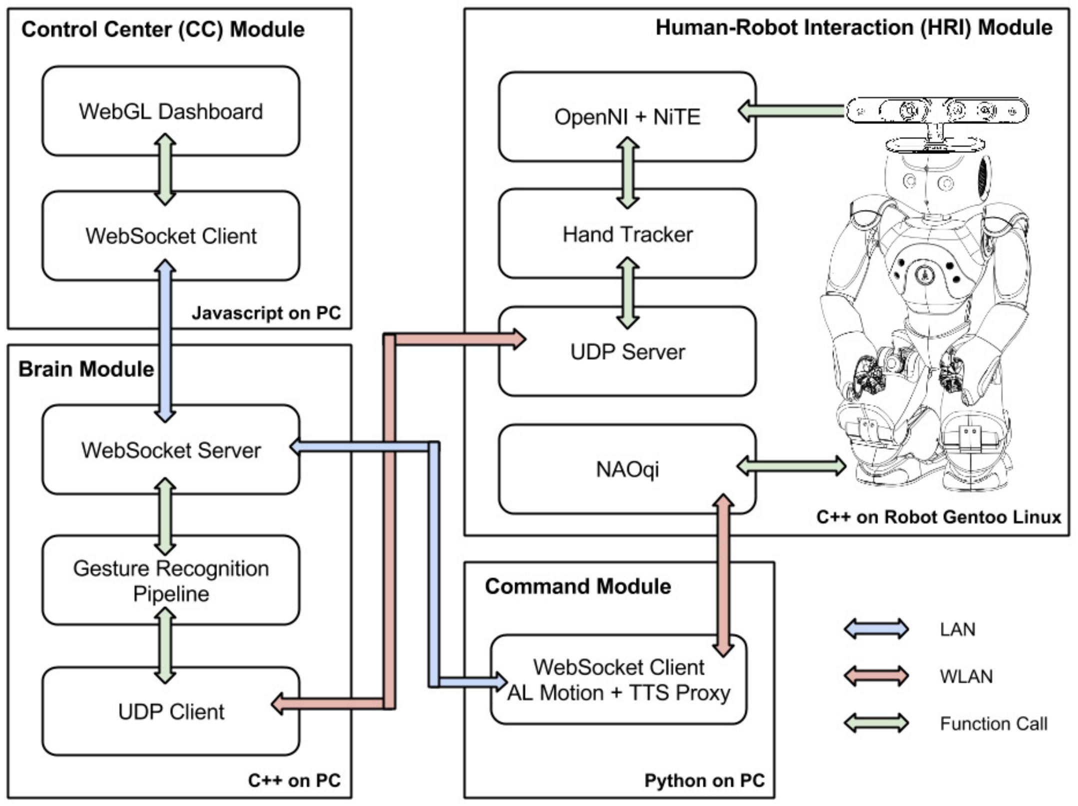


Figure 3.1: Architecture of the proposed solution to build a real time hand gesture recognition using depth camera.

skeletal joints tracking. This module is deployed on the general purpose computer that is running inside the robot with necessary libraries and drivers.

Boost is a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudo random number generation, multi threading, image processing, regular expressions, and unit testing. It contains over eighty individual libraries.

HRI module is composed of 3 components which are UDP Server, Gesture and Skeleton tracker. Figure 3.2 shows the data and control flow of this module where the user is asked to select Gesture or Skeleton tracker, when the program is started. It creates 2 threads depending on the selection:

- UDP Server thread - Asynchronously send data to the client and thread is always running.
- Gesture or Skeleton tracker thread - A loop in the thread polls for a new frame from the depth camera till some key is pressed. If loop is interrupted, then the thread is exited and finally program is closed.

Gesture and Skeleton tracker serve the purpose in extracting features from the raw data to implement a hand gesture recognition system. However, Skeleton tracker tracks 15 skeletal points in the human body and that leads to very intensive computation. Due to processing limitations of NAO, we chose to use Gesture tracker as it tracks only hand joints. Following sections describe internal working of HRI module.

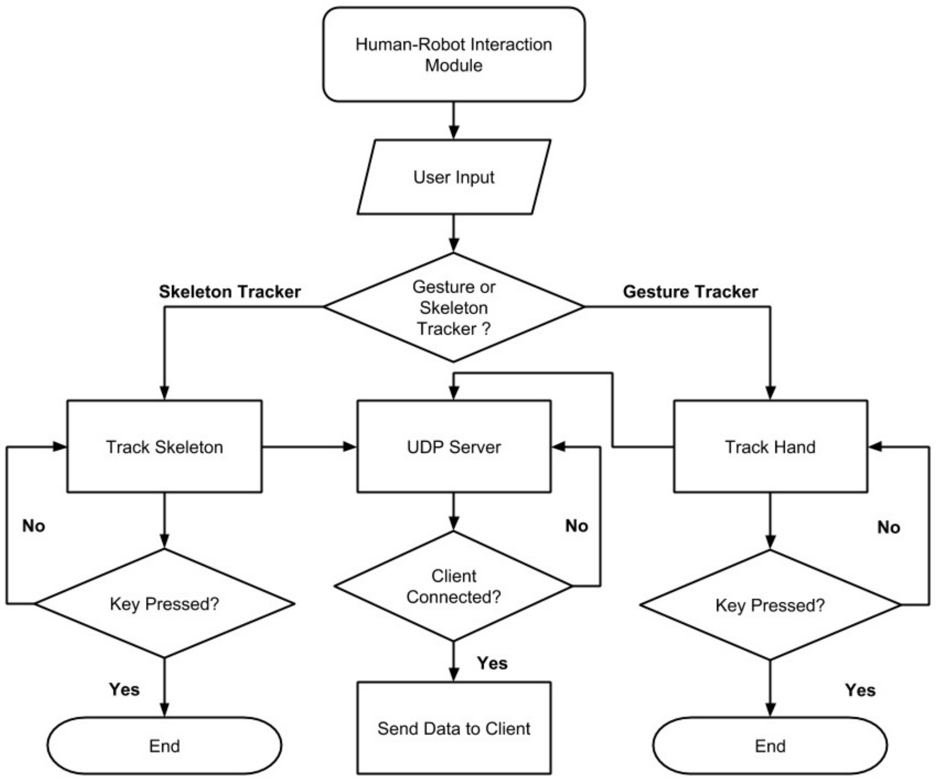


Figure 3.2: Flow chart illustrates the control and data flow of HRI module.

3.1.1.1 UDP Server

HRI module has to process the raw information from the depth camera and it has to send it to Brain module for the purpose of gesture recognition. As show in the architecture diagram 3.1, Brain module must be connected via Wireless Local Area Network (WLAN). WLAN at 2.4GHz readily is available on NAO and lead us to a solution, where we have to choose an UDP protocol to transmit the processed data from depth camera. UDP is chosen over other protocols because depth camera produces 30 depth images per second and transferring such a large amount of data using conventional communication technologies such as TCP will be create much overhead and delay in the communication.

Due to asynchronous requirement of the server, Boost Asio library is used to implement UDP server. Boost.Asio is a cross-platform C++ library for network and low-level

I/O programming that provides developers with a consistent asynchronous model using a modern C++ approach.

UDP Server is basically an asynchronous programs that creates an UDP socket and listens to an port on the local machine. In this case, we have created a common configuration file named as *hri.json* that contains port numbers for each module in this project. Therefore, this server listens to the 5005 on NAO and waiting for the clients to connect.

Once the client is connected, it stores the endpoint details of the client such as IP address and the port number of the UDP client (Brain module), so that it can communicate with the Brain module whenever there is some data to be transmitted. Asynchronous functionality Boost.Asio calls the callback handler only when there is communication with the clients and waits in the thread for the next communication.

3.1.1.2 Gesture Tracker

Gesture tracker is a component of HRI module that makes use of NiTE framework to localize the hand of the user in the field of view and track the hand position till the hand leaves the field of view (FOV) or hand is touching another object or hidden by an object.

It uses *HandTracker* class of NiTE framework and it needs to go through following steps before it can track a hand. Section 2.2.3.2 discusses extensively about the functionalities of NiTE framework.

- NiTE framework must be initialized using *nite::initialize()* function.
- Depth camera must be connected and *nite::HandTracker* must be created using OpenNI compatible device id. If not, default depth camera will be selected.
- NiTE focus gesture WAVE must be initiated to localize the hand at first.
- *nite::HandTrackerFrameRef* must be read continuously for a new gesture.
- If WAVE gesture is detected, then hand tracking will be started using the position of hand that triggered the gesture.

Once the hand is been tracked, the hand will be added an id and it will be added to *HandTrackerFrameRef*. NiTE framework allow users to add many number of hands and it will be tracked till there is enough computation power and hands are not overlapping. *HandTrackerFrameRef* contains the array of all active hands and every hand is an object of *nite::HandData*. It contains the position of the hand in 3 dimensional float stored in a class called *Point3f*.

Unlike *nite::UserTracker*, *HandTracker* class can return only the hand position in the space and it can not specify whether it is a left or right hand. It is very necessary information for hand gesture training and classification because confused hand names will lead to a false model of the hand gesture and ultimately resulting in a bad performance. Hence, we have implemented a simple logic with the help of an assumption that user will gesticulate the focus gesture only in the order of right hand first and left hand second.

However, functionalities gesture tracker are not only to track hand, but also send these information to Brain module via UDP. Therefore, C++ *nite::HandData* objects must be serialized before transmitted over the network. Therefore, we chose JSON serialization and send them across the network as strings as shown in 3.1.1.2

```
{ "RIGHT": ["275.456", "339.026", "1841.850"], "LEFT":
   ["-456.289", "353.880", "1761.360"] }
```

Furthermore, HRI module send informations such as detected focus gesture and info messages to Brain module as shown in 3.1.1.2 to be displayed on the control center dashboard. Info messages helps us to know the status of the hand tracking algorithm which is the core component of HRI module.

```
{"GESTURE":"WAVE"} {"GESTURE":"CLICK"} {"INFO": "Found new
   hand with id 1"} {"INFO": "LEFT Hand is lost"} {"INFO":
   "RIGHT Hand is lost"} {"INFO": "Both hands are lost"}
   {"INFO": "LEFT Hand is at FOV"}
```

3.1.1.3 Skeleton Tracker

Skeleton Tracker is a component of HRI module that is more complex and computational intensive, since it uses *nite::UserTracker* to track 15 bone joints of human body. Like Gesture Tracker in the section 3.1.1.2, this component has to follow few procedure before tracking and it starts with an UDP server to unicast joint positions to Brain module.

- NiTE framework must be initialized using *nite::initialize()* function.
- Depth camera must be connected and *nite::UserTracker* must be created using OpenNI compatible device id. If not, default depth camera will be selected.

- Pose in front the camera as shown in the figure 3.3 to let the algorithm calibrate the body position.
- *nite::UserTrackerFrameRef* must be read continuously for a new user and if a new user is found, skeleton tracking will be started.

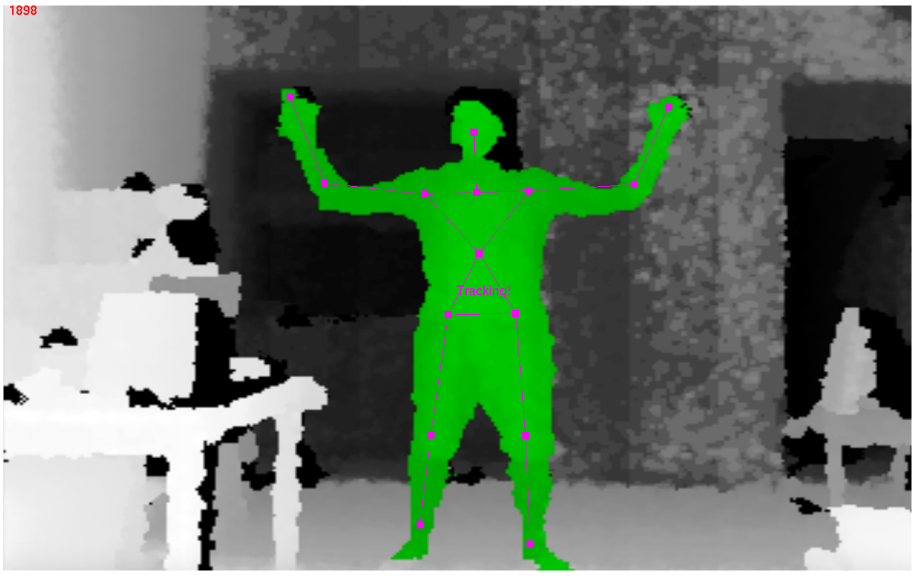


Figure 3.3: Image captured while NiTE tracks 15 skeletal joints of the user using depth camera Asus Xtion.

Unlike *nite::HandTracker*, *UserTracker* class of NiTE uses complex algorithms to keep tracking the skeleton even when the user poses in many ways. Therefore, it needs the data provided by NiTE framework which contain models of 1 million training samples. In addition, *UserTracker* can return 15 skeletal joints position and orientation and they are labeled by the joint name. This feature helps us to avoid the implementation to find the hand name. Moreover, details of joint orientations offer us a chance to calculate positions not only in Cartesian coordinates, but also in spherical coordinates system which is essential for many complex hand gesture recognition solutions [**?**]. Furthermore, *SkeletonJoint* class indicates how sure the NiTE skeleton algorithm is about the joint position. The value is between 0 and 1, with increasing value indicating increasing confidence. Section 2.2.3.2 discusses extensively about the algorithm of NiTE.

Finally, Skeleton tracker serializes the C++ *nite::UserData* objects to JSON and sends asynchronously to the client for further gesture recognition procedures.

3.1.2 Brain Module

Brain module is the core functional part of this thesis. It is named as Brain since it refers to the anatomical brain that plays the vital role of the human life in learning, classifying, predicting and decision making.

Brain module is composed of 3 components which are UDP Client, Brain (Gesture Recognition Pipeline) and WebSocket Server. Figure 3.4 shows the data flow of this module where the user is asked to select Prediction or Training or Hand Viewer mode, when the program is started. It creates a thread and runs a loop on the main thread depending on the selection:

- UDP Client thread - Asynchronously receiving data from HRI module and thread is always running.
- Prediction or Training of Hand Viewer on main program thread - Loop in the main thread run always and check if the Brain module is in prediction or training mode. If loop is interrupted, then the thread is exited and finally program is closed.

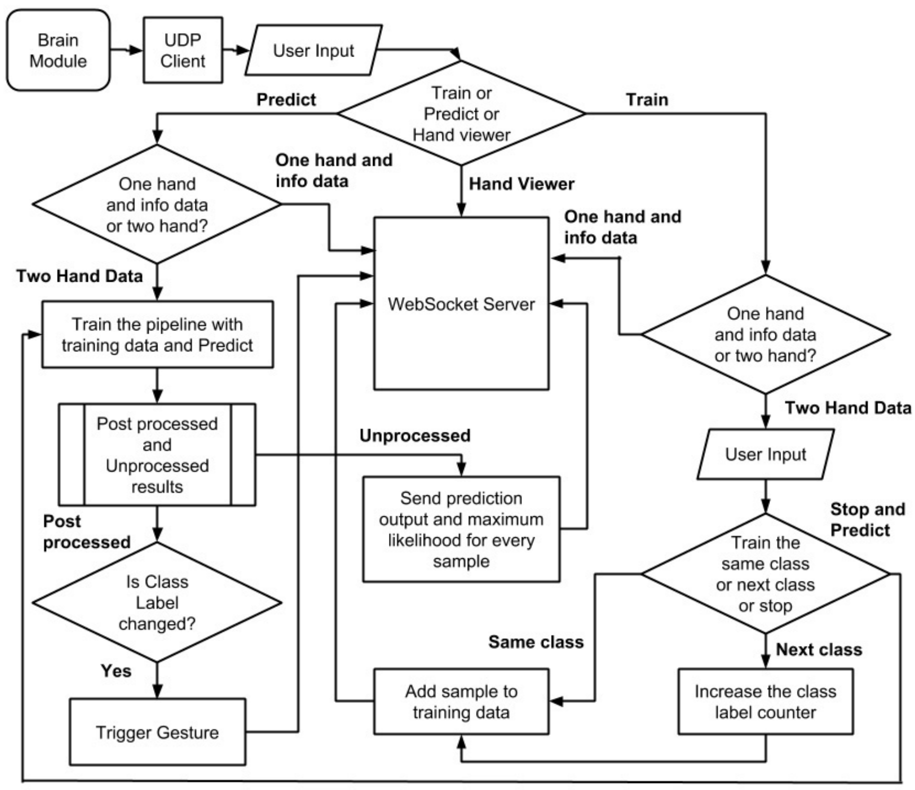

Brain Module
UDP Client
User Input
Train or Predict or Hand viewer
Predict
Train
Hand Viewer
One hand and info data or two hand?
One hand and info data
Two Hand Data
WebSocket Server
Train the pipeline with training data and Predict
Post processed and Unprocessed results
Unprocessed
Send prediction output and maximum likelihood for every sample
Post processed
Is Class Label changed?
Yes
Trigger Gesture
User Input
Train the same class or next class or stop
Stop and Predict
Same class
Next class
Add sample to training data
Increase the class label counter

Figure 3.4: Flow chart illustrates the control and data flow of Brain module.

3.1.2.1 UDP Client

Brain module receives processed information such as joint positions, detection of focus gestures and info messages from the HRI module as UDP stream of JSON strings via WLAN. Like the UDP Server built inside HRI module, this is also an asynchronous client that starts at port 5006 and connects to the server by resolving the *serverHost-Name* and port number from the common configuration file. Once it is connected, it receives the data from HRI module, when it is started tracking a hand or skeleton and asynchronously calls the callback handler.

Since data is transmitted as JSON strings, it has to be parsed and relevant informations must be extracted. For this purpose RapidJSON parser is used. Data flow of Brain module is mainly handled in the callback handler of UDP client because it acts as a source of input. Whenever there is a new data arrived, this asynchronous callback handler is called and it does the following tasks as shown in the Figure 3.4 :

- Extract only newly received data from the buffer by trimming the JSON
- Parse the trimmed JSON to populate hand data vectors.
- If focus gesture or info messages or only one hand data is received, send it via WebSocket to the clients
- Check if the module is Prediction or Training or Hand Viewer mode
- In the prediction mode :
  - If the positions of both hands are received, predict the class label
  - Add predicted class label and maximum likelihood to the sample, and send it via WebSocket
  - If there is a class label not than 0, then send the respective gesture name via WebSocket
- If it is in the training mode and both hands are received, then add them to the training data
- If it is in the hand viewer mode, just forward all the data to the clients via WebSocket

3.1.2.2 Brain

This is the core component of Brain module that plays a vital role in training, classifying and predicting the hand gestures. As described in the section 2.2.4.2, this component is based on the gesture recognition pipeline provided by GRT.

Figure 3.4 shows various tasks involved in training and predicting phase of this module. However, GRT pipeline must be configured and customized in order to be a productive gesture recognition system.

Classifier ANBC is used in this thesis as described in the section 2.2.4.1. Training data for the same gesture will vary in range from person to person and position to position. Therefore the classifier is enabled for Min-Max scaling that is basically a normalization by rescaling the values between 0 to 1. This is done by calling *enableScaling(true)* function of the classifier.

Null Rejection Enabling the scaling with ANBC will classify every input samples to belong to any of the class and thereby, do not have the ability to detect non-gestures. To avoid this catastrophe GRT offers Null Rejection features to the algorithms, by this function *enableNullRejection(true)* and also provides a function to set how big the rejection region should be, by *setNullRejectionCoeff(1).*

Post Processing As discussed in the section 2.2.4.2, prediction output must be post processed in order to avoid false prediction spikes. Therefore, class label filter is added to the pipeline by with this function *ClassLabelFilter(30,60).* Minimum count is set to 30 with the buffer size of 60 for the reason that the user must gesticulate for minimum of one second since depth camera produces 30 frames per second. Additionally *ClassLabelChangeFilter()* is added so that there is only one output of the predicted class label, when there is a change in the gesture and all other time it outputs 0, that is reserved for non-gesture.

Training Data We used *ClassificationData* data structure of GRT to collect training data of static gestures. It must be initialized with number of dimensions the samples will be. In our thesis we modeled hand gestures with two hand positions in 3 dimensional Cartesian coordinates, therefore training dataset has 6 dimensions. As described in the section 2.2.4.2, GRT enables us to execute various operations on the training data such as recording, labeling, partitioning and testing.

Training When Brain is set to training mode, it starts the *TrainingDataRecording-Timer*. We have configured 20 seconds recording time and 15 seconds preparation time. Preparation time helps the trainer to go in front of depth camera and stay in the pose of the gesture that is going to be recorded. Furthermore, It initializes the Class Label to 1 and it will be increased by one for other classes. Class Label can not be assigned to 0 because GRT reserves it for non-gestures. If positions of left and right hand are received from the HRI module, Brain starts to add the samples with the chosen Class Label to the training dataset till the timer is in recording mode and simultaneously it sends to received samples via WebSocket to the clients to visualize. When the recording timer is stopped, Brain requests the trainer to choose any of the following options :

- Train the same class again - New samples will be added to the training dataset for same Class Label.
- Train the next class - Class Label is increased by one and new samples are added.
- Stop training and go to prediction mode - Saves the training dataset to a file named as *hri-training-dataset.txt* and trains the pipeline and goes into prediction mode

Prediction When Brain is set to prediction mode, first thing it does, is loading the training labeled classification data and train the pipeline to create models for each gesture. Second step is to look for any specific pipeline configuration such as classifier and pre/post processing modules. Such configurations can also be loaded into pipeline as GRT pipeline files. This feature of GRT offers us an opportunity to run the gesture recognition application using dynamic configurations. Once Brain starts to receive input samples via UDP, it feeds it to the pipeline to predict. Finally, the prediction results such as predicted class label, maximum likelihood, class distances and weights are returned by the pipeline. Flexible GRT pipeline provides many more features such as post-processed and unprocessed prediction results. Therefore, the prediction results for every input sample can be obtained. The post-processed result will allow Brain to send the detected gesture only once, even if the user is continuously gesticulating the same gesture.

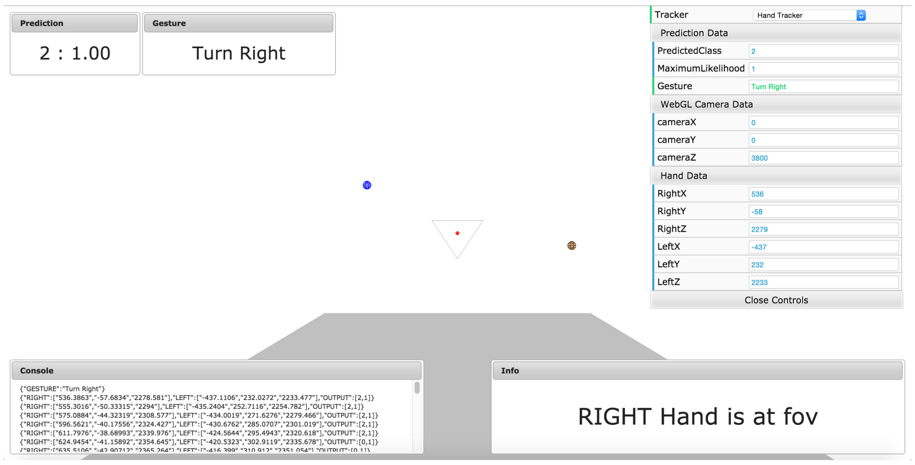
3.1.2.3 WebSocket Server

WebSocketServer class is developed using websocketpp C++ library that basically uses BOOST libraries. It is a simple implementation of WebSocket server that listens to the port number 5008. The port number can be configured dynamically by loading the common configuration file. WebSocket class is initialized by UDP Client class and keeps the server running in a separate thread. Once clients such as CC module and Command module are connected, it stores the endpoint connection handlers of them for later communication.

3.1.3 Control Center (CC) Module

Control Center plays an important role in this thesis. It is the eye that visualizes the internal status of the system. It is first built for the purpose of visually render the skeletal points of the human body that is being tracked by NiTE. Later, it has become one place to interact with the whole system.

CC is developed in Javascript with the help of WebGL and jQuery. The cloud computing is day by day pushing computer applications to the Internet, which allows softwares to be operated using internet-enabled devices. Due to this reason browser based cross-compatible applications are getting popular and that leads to the huge involvement of development in Javascript. Therefore, we chose a cross-compatible platform that work out of the box than implementing the same in C++ using OpenGL.



Prediction	Gesture
2 : 1.00	Turn Right

Tracker	Hand Tracker
Prediction Data	
PredictedClass	2
MaximumLikelihood	1
Gesture	Turn Right
WebGL Camera Data	
cameraX	0
cameraY	0
cameraZ	3800
Hand Data	
RightX	536
RightY	-58
RightZ	2279
LeftX	-437
LeftY	232
LeftZ	2233

```
{"GESTURE":"Turn Right"}
{"RIGHT":["536.3863","-57.6834","2278.581"],"LEFT":["-437.1106","232.0272","2233.477"],"OUTPUT":[2,1]}
{"RIGHT":["555.3016","-50.33315","2294"],"LEFT":["-435.2404","252.7116","2254.782"],"OUTPUT":[2,1]}
{"RIGHT":["575.0884","-44.32319","2308.577"],"LEFT":["-434.0019","271.6276","2279.466"],"OUTPUT":[2,1]}
{"RIGHT":["596.5621","-40.17556","2324.427"],"LEFT":["-430.6762","285.0707","2301.019"],"OUTPUT":[2,1]}
{"RIGHT":["611.7976","-38.68993","2339.976"],"LEFT":["-424.5644","295.4943","2320.618"],"OUTPUT":[2,1]}
{"RIGHT":["624.9454","-41.15892","2354.645"],"LEFT":["-420.5323","302.9119","2335.678"],"OUTPUT":[0,1]}
{"RIGHT":["635.5106","-42.90712","2365.264"],"LEFT":["-416.399","310.912","2351.054"],"OUTPUT":[0,1]}
```

RIGHT Hand is at fov

Figure 3.5: Control Center displays received data of hand positions and prediction results

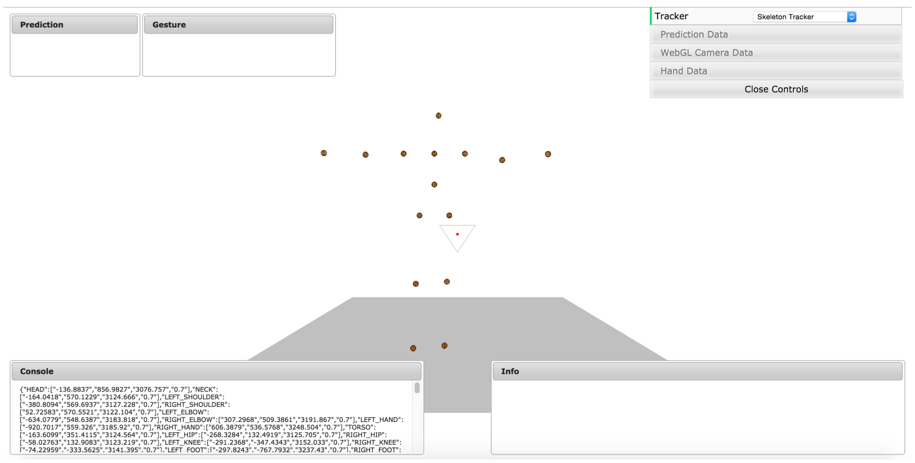
Javascript It is a dynamic programming language whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. However, It is also used in server-side programming with runtime environments such as Node.js, game development and the creation of desktop and mobile applications.

ThreeJS It is a lightweight 3D library with a very low level of complexity, written purely in Javascript that can render 3D objects in various renderer such as canvas, svg, CSS3D and WebGL. In this thesis, we have chosen WebGL renderer to implement the Control Center since it is faster than others in rendering tracked skeletal points at 30 frames per second.

WebSocket Client CC receives the data from Brain modules via WebSocket. The client uses the native Javascript WebSocket implementation that is supported by many latest browsers. It connects to the WebSocket server that is listening on the port 5008. When the client receives the data, it updates the data buffer asynchronously.

Architecture Control Center is implemented in MV* (Model View) design pattern that is quite popular among Javascript developers. Since the requirement of this module needs many libraries, a dependency injection library called RequireJS is used to load all the libraries when the application is opened in the browser.

Libraries Along with ThreeJS, libraries such as jQuery, underscore, TrackBallControl and datGUI are used in this module. jQuery is most common library for Document Object Model (DOM) manipulation in the browser. Operations on arrays and objects are made easier with the help of underscore. TrackBallControl allows to do manipulations such as rotate, revolve and transform the 3D objects which are rendered by WebGL. datGUI is a lightweight simple library to create GUI elements to build a dashboard in few lines of code.



```
{"HEAD":["-136.8837","856.9827","3076.757","0.7"],"NECK":
["-164.0418","570.1229","3124.666","0.7"],"LEFT_SHOULDER":
["-380.8094","569.6937","3127.228","0.7"],"RIGHT_SHOULDER":
["52.72583","570.5521","3122.104","0.7"],"LEFT_ELBOW":
["-634.0779","548.6387","3183.818","0.7"],"RIGHT_ELBOW":["307.2968","509.3861","3191.867","0.7"],"LEFT_HAND":
["-920.7017","559.326","3185.92","0.7"],"RIGHT_HAND":["606.3879","536.5768","3248.504","0.7"],"TORSO":
["-163.6099","351.4115","3124.564","0.7"],"LEFT_HIP":["-268.3284","132.4919","3125.705","0.7"],"RIGHT_HIP":
["-58.02763","132.9083","3123.219","0.7"],"LEFT_KNEE":["-291.2368","-347.4343","3152.033","0.7"],"RIGHT_KNEE":
["-74.22959","-333.5625","3141.395","0.7"],"LEFT_FOOT":["-297.8243","-767.7932","3237.43","0.7"],"RIGHT_FOOT":
```

Figure 3.6: Control Center renders real time positions of 15 human skeleton joints.

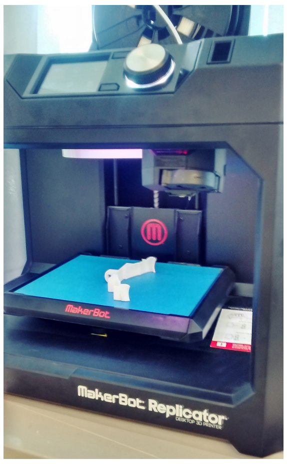
Model and View To avoid complexity, this Javascript application does not have any sophisticated model. It simply uses an array named *skeletonBuffer* that holds the JSON data received via WebSocket. All these actions are carried out in the store of the application. View does large part of the work for CC. At first it initializes the DOM and add GUI elements to it. Then, ThreeJS scene is created with WebGL renderer and adds a perspective camera, a plane geometry as a base and a triangle to show origin of the sensor. By default CC is in hand tracking mode and it creates two spheres to visualize the position of left and right hand. In skeleton tracking mode it creates 15 spheres two show all the skeletal points that are being tracked by NiTE. Control Center offers us to replay the positions of joints by storing them to a file and selecting *Hand Tracker From Data* option in the GUI. View automatically iterates through all the objects in the array and renders them at 60 fps.

User Interface (UI) Figure 3.5 the dashboard of the Control Center. Console box is an UI element that shows all the incoming data via WebSocket. It allows us to scroll through the data, if there is a necessity to cross check the data. Right bottom shows Info box which is created for the purpose of showing all intercommunication messages among all the modules. For example, *RIGHT Hand is at FOV* is an info message triggered by NiTE to inform that hand is closer to the field of view (FOV) and it may lose the hand. Left top corner displays two UI elements which are meant to show the prediction result for every input sample and recognized gesture that is triggered only after gesticulating it for more than one second. Furthermore, top right corner of the dashboard reveals more internal variables such as WebGL camera positions and real time hand in 3 dimensional Cartesian coordinates. CC can not only render hand joints, but also complete human skeleton with 15 skeletal point which are being tracked as shown in the figure 3.6. It also allows us to save tracking data to a json file and replay them by choosing appropriate mode from the drop down list on the top right corner.

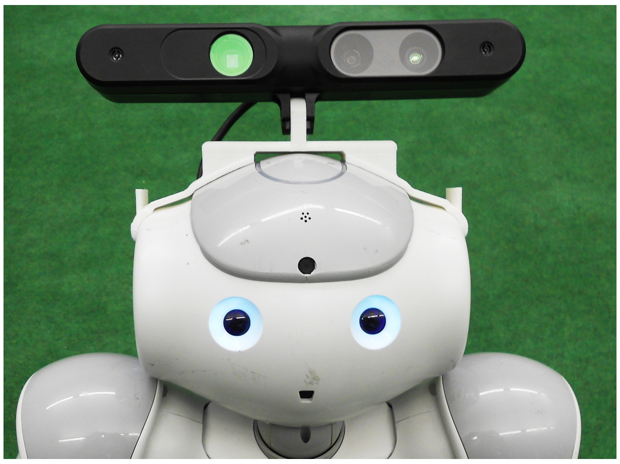
3.1.4 Command Module

Last but not the least module to complete the functionalities of our hand gesture recognition system is the Command module. All other modules which are described above need the Command module to interact with the robot.

Command module is developed in Python with WebSocket and NAOqi libraries. Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express



3D printing the mount



Asus Xtion mounted on NAO

Figure 3.7: 3D printed head mount to support the depth camera on the head of NAO.

concepts in fewer lines of code than would be possible in languages such as C++ or Java.

Command modules initiates the WebSocket Client and it connects to the Brain modules WebSocket Server at a given port number by loading the common configuration file. WebSocket client keeps the main thread run forever and it executes the respective call back handlers. When there is a new message, it calls the *onMessage* handler and parses the received JSON data to a python object. Whenever gesture data is received, it is translated to a robotic speech and motion via NAOqi proxies.

We have used ALMotions Locomotion Control extensively to move the robot from one position to another based on the recognized hand gesture such as "Turn Left" or "Move Right". Additionally, Gesture-To-Gesture actions where a human hand gesture is translated to the robot hand gesture by using the Joint Control of ALMotion module.

3.1.5 Head Mount

As described earlier in the section 2.1.4, integrated hardwares of NAO Vision are not sufficient to provide precise three dimensional data to the complex algorithms to track human skeletal joints. Therefore, Asus Xtion PRO LIVE 3D camera is chosen to be used for this gesture recognition system. Section 3.1 shows the final architecture that proposes to mount Asus Xtion on the head of the robot.

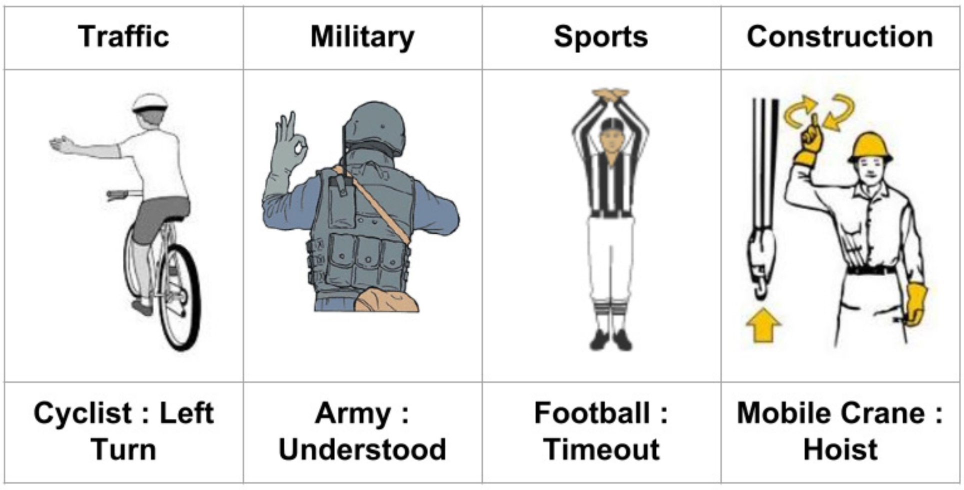
3D Printed NAO Xtion Mount We have found a solution that is designed by emotion-robotics.com. Therefore, we have used their 3D model to print it using MakerBot Replicator 5th Generation 3D Printer as shown in the figure 3.7. The original base of Asus Xtion is removed and the camera is screwed with the 3D printed mount, and easily fixed on the head of the robot.

3.2 Gesture Recognition

Above sections described the necessary tools that are implemented to execute a real time hand gesture recognition system. In this thesis, we have decided to train the system with static gestures. However, the system can be easily extended to recognize temporal gestures with the flexibility of GRT. Initially a set of simple gestures are chosen and the training data is collected for all those gestures.

3.2.1 Hand Gestures Modeling

In this thesis, we have modeled five static hand gestures involving both the hands of the user. These are communicative hand gestures and they symbolize few referential action. Apart from Sign Language used by people with speech disability, various hand gestures are being used by humans in their day to day living. Figure 3.8 shows the hand signals used by different personnels in wide variety of application.



Traffic	Military	Sports	Construction
Cyclist : Left Turn	Army : Understood	Football : Timeout	Mobile Crane : Hoist

Figure 3.8: Non-Verbal hand signals used by different personnels in wide variety of application. [?]

This thesis focuses on hand gesture recognition to felicitate Human-Robot interactions. One greater application using hand gestures for robots is commanding the robot to move to another position. Additionally it could translate the gestures to spoken words to help people with speech disability.

Therefore, we have chosen five simple static gestures as shown in the figure 3.14 which are conceptualized by the traffic police hand signals. All the gestures are modeled to the direction of the user and they will be understood as mirrored gestures. For example, left side of the user will be right side to the robot that is facing the user. Additionally our system makes use of two dynamic gestures of NiTE which are used as focus gestures to gain control or start hand tracking.

Walk It is gesticulated as shown in the figure 3.12 by holding the left and right hand up. It refers to an action that keep moving in the forward direction.

Turn Left It is gesticulated as shown in the figure 3.10 by holding the right hand up and left hand wide open. It refers to an action that turn to left and stay in position.

Turn Right It is gesticulated as shown in the figure 3.9 by holding the left hand up and right hand wide open. It refers to an action that turn to right and stay in position.

Move Left It is gesticulated as shown in the figure 3.12 by holding the right hand down and left hand wide open. It refers to an action that turn to left and keep moving in the forward direction.

Move Right It is gesticulated as shown in the figure 3.12 by holding the left hand down and right hand wide open. It refers to an action that turn to right and keep moving in the forward direction.

3.2.2 Training

Our gesture recognition pipeline is configured to have 15 seconds preparation time and 20 seconds recording time with 6 dimensional input of both left and right hands at positions x, y and z in the Cartesian coordinates. Depth camera is at the origin of the coordinate system as shown in the figure 3.15.

Brain is set to training mode and CC is started to visualize the hand positions in order to align the trainer during the preparation time. Each gesture is isolated in time and gesticulated for 20 seconds. Samples are added to the training dataset and when the timer stopped the recording, Brain asked the trainer to train the same class again or

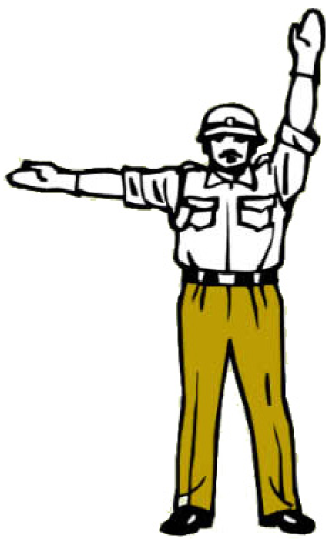


Figure 3.9: Turn Right Gesture

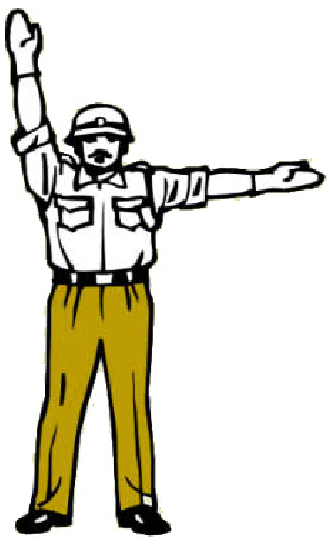


Figure 3.10: Turn Left Gesture

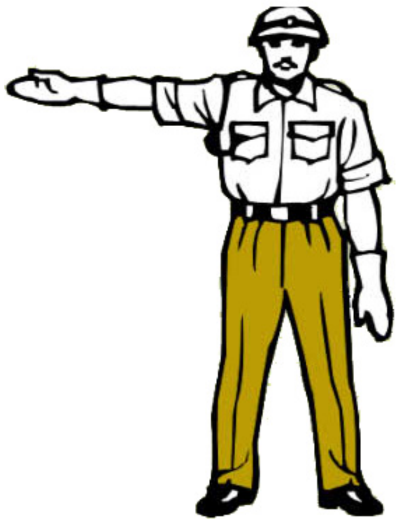


Figure 3.11: Move Right Gesture

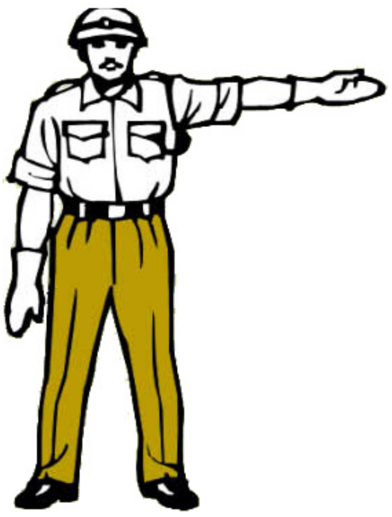


Figure 3.12: Move Left Gesture

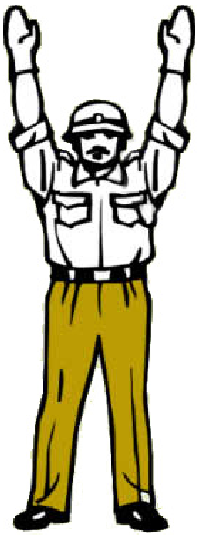


Figure 3.13: Walk Gesture

Figure 3.14: In this thesis, five static hand gestures are modeled based on the traffic police hand signals. [**?**]

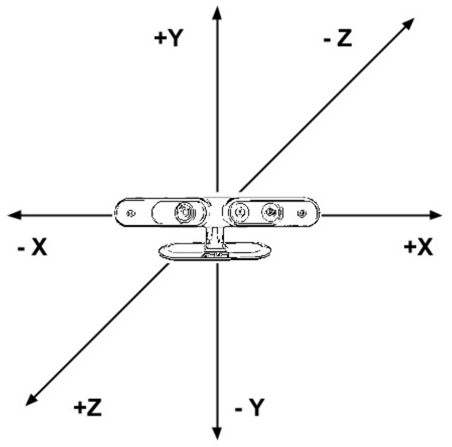


Figure 3.15: Coordinate system of depth camera according to OpenNI and NiTE framework. [**?**]

another. Every gesture is assigned a class label from 1 to 5 and the mapping of class label to hand gesture is stored in a configuration file named *signs.json*.

Minimum and Maximum Distance Training If the gestures are gesticulated with only one person at a static position in space in front of the camera, then the recognition algorithm would not recognize the same gesture gesticulated by another person or the same person in different position. In order to scale the range of recognition, every gesture is gesticulated in 4 different positions as shown in the plot 3.16 and in all possible combinations that hands are kept wider or narrower as shown in the plots 3.17. Therefore, each gesture in the training data is recorded in 4 positions with each for 20 seconds at 30 samples per second created 2400 samples per gesture.

ANBC is an iterative learning algorithm that improves the classification accuracy with increase in positive training data. Plot 3.16 shows that the trained data makes our gesture recognition system to detect gestures at the minimum distance from 1700 mm to the maximum distance 2500 mm away from the sensor and 800 mm left or right to the sensor. If the user leaves this field of view, the hand tracking algorithm will lose the hand or gesture will fall in the Null Rejection region of the classifier.

Training Data Once all the gestures are recorded, they replayed using CC to find out, if there is any false samples are added to the training data. Such false data leads to an incorrect model that will ultimately affect the prediction performance. Such samples are removed from the training data and a final dataset with all 5 classes are stored as *hri-training-dataset.txt*. Additionally, some test data for each gesture is recorded in order

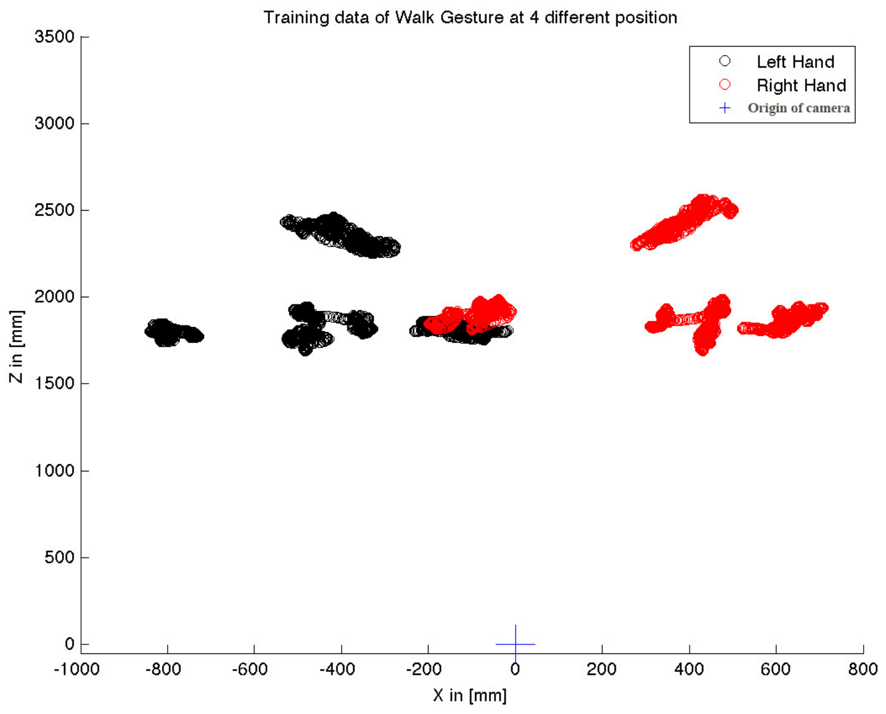


Figure 3.16: Training data of walk gesture recorded in 4 different positions with the minimum distance from 1700 mm to the maximum distance 2500 mm away from the sensor and 800 mm left or right to the origin of the depth camera.

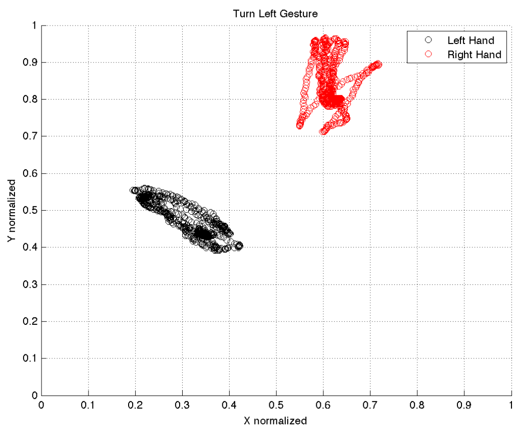
to evaluate the accuracy of the recognition system. Furthermore, a set of non-gesture dataset is recorded to test the Null Rejection accuracy of the classifier.

3.2.3 Prediction

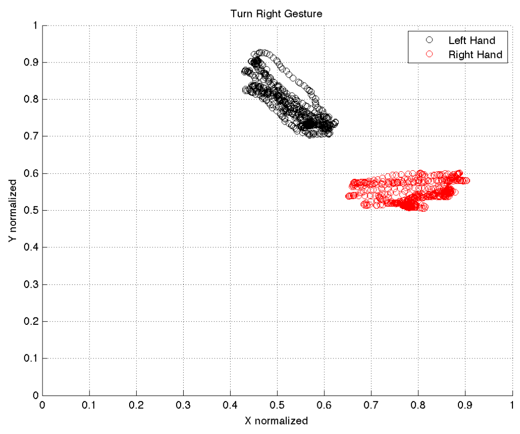
After successfully collecting the training data for all the gestures, Brain is set to prediction mode where the pipeline is trained. HRI module starts to track the user's hand, Brain predicts a gesture when both the hands are present in the input sample. Figure 3.18 shows Control Center where prediction output for every sample with maximum likelihood is displayed all the time. The predicted gesture is triggered only after it is gesticulated for more than one second.

3.3 Human-Robot Interaction

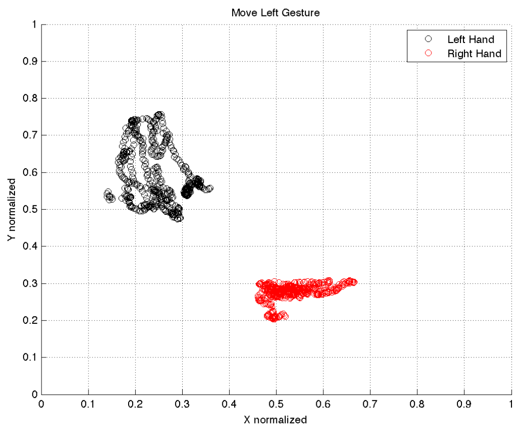
Fundamental goal of this thesis to build a systematic hand gesture recognition system to interact with machines such as robot or a computer. Interaction with them are mostly



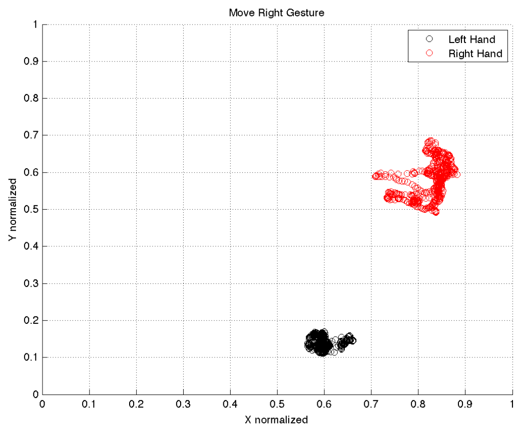
Turn Left Gesture



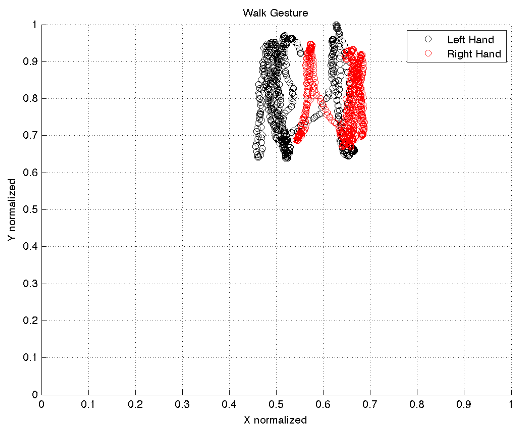
Turn Right Gesture



Move Left Gesture

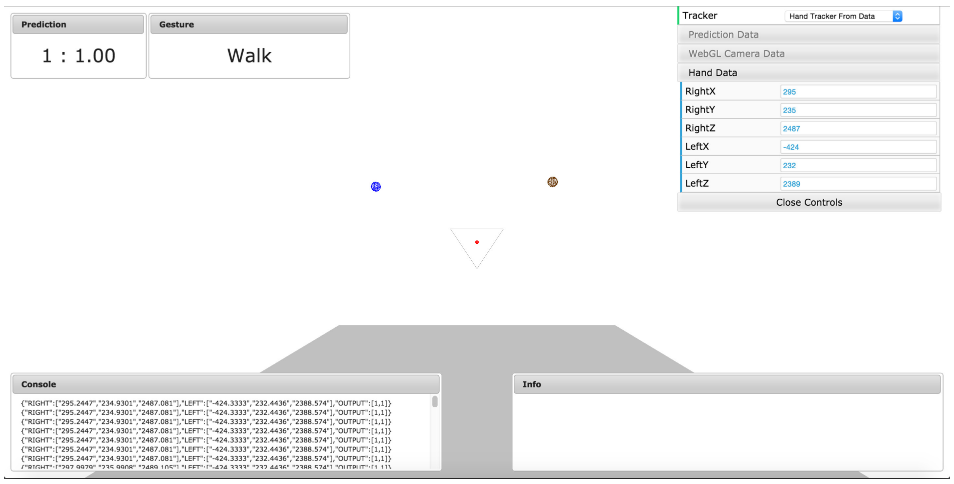


Move Right Gesture



Walk Gesture

Figure 3.17: Normalized training dataset of all 5 gestures are plotted in x and y axis to show that the position of hands are moved during the recording time to get more variations of the same gesture.



Prediction	Gesture
1 : 1.00	Walk

Tracker: Hand Tracker From Data

RightX	295
RightY	235
RightZ	2487
LeftX	-424
LeftY	232
LeftZ	2389

```
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["297.9979","235.9908","2489.105"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
```

Figure 3.18: Control Center displays the recognized walk gesture in real time with the positions of left and right hand in 3 dimensional space.

through displays, keyboards, mouse and touch interfaces. These devices have grown to be familiar but inherently limit the speed and naturalness. Previous sections have explained how we have built a system to facilitate a natural interaction with the humanoid robot called NAO. Following sections illustrate how robot reacts to the hand gestures in real time with the help of Command module.

3.3.1 Gesture-to-Speech

Easiest translation from the recognized hand gesture is to speak it out loud. We have used Text-To-Speech (TTS) engine that is built internally inside Aldebaran modules. When the user gesticulate the focus gesture, NAO says "WAVE" and denoting that hand tracking is started. Furthermore, the robot says words such as "Walk", "Turn Left", "Turn Right", "Move Left" and "Move Right", whenever those gestures are recognized. Additionally, it says info messages such as "Left Hand is lost", "Right Hand is lost" and "Both hands are lost" to inform the user about the internal status of hand tracking.

3.3.2 Gesture-to-Motion

This thesis is initially conceived as a hand gesture translator just to say the recognized gestures loud. To make this system more useful, Gesture-to-Motion feature is added to the Command module. This functionality helps us to move the robot from one position to another in 2 dimensional space. Therefore each gesture is assigned a locomotion task as follows:

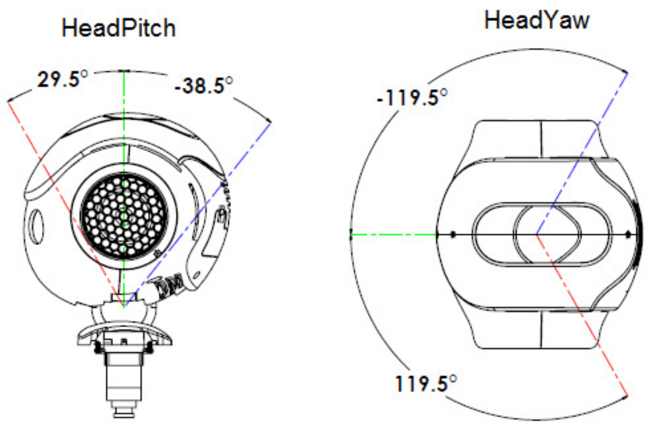


Figure 3.19: NAOs head Pitch and Yaw angle range that can be set with the help of joint control methods of NAOqi API. [?]

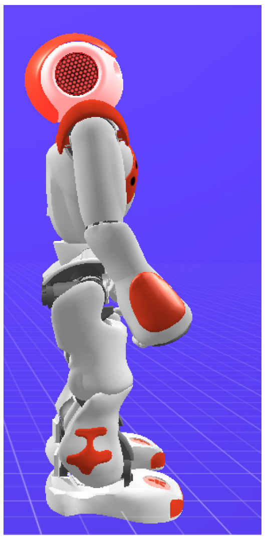


Figure 3.20: Virtual NAO in Aldebaran Choregraphe with head pitch set to -18 degrees look at the upper body of the user.

Walk This gesture commands the robot to walk in forward direction with the given step frequency of 0.5. Robot walks approximately for 5 seconds and waits for the next command.

Turn Left, Turn Right This gesture commands the robot to rotate itself around z-axis in the left/right direction for 3 seconds.

Move Left, Move Right This gestures combines Walk and Turn by commanding the robot to rotate itself around z-axis in the left/right direction for 3 seconds and walk forward for 5 seconds.

Click This gesture is used to gain the control of the robot, when the robot lost its balance and is fallen down. When this gesture is executed, robot wakes up from the sleeping mode and sets itself to the standing position.

Head Position

As described in the section 3.2.2, collection of training data for each gesture is carried out in 4 different positions in front of the robot. During this phase robot is set to standing position where the height of the robot is 58 cm. Figure 3.19 shows that NAOs head can be tilted by adjusting the pitch and yaw of the head joint. In order to avoid confusing camera perspective during the training, NAOs head pitch is set to -18.0 degrees and yaw is set to 0.0 degree as shown in the figure 3.20. At this angle, field of view of the depth camera is enough to cover upper body of the user.

However, keeping the head tilted with mounted camera will cause the robot to lose balance. Therefore, NAOs head position is reset to initial stand position before it executes the received Gesture-to-Motion command. Once the locomotion phase is completed, it looks back at the user. This functionality greatly improves in locating the user at any position in the Minimum-Maximum range as show in the plot 3.17.

3.3.3 Gesture-to-Gesture

Apart from offering the essential functionalities, Command module also provides Gesture-to-Gesture translation where NAO will be imitating hand gestures of the user. Shoulder Roll and Pitch, Shoulder Roll and Yaw angles are measured by manually by positioning NAO for every gesture. When a gesture is detected, the Command modules sets the predefined angles to the shoulder and elbow joints of both the hands of NAO, therefore, translating the human hand gesture to a robotic hand gesture.

3.4 Summary

In this chapter, we have talked about the implementation details of the hand gesture recognition system for human-robot interaction using skeletal points tracking algorithm. Furthermore, we discussed the machine learning techniques which are used to model, train, classify and predict five static hand gestures. Finally, we explained how these trained gesture are used to interact with the humanoid robot NAO.

Chapter 4

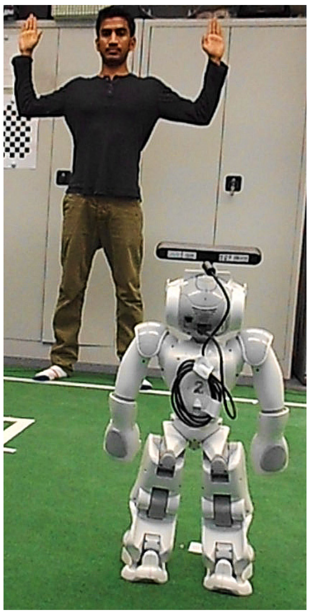
Results

During the training sessions, we have recorded five static gestures in 4 different positions in front of the robot. NAO is equipped with Asus Xtion and set in "Stand" posture. Head pitch of NAO is set to -18 degrees to look at the upper body of the user 1800 mm away from the sensor. First 3 positions of training are recorded at 1800 mm distance from the sensor in z axis and + /- 800 mm in x axis. Last training position is recorded at 2200 mm distance. Therefore, training data is recorded for 80 seconds of each gesture.

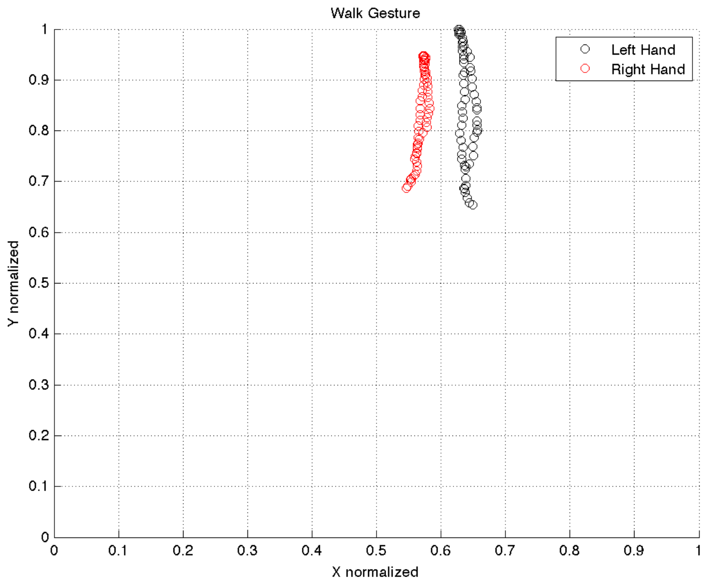
In this chapter, we present the results of real time hand gesture recognition for Human-robot interaction based on skeletal points tracking using depth camera. Training data for 5 classes with 11918 samples of 6 dimensional vector are trained with Adaptive Naive Bayes Classifier. Min-Max scaling and Null Rejection with coefficient of 2.0 are enabled. Following sections illustrates the results of end-to-end interaction with robot using five gestures named as Walk, Turn Right, Turn Left, Move Right, Move Left gestures which are represented by the class labels 1,2,3,4,5 respectively.

4.1 Gesture-To-Motion Results

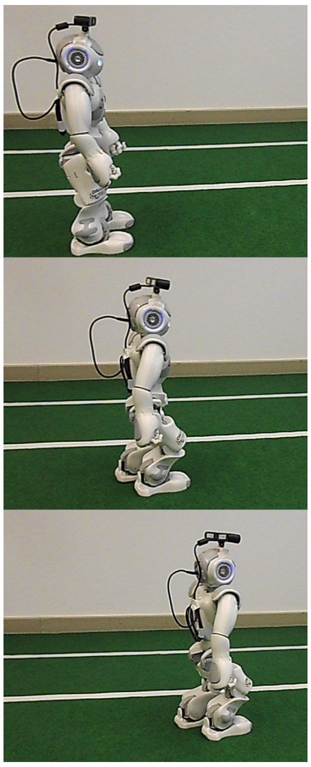
Following sections shows that NAO is looking at the user to detect any possible gestures. When it recognized the gesture, Command module commands the robot to execute the appropriate Gesture-To-Motion task. Additionally, results shows the normalized x, y positions of left and right. It is plotted using 60 input samples of the detected gesture. Finally, Control Center displays the prediction results and detected gesture with x,y,z positions of left and right hand in 3D.

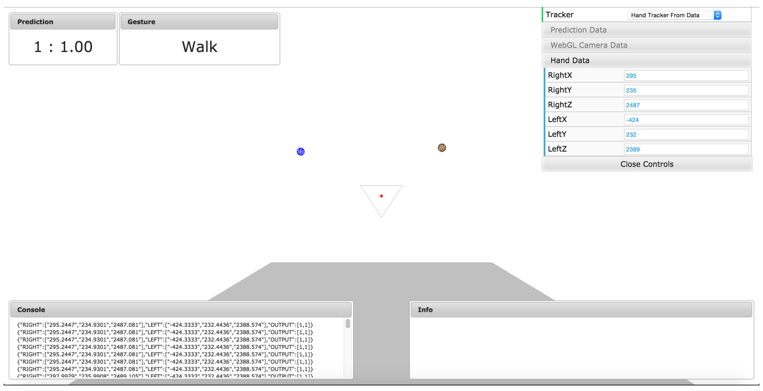


(a)



(b)





Prediction	Gesture
1 : 1.00	Walk

Tracker: Hand Tracker From Data

Prediction Data

WebGL Camera Data

Hand Data

RightX	295
RightY	235
RightZ	2487
LeftX	-424
LeftY	232
LeftZ	2389

Close Controls

Console

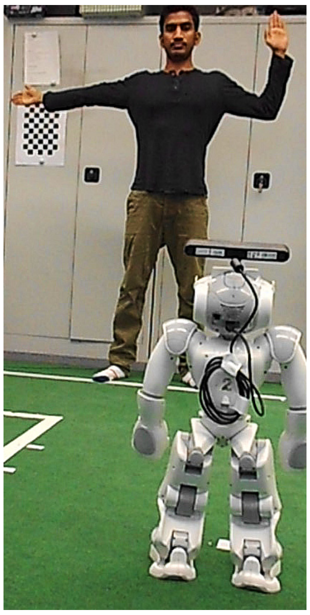
```
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["295.2447","234.9301","2487.081"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
{"RIGHT":["297.9979","235.9908","2489.105"],"LEFT":["-424.3333","232.4436","2388.574"],"OUTPUT":[1,1]}
```

Info

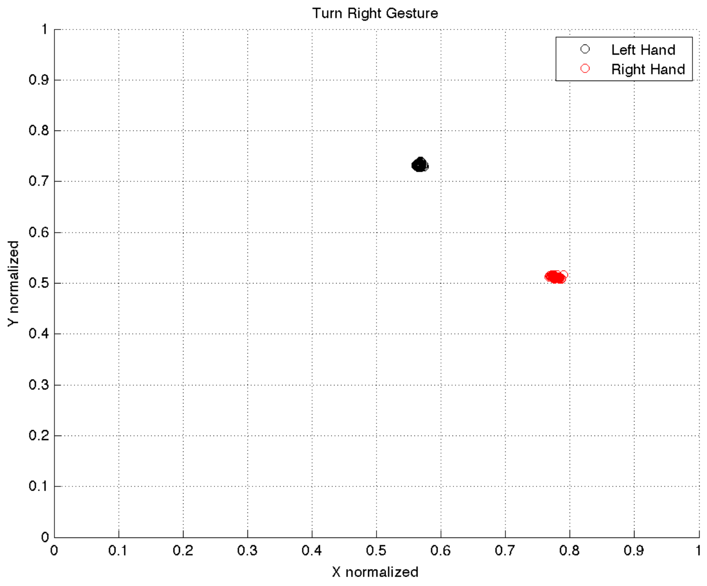
(c)

(d)

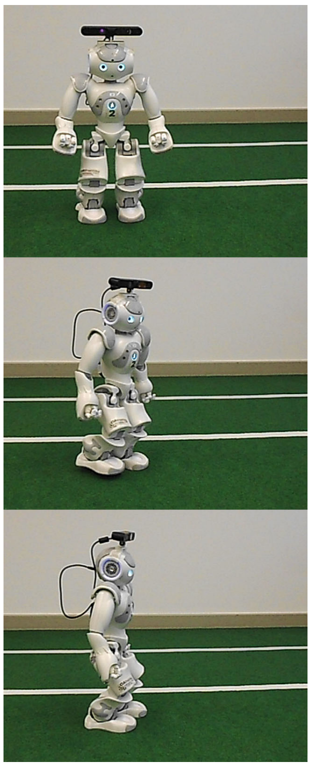
Figure 4.1: (a) User gesticulating Walk gesture. (b) Normalized x,y positions of both hands. (c) CC dashboard shows the prediction results. (d) NAO executes Gesture-To-Motion Task.



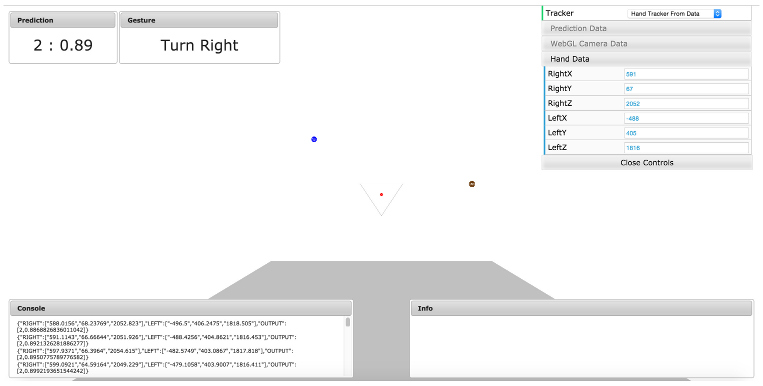
(a)



(b)



(d)



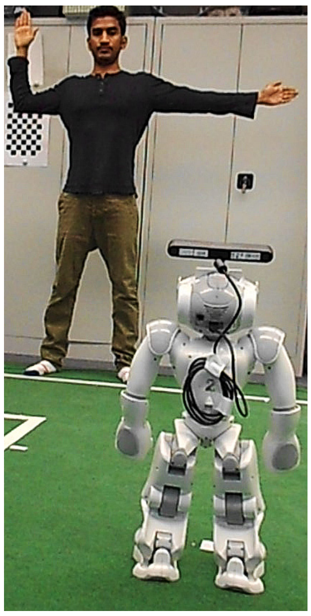
Prediction	Gesture
2 : 0.89	Turn Right

Tracker	Hand Tracker From Data
RightX	591
RightY	67
RightZ	2052
LeftX	-488
LeftY	405
LeftZ	1816

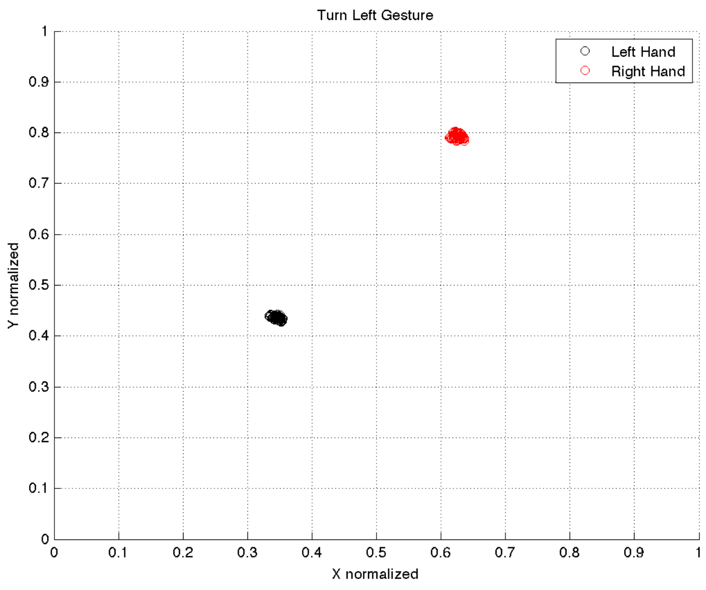
```
{"RIGHT":["588.0156","68.23769","2052.823"],"LEFT":["-496.5","406.2475","1818.505"],"OUTPUT":
[2,0.8868826836011042]}
{"RIGHT":["591.1143","66.66644","2051.926"],"LEFT":["-488.4256","404.8621","1816.453"],"OUTPUT":
[2,0.8921326281886277]}
{"RIGHT":["597.9371","66.3964","2054.615"],"LEFT":["-482.5749","403.0867","1817.818"],"OUTPUT":
[2,0.8950775789776582]}
{"RIGHT":["599.0921","64.59164","2049.229"],"LEFT":["-479.1058","403.9007","1816.411"],"OUTPUT":
[2,0.8992193651544242]}
```

(c)

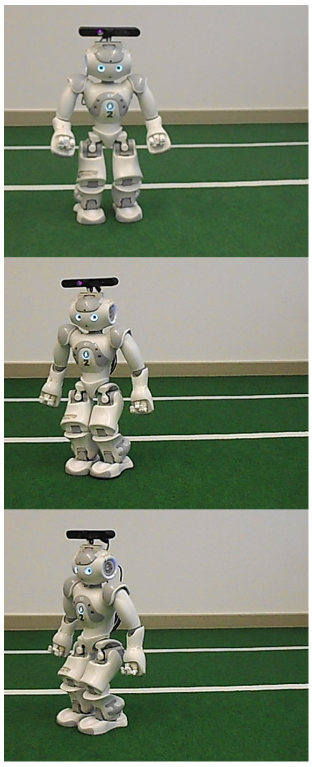
Figure 4.2: (a) User gesticulating Turn Right gesture. (b) Normalized x,y positions of both hands. (c) CC dashboard shows the prediction results. (d) NAO executes Gesture-To-Motion Task.

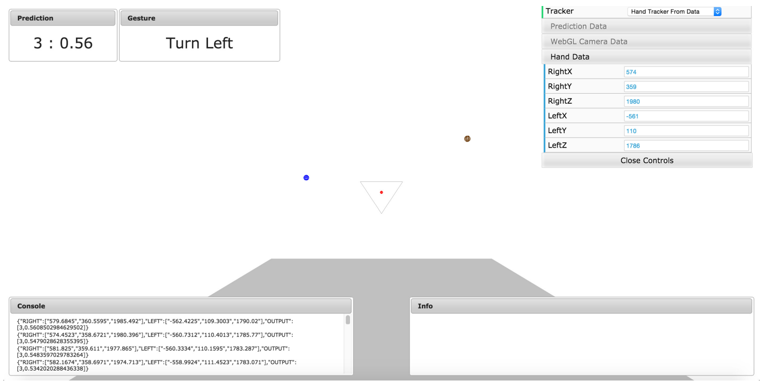


(a)



(b)





Prediction	Gesture
3 : 0.56	Turn Left

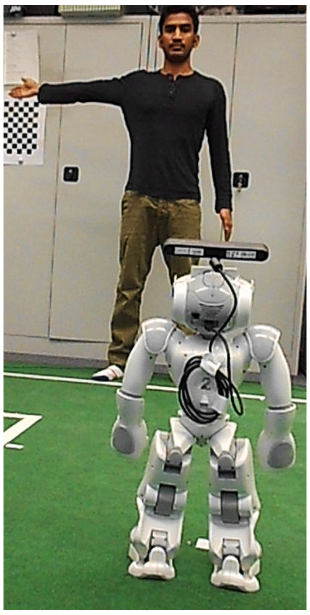
RightX	574
RightY	359
RightZ	1980
LeftX	-561
LeftY	110
LeftZ	1786

```
{"RIGHT":["579.6845","360.5595","1985.492"],"LEFT":["-562.4225","109.3003","1790.02"],"OUTPUT":
[3,0.5608502984629502]}
{"RIGHT":["574.4523","358.6721","1980.396"],"LEFT":["-560.7312","110.4013","1785.77"],"OUTPUT":
[3,0.5479028628355395]}
{"RIGHT":["581.825","359.611","1977.865"],"LEFT":["-560.3334","110.1595","1783.287"],"OUTPUT":
[3,0.5483597029783264]}
{"RIGHT":["582.1674","358.6971","1974.713"],"LEFT":["-558.9924","111.4523","1783.071"],"OUTPUT":
[3,0.5342020288436338]}
```

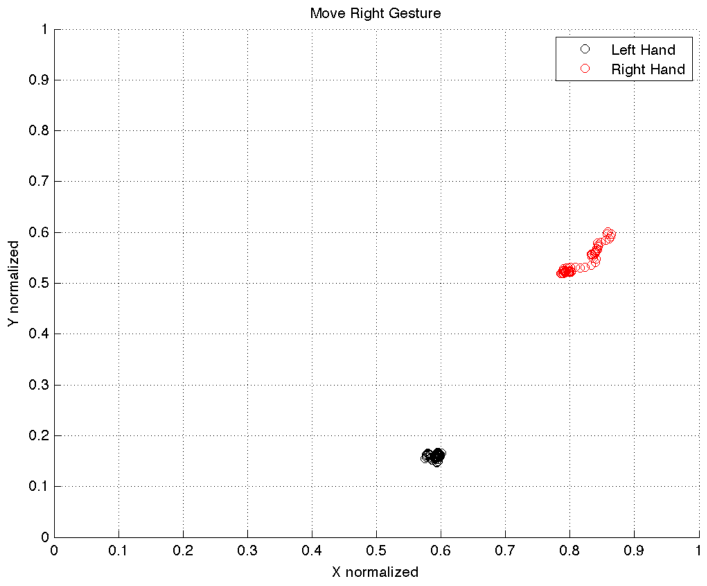
(c)

(d)

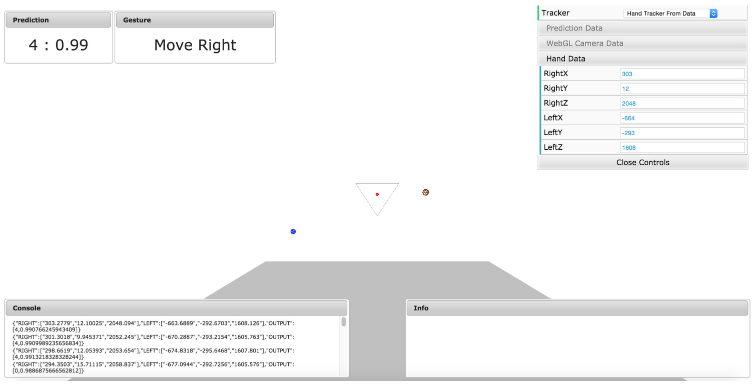
Figure 4.3: (a) User gesticulating Turn Left gesture. (b) Normalized x,y positions of both hands. (c) CC dashboard shows the prediction results. (d) NAO executes Gesture-To-Motion Task.



(a)



(b)



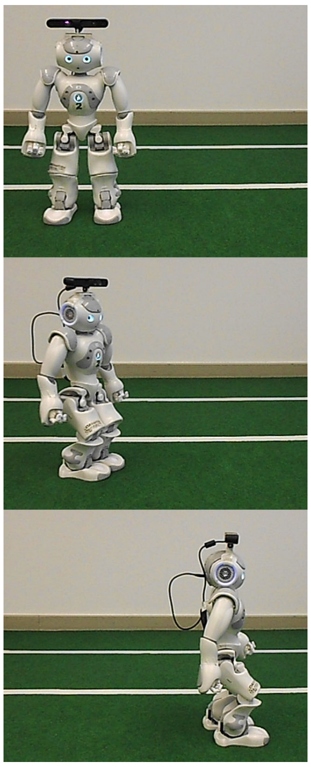
Prediction	Gesture
4 : 0.99	Move Right

Tracker: Hand Tracker From Data

RightX	303
RightY	12
RightZ	2048
LeftX	-664
LeftY	-293
LeftZ	1608

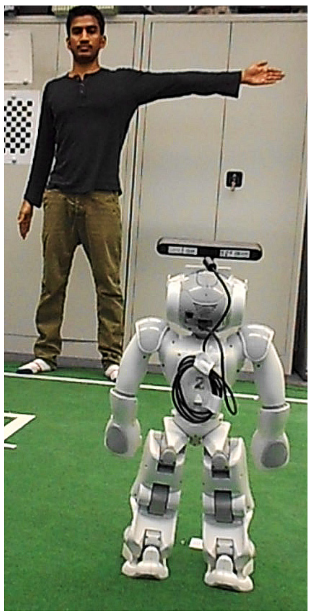
```
{"RIGHT":["303.2779","12.10025","2048.094"],"LEFT":["-663.6889","-292.6703","1608.126"],"OUTPUT":
[4,0.990766245943409]}
{"RIGHT":["301.3018","9.945371","2052.245"],"LEFT":["-670.2887","-293.2154","1605.763"],"OUTPUT":
[4,0.9909989235656834]}
{"RIGHT":["298.6619","12.05393","2053.654"],"LEFT":["-674.8318","-295.6468","1607.801"],"OUTPUT":
[4,0.9913218328328244]}
{"RIGHT":["294.3503","15.71115","2058.837"],"LEFT":["-677.0944","-292.7256","1605.576"],"OUTPUT":
[0,0.9886875666562812]}
```

(c)

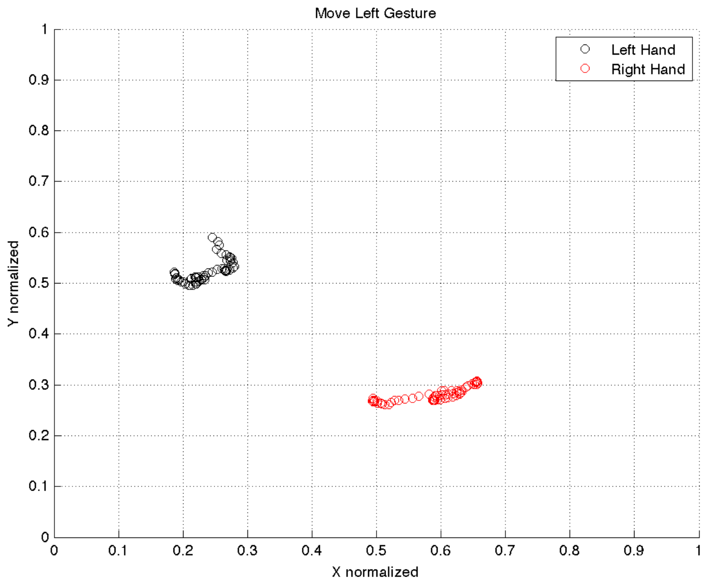


(d)

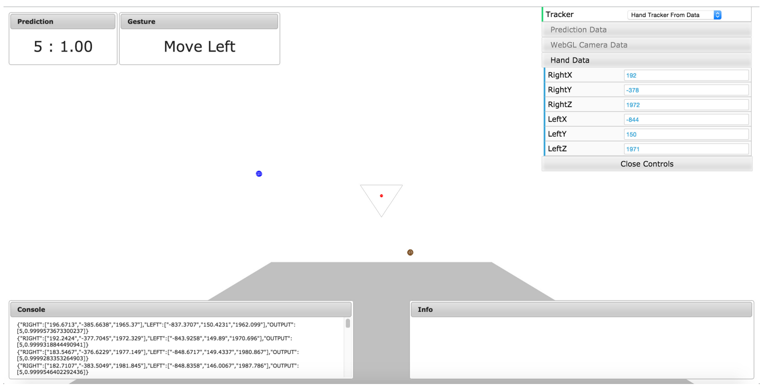
Figure 4.4: (a) User gesticulating Move Right gesture. (b) Normalized x,y positions of both hands. (c) CC dashboard shows the prediction results. (d) NAO executes Gesture-To-Motion Task.



(a)



(b)



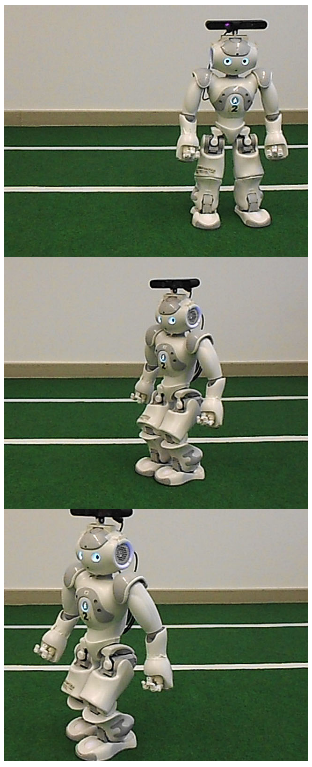
Prediction	Gesture
5 : 1.00	Move Left

Tracker: Hand Tracker From Data

RightX	192
RightY	-378
RightZ	1972
LeftX	-844
LeftY	150
LeftZ	1971

```
{"RIGHT":["196.6713","-385.6638","1965.37"],"LEFT":["-837.3707","150.4231","1962.099"],"OUTPUT":
[5,0.9999573673300237]}
{"RIGHT":["192.2424","-377.7045","1972.329"],"LEFT":["-843.9258","149.89","1970.696"],"OUTPUT":
[5,0.9999318844490941]}
{"RIGHT":["183.5467","-376.6229","1977.149"],"LEFT":["-848.6717","149.4337","1980.867"],"OUTPUT":
[5,0.9999283353264903]}
{"RIGHT":["182.7107","-383.5049","1981.845"],"LEFT":["-848.8358","146.0067","1987.786"],"OUTPUT":
[5,0.9999546402292436]}
```

(c)



(d)

Figure 4.5: (a) User gesticulating Move Left gesture. (b) Normalized x,y positions of both hands. (c) CC dashboard shows the prediction results. (d) NAO executes Gesture-To-Motion Task.

4.2 Gesture-To-Gesture

Figure 4.6 shows how human hand gestures are translated to robotic hand gestures. When a gesture is detected, the Command module sets the predefined angles to the shoulder and elbow joints of both the hands of NAO to perform Gesture-To-Gesture translation.

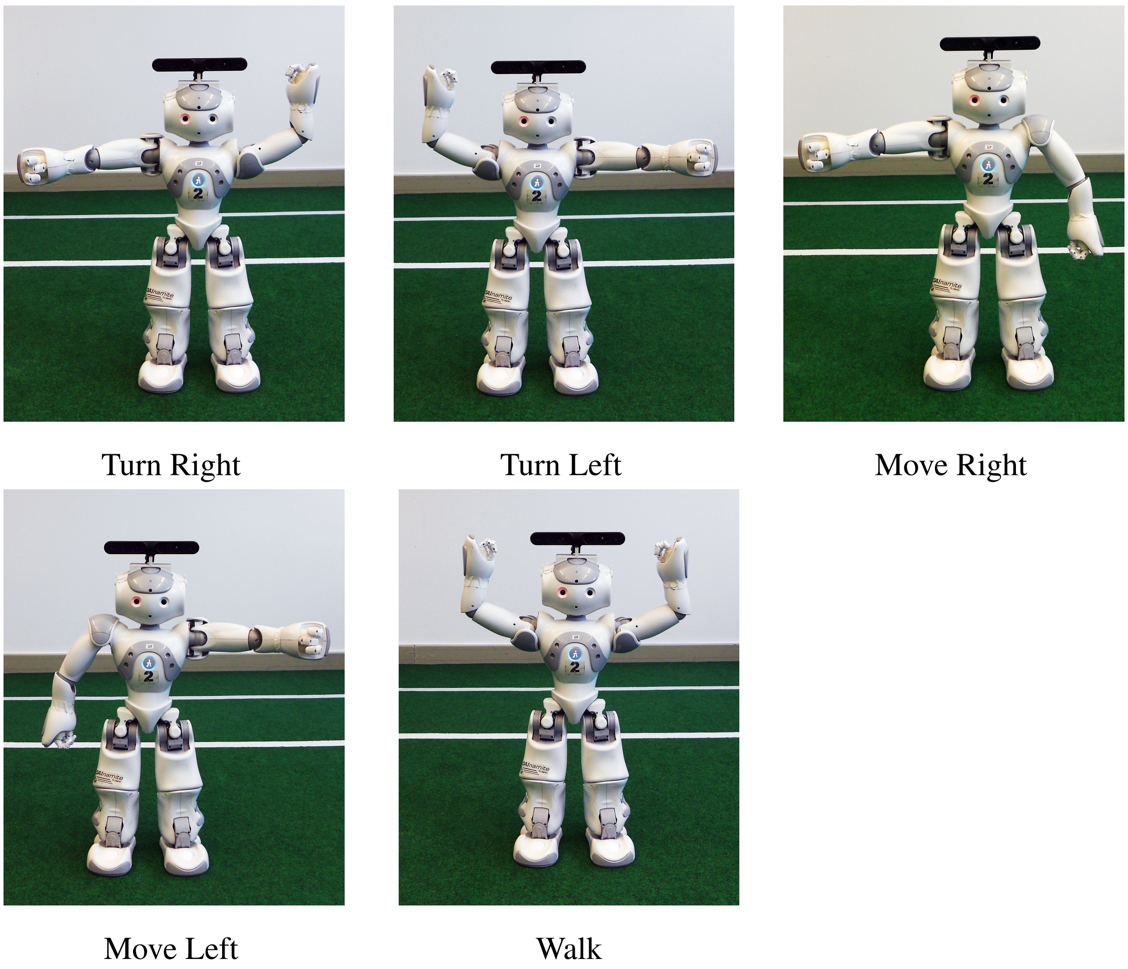


Figure 4.6: Results of Gesture-To-Gesture translation

4.3 Evaluation

In this section, we present the experiments carried out to evaluate and validate our system to recognize hand gestures using skeletal points. The goal is to demonstrate the effectiveness of the classifier and to evaluate its potential for real time prediction. In the classification phase, input samples are normalized using Min-Max Scaling and Null

Rejection is enabled to detect non-gestures. Therefore, the evaluation demonstrates the prediction accuracy of ANBC with various null rejection coefficient and the comparison it with other supervised learning classifier such as Minimum Distance (MinDist).

Class Label	Left X	Left Y	Left Z	Right X	Right Y	Right Z
1	0.55	0.76	0.76	0.57	0.76	0.78
2	0.48	0.73	0.78	0.75	0.51	0.79
3	0.36	0.42	0.78	0.67	0.8	0.8
4	0.58	0.13	0.73	0.79	0.57	0.79
5	0.29	0.52	0.79	0.58	0.24	0.72

Table 4.1: Normalized mean values of 3 dimensions of left and right hand

Class Label	Left X	Left Y	Left Z	Right X	Right Y	Right Z
1.000	0.261	0.116	0.079	0.225	0.092	0.083
2.000	0.189	0.086	0.075	0.149	0.053	0.080
3.000	0.178	0.072	0.088	0.141	0.079	0.093
4.000	0.182	0.060	0.076	0.159	0.070	0.089
5.000	0.128	0.102	0.088	0.114	0.061	0.083

Table 4.2: Standard deviations of 3 dimensions of left and right hand

4.3.1 Mean and Standard Deviation

During the training phase, first all the input samples are normalized with the range from 0 to 1 and then GRT computes mean μ and standard deviation σ to create a model for each class. During the prediction phase, it basically computes the maximum a posterior probability of an input vector belonging to any of the trained class. Figure 4.7 shows the mean positions of left and right hand for every gesture. Table 4.1 and 4.2 show mean and standard deviations of the labeled training data of all the five classes.

4.3.2 Classification and Prediction

Our gesture recognition pipeline is trained with 11918 input samples of 6 dimensional vector for 5 classes. Classes are labeled as 1,2,3,4,5 and they represent Walk, Turn Right, Turn Left, Move Right, Move Left gestures respectively. Class label 0 is reserved for non-gesture by GRT. We have carried out experiments to evaluate the classification, prediction and post processing efficiency of our system. Figure 4.8 show change is positions of left and hand in Cartesian coordinates while test data is recorded and corresponding prediction for every input sample.

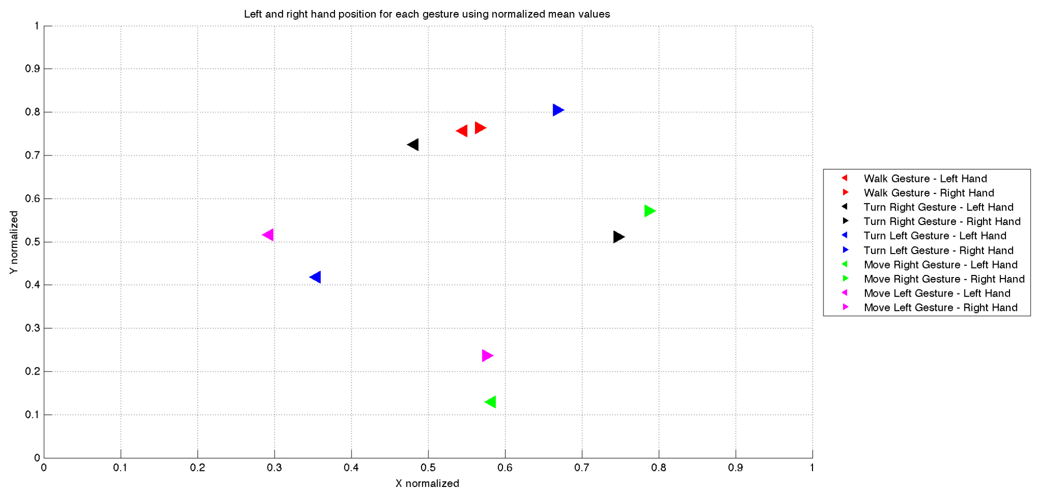


Figure 4.7: Left and right hand position for each gesture using normalized mean values

	Class 1	Class 2	Class 3	Class 4	Class 5
Precision	0.990	0.969	0.996	1.000	1.000
Recall	0.937	0.949	0.939	0.972	0.958
F-measure	0.963	0.959	0.967	0.986	0.978

Table 4.3: Precision, Recall and F-Measure calculated by validating 10% of training dataset. ANBC Classifier trained with Null Rejection coefficient 2.0

Test dataset is a 10% of training dataset and it is chosen randomly from class. Test dataset is validated against the remaining training dataset, therefore, Precision, Recall, F-Measure and Confusion Matrix are computed. Table 4.3 and 4.4 show the results of prediction using ANBC with null rejection coefficient 2.0. Figure 4.8 shows the plot of normalized distances of x,y,z axes of both hands from the test data and their prediction in real time.

	Non-Gesture	Class 1	Class 2	Class 3	Class 4	Class 5
Non-gesture	0.000	0.000	0.000	0.000	0.000	0.000
Class 1	0.032	0.937	0.032	0.000	0.000	0.000
Class 2	0.043	0.009	0.949	0.000	0.000	0.000
Class 3	0.061	0.000	0.000	0.939	0.000	0.000
Class 4	0.023	0.000	0.000	0.005	0.972	0.000
Class 5	0.042	0.000	0.000	0.000	0.000	0.958

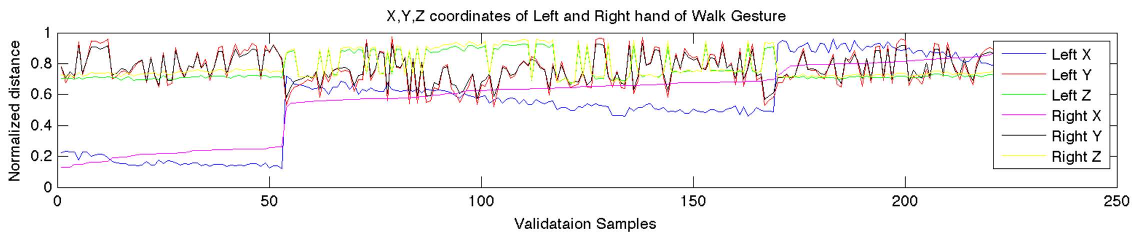
Table 4.4: Confusion Matrix calculated by validating 10% of training dataset. ANBC Classifier trained with Null Rejection coefficient 2.0

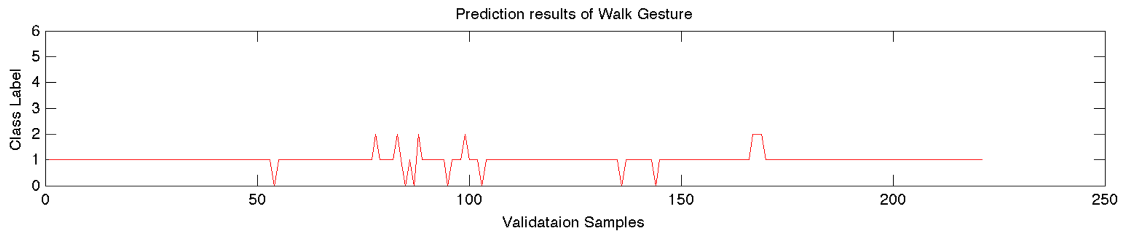
4.3.3 Prediction Accuracy Vs Null Rejection Accuracy

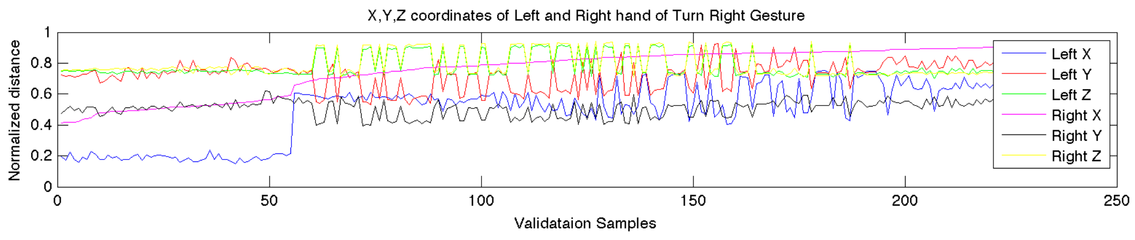
Classifiers of GRT offers various customization that could produce different results for the same test data. Accuracy of a gesture recognition system does not depend only on th precise predictions of trained gestures, but also differentiating them from unintended hand gestures. GRT allows us to set null rejection coefficient for the classifier. Evaluation results are obtained by computing accuracies of 5 trained gestures using ANBC with varying null rejection coefficient from 0 to 10.

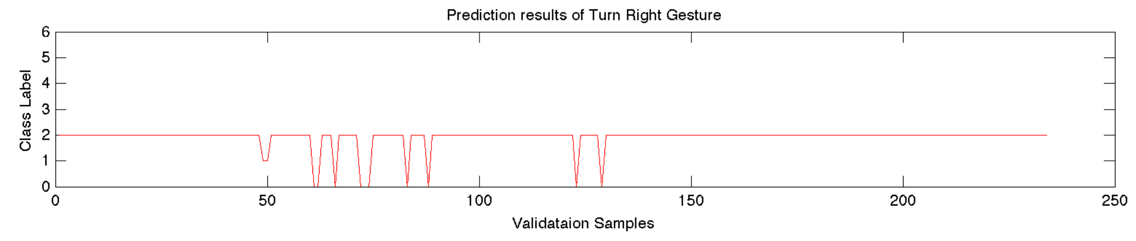
Figure 4.9 shows that increase in null rejection coefficient causes an increase in the accuracy of trained gestures, however, causes a decrease in the accuracy of non-gesture. Therefore, it is optimal to use a null rejection coefficient of 2.0 with ANBC.

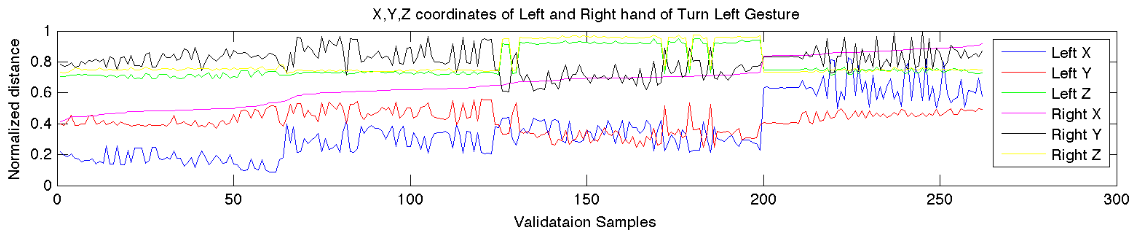
Figure 4.10 shows some interesting results of Minimum Distance classifier with 4 clusters. Figure shows that prediction results are unpredictable as there is increase in null rejection coefficient. However, it produces accuracy above 95% with null rejection coefficient from 2.0 till 4.0. This shows that MinDist could be a better alternative to ANBC.

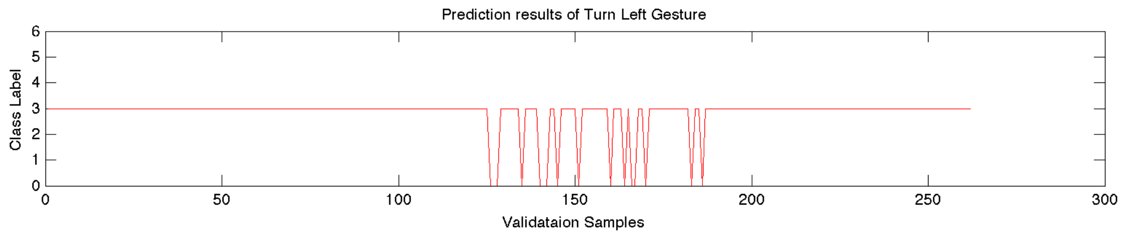


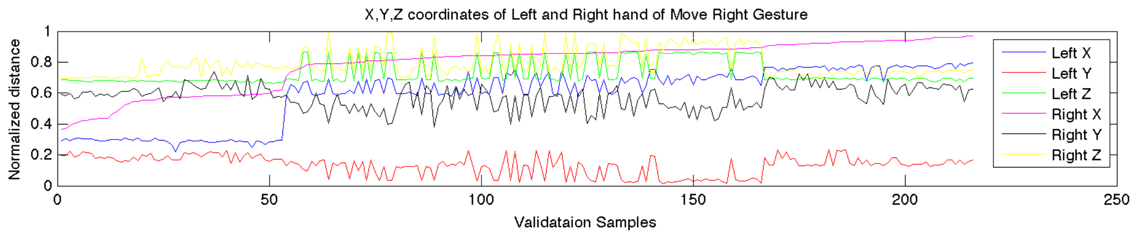


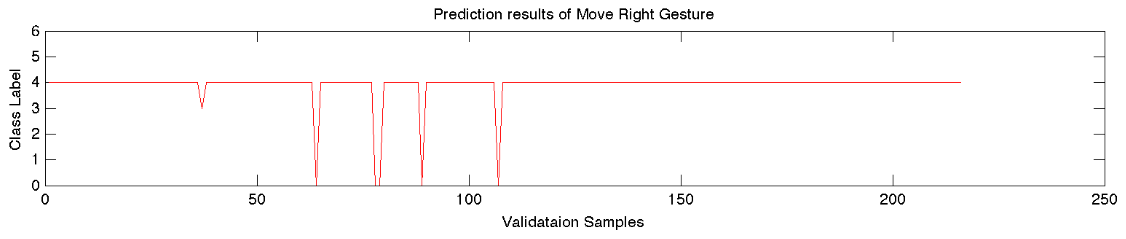


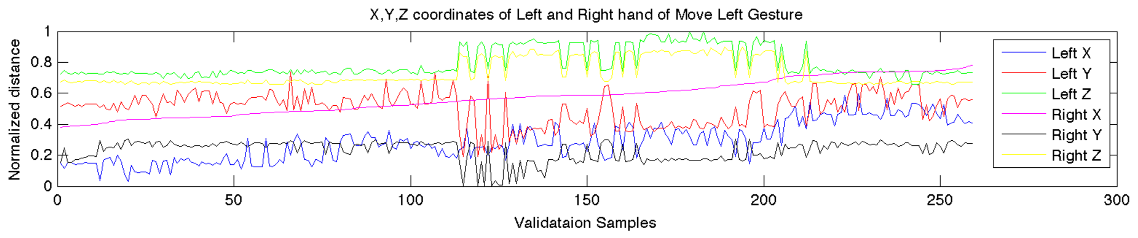

Prediction results of Turn Right Gesture
Class Label
Validataion Samples


X,Y,Z coordinates of Left and Right hand of Turn Left Gesture
Normalized distance
Validataion Samples
Left X
Left Y
Left Z
Right X
Right Y
Right Z


Prediction results of Turn Left Gesture
Class Label
Validataion Samples


X,Y,Z coordinates of Left and Right hand of Move Right Gesture
Normalized distance
Validataion Samples
Left X
Left Y
Left Z
Right X
Right Y
Right Z


Prediction results of Move Right Gesture
Class Label
Validataion Samples


X,Y,Z coordinates of Left and Right hand of Move Left Gesture
Normalized distance
Validataion Samples
Left X
Left Y
Left Z
Right X
Right Y
Right Z

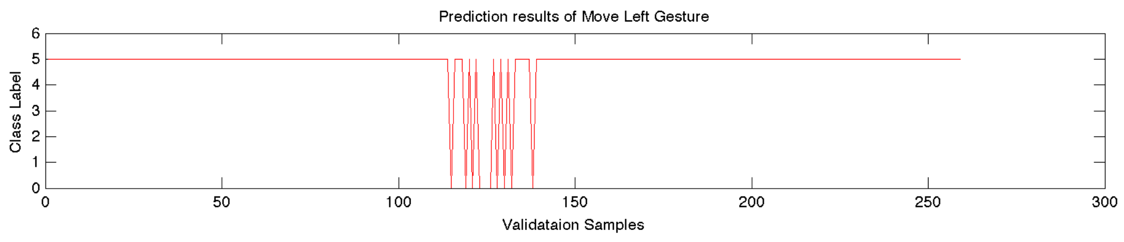


Figure 4.8: Prediction results of test data.

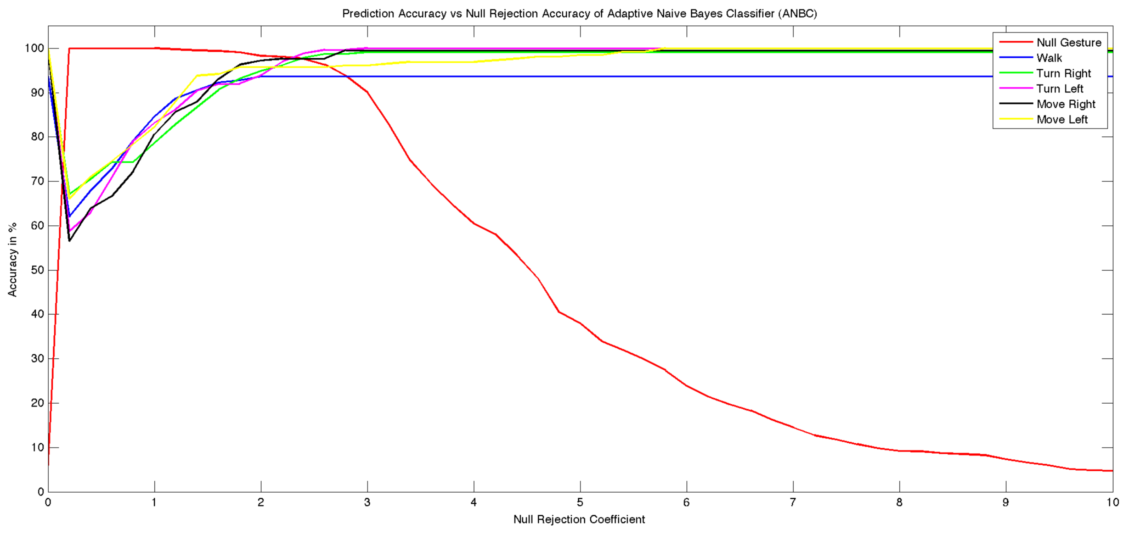


Figure 4.9: Prediction vs Null Rejection of ANBC

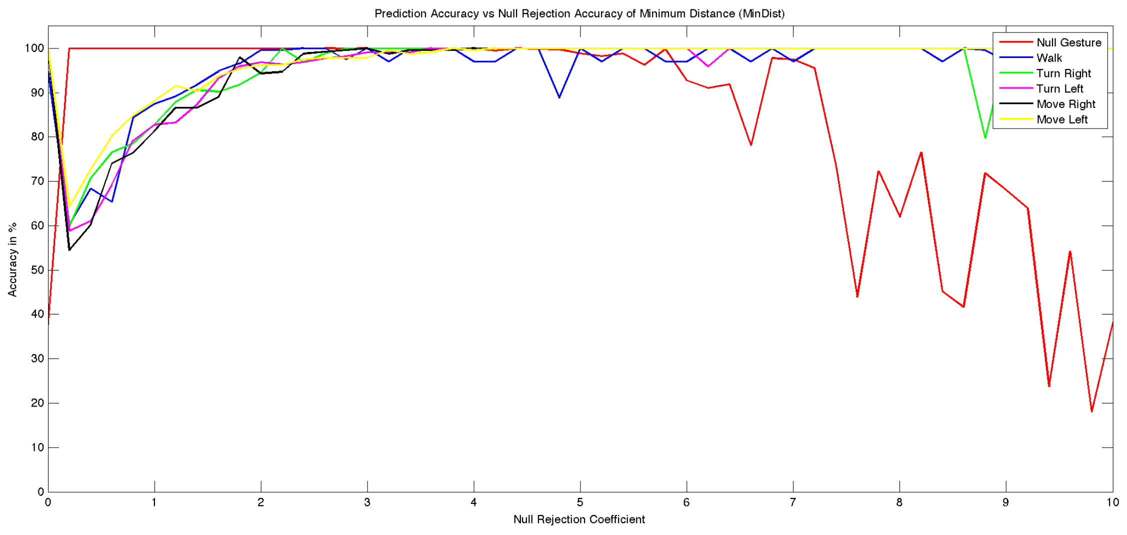


Figure 4.10: Prediction vs Null Rejection of MinDist

Chapter 5

Conclusion and Future work

During this thesis, we have proposed a promising system to recognize hand gestures based on skeletal points tracking using depth camera. This system is built for the purpose of human-robot interaction (HRI) with the humanoid robot named as NAO. We have validated this approach by training the system with five static gestures and obtained results as shown in the chapter 4.

We have partitioned our goal into 4 modules and reached the goal by implementing them in a decoupled environment, and finally, integrated all of them into one system. We have shown that proposed approach is sufficiently robust and flexible to deal with static hand gestures.

Human-Robot Interaction (HRI) module deals with integrating the depth camera into the robot and processing the depth information to track the skeletal points of the user, and finally send them via network to the Brain module. Brain module supports the core functionalities of this system by receiving the skeletal point information of the user, and recording them as training data in the training mode or computing the prediction results in the prediction mode. Control Center (CC) module plays vital role in visualizing these interactions, therefore, it is considered as the eye of the system. Finally, Command module comprehends the predicted gesture and translating them to a robotic Motion or Speech or Gesture itself.

Finally, we have carried out several experiments and provided the results which illustrate the robustness of the system. Furthermore, we have done evaluations on the test data and plotted them on graph to visually understand the performance of the system.

5.1 Discussion

We have faced issues in implementing few contemplated solutions due to various limitations. Following section discusses about few experimental designs which are conceived during thesis.

Everything On-Board

First experiment design is conceived in a way that depth camera, skeletal joint tracking, gesture recognition infrastructure and robot motion will be embedded into the on-board computer of NAO. However, gesture recognition infrastructure is composed of computationally intensive machine learning processes and along with skeletal joint tracking by NiTE had pushed NAO to full CPU load consistently [**?**].

Extending NAO with Single Board Computer

In order to overcome the computational limitation of NAO, another experimental design is contemplated that the robot will be extended as shown in the figure 5.1 with a powerful Single Board Computer such as pcDuino or RaspberryPi. However, Asus Xtions higher power consumption of 2.5 Watts with weight of 250 grams, pcDuinos power consumption of 2A at 5VDC with weight of 100 grams and additional weight by 3D printed mounts, heat sinks and wires will make NAO heavier and ultimately results in poor motion performances and higher power consumption.

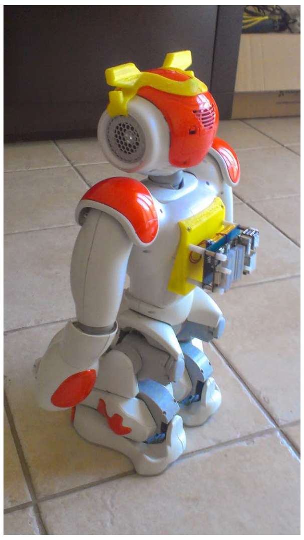


Figure 5.1: 3D printed mount to extend NAO with an external Single Board Computer. [**?**]

Everything Off-Board

This experimental design pushes all the components to an off-board computer that could be a PC connected with depth camera at a fixed location. User will gesticulate in front of the camera and all processing will be done on PC. Finally predicted gesture will be transformed into a motion and voice, and it will be sent to NAO via Aldebaran proxies using WLAN. This design completely decouples the robot from other components and degrades the natural interaction between human and the robot. However, this design suits applications for indoor navigation and localization of NAO [**?**].

Summary

After analyzing the disadvantages of these experimental designs, the final design that was implemented, integrates depth camera into NAO with on-board skeletal tracking application and off-board gesture recognition algorithm. If the computational limitations of NAO are mitigated, Everything On-Board design would be better solution to make NAO completely autonomous while recognizing the hand gestures.

5.2 Future Work

The proposed design for gesture recognition based on skeletal points tracking using depth camera can be improved in several ways. In this section, we overview the future work by discussing the limitations of the proposed methods and proposing the alternatives.

- **Skeleton Joints** : Due to computational limitations of NAO, in this thesis, we have used only hand joints of the user to train and classify the gestures. Classification based on positions of only hand joints does not allow us to train many gestures because the gesture model results in higher rate of confusion. For instance, in Cartesian coordinates "Hands Up" gesture trained at different distances from the sensor will be confused with "Hands Wide". Therefore, we propose to make use of other skeleton joints such as shoulder and arm to calculate the orientation of hand in polar coordinates [**?**].
- **Temporal Gestures** : During this thesis, we have trained our system to recognize only static gestures which imply an action such as traffic police signals. However, human natural interaction is comprised of many more sophisticated dynamic

gestures. Therefore, we propose to extend this system with Dynamic Time Warping classifier of GRT with Time-Series-Classification data to recognize temporal gestures.

- **Computational Limitations of NAO** : In this thesis, HRI module is deployed to general purpose computer of NAO. HRI module is responsible for tracking the hand or full skeleton joints with the help of NiTE framework. NiTE uses computationally intensive algorithms and causes higher CPU utilization of NAO. Therefore, we propose to transmit the depth information from OpenNI device completely to NiTE application on an off-board computer. This could be achieved by using Robot Operating System (ROS) framework that already has a solution to transmit depth information via network.

- **Graphical User Interface** : Since the components of this system are modularized and developed independently, the development environment varies with each other. Even though WebGL is easier and flexible graphics library, it takes a lot of processing power as it runs on the browser. Instead of using several programming languages, the system could be implemented with C++ GUI using QT or Microsoft Visual C++ and OpenGL. We propose to integrate the existing code into such framework to build this system as standalone application.

- **Networking** : 4 modules of this system is connected via several networking components such as UDP Server-Client, Websocket Server-Client and NAOqi proxy. Server-Client networking topology involving different communication protocol is a big limitation, since every module have implemented their own server or client functionalities of UDP or WebSocket. Furthermore, data is serialized as JSON strings and must be parsed at the receiver side to extract the data. Hence, we propose Open Sound Control (OSC) protocol that covers all these requirements in one framework with distributed networking topology.

List of Figures

List of Tables

Abbreviations

HRI	Human-Robot Interaction
OpenNI	Open Natural Interaction
NiTE	Natural Interaction Technology for End-user
GRT	Gesture Recognition Toolkit
ANBC	Adaptive Naive Bayes Classifier
CC	Control Center
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network
FOV	Field Of View
JSON	JavaScript Object Notation
DOF	Degrees Of Freedom
TTS	Text-To-Speech
API	Application Program Interface
DP	Dynamic Programming
MAP	Maximum A Posterior Probability
CSV	Comma Separated Values
DTW	Dynamic Time Warping
HMM	Hidden Markov Models
KNN	K-Nearest Neighbor
SVM	Support Vector Machines
PCA	Principal Component Analysis
GUI	Graphical User Interface
IDE	Integrated Development Environment

Appendix A

Toolchain

During the implementation of this thesis, many tools are used for various purposes. Every module in this thesis uses different programming language, therefore, different toolchains are used. Following sections talk about the tools that are used to develop, build, deploy and document this thesis work.

Develop

Xcode Core functionalities of this thesis are developed in C++ on Mac OSX. Therefore, Xcode was used to develop HRI and Brain module. Xcode is an IDE containing a suite of software development tools developed by Apple for developing software for OS X and iOS.

WebStorm Control Center module was developed in Javascript with the help of a popular IDE for Web development called as WebStorm. It is a commercial IDE for JavaScript, CSS and HTML built on JetBrains IntelliJ IDEA platform.

PyCharm Command module was developed in Python using an IDE name as PyCharm. It is implemented by a company called JetBrains and it provides code analysis, a graphical debugger, an integrated unit tester and supports web development with Django.

Build

Javascript and Python They are traditionally implemented as interpreted languages and therefore they do not need any special compilers to build them. Control Center module needs just a latest browser with WebSocket and WebGL support to run the Javascript code. Python binary is available is most modern operating systems and we used Python version 2.7.6 to run Command module.

C++ The code that was implemented in C++ used 2 different compilers to build it, because the development was done on 64-bit Mac operating system and target system is a 32-bit Gentoo Linux operating system.

Clang and Xcode Development code is built using Clang with LLVM libc++ Standard library. Clang is a compiler developed by Apple for C, C++, Objective-C and Objective-C++ programming languages. Build settings such as header, library search paths, macros, environment variables and linking are configured using Xcode.

GCC and Cmake Production code is built using GNU Compiler Collection (GCC) with libstdc++ GNU++11 Standard library. It is a compiler system produced by the GNU Project supporting various programming languages such as C, C++, Objective-C, Objective-C++, Fortran, Java, Ada, and Go. Build settings such as header, library search paths, macros, environment variables and linking are configured using Cmake. CMake is cross-platform free and open-source software for managing the build process of software using a compiler-independent method.

The repository is cloned on OpenNAO and then HRI module is built as shown in A and then the executable is copied to NAO OS. However, a patch as described in the section A must be applied on OpenNAO and NAO OS to build the sources.

```
# Clone the source repository
git clone git@github.com:AravinthPanch/gesture-recognition
         -for-human-robot-interaction.git ~/hri

# Install dependencies on Mac OSX
xcode-select --install
brew update; brew install git boost cmake

cd ~/hri/source/human-robot-interaction
sudo cp lib/OpenNI2/libOpenNI2.dylib /usr/lib
sudo cp lib/NiTE2/libNiTE2.dylib /usr/lib
sudo cp lib/NiTE2/NiTE.ini /usr/lib
sudo cp -R lib/NiTE2/NiTE2 /usr/lib
```

```
# Install dependencies on 64-bit Ubuntu
sudo apt-get update
sudo apt-get install git build-essential cmake libboost-all-dev

cd ~/hri/source/human-robot-interaction
sudo cp lib/OpenNI2/libOpenNI2.so /usr/lib
sudo cp lib/NiTE2/libNiTE2.so /usr/lib
sudo cp lib/NiTE2/NiTE.ini /usr/lib
sudo cp -R lib/NiTE2/NiTE2 /usr/lib

# On OpenNAO / NAO OS
cd /source/human-robot-interaction
sudo cp lib/NiTE2/libNiTE2-32.so /usr/lib/libNiTE2.so
sudo cp lib/NiTE2/NiTE.ini /usr/lib
sudo cp -R lib/NiTE2/NiTE2 /usr/lib

# Build it on Mac OSX / Ubuntu / OpenNAO
cd ~/hri/source/human-robot-interaction
mkdir build; cd build
cmake ..
make
```

Patch

OpenNAO / NAO OS Aldebaran provides an image of the NAOs operating system named as OpenNAO to use the robotic system virtually and it can run in a virtual machine in any host. OpenNAO is 32-bit Gentoo Linux modified by Aldebaran for Intel Atom Processor with i686 Architecture.

Emerge Emerge is the package manager for Gentoo Linux and Portage is the package tree. Aldebaran forces users not to update Emerge to avoid conflict with Aldebaran modules. Portage tree used with NAO OS is last updated on 11 Jan 2012.

GCC 4.5.3 Due to outdated packages on OpenNAO, GCC version is 4.5.3 and C++ library version is libstdc++.so.6.0.14. NiTE middleware library (libNiTE2.so) was built by PrimeSense using higher version of GCC (higher than GLIBCXX_3.4.14 or libstdc++.so.6.0.14). Therefore, building HRI module on OpenNAO will throw an error while linking libNiTE2.so, */usr/lib/libstdc++.so.6: version GLIBCXX_3.4.15 not found*

This issue was solved by finding higher version of libstdc++ from debian repository for 32-bit architecture and copying that to */usr/lib* folder on OpenNAO/NAO OS and linking current libstdc++ to the copied version. Repository of our thesis contains the required version of libstdc++. Following shell commands show how it must be patched on OpenNAO or NAO OS to run HRI module without any errors.

```
# On OpenNAO/NAO OS, execute the following to apply the patch
cd ~/hri/source/human-robot-interaction
sudo cp lib/libstdc++.so.6.0.16 /usr/lib
sudo rm libstdc++.so
sudo ln -s libstdc++.so.6.0.16 libstdc++.so

# This command should show versions upto GLIBCXX_3.4.16
strings /usr/lib/libstdc++.so.6 | grep GLIBC
```

Version Control

The Proposal till the final results of a project goes through many iterations or modification of the source files. Such changes are intentional and sometimes they are accidental. Therefore, source files of this thesis are stored and tracked using a version control system called Git. Git is a version control is a system that records changes to a file or set of files over time. To avoid accidental lose of source files, free git online service such as GitHub allows us to store them in a remote repository that can be cloned from anywhere via Internet. `https://github.com/AravinthPanch/gesture-recognition-for-human-robot-interaction` is the link to the online repository that contains all the informations regarding this thesis.

Data Analysis

During this thesis work, significant amount (8 MB) of datasets are collected. The datasets contain training data of gestures, prediction results, validation datasets which are recorded to train, test and evaluate the hand gesture recognition system. This information must be analyzed and studied statistically. Hence, MATLAB was used to plot the data as shown in the graph 3.17,