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.-Ing. Yuan Xu

Sivalingam Panchadcharam Aravinth
Matrikelnummer: 342899
Sparrstr. 9
13353 Berlin

Statement of Authorship

I declare that I have used no other sources and aids other than those indicated. All passages quoted from publications or paraphrased from these sources are indicated as such, i.e. cited and/or attributed. This thesis was not submitted in any form for another degree or diploma at any university or other institution of tertiary education

Place, Date Signature

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 an attractive alternative to complex interfaced devices for HRI. In particular, visual interpretation of hand gestures can help in achieving 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. Important differences in the gesture interpretation approaches arise depending on whether 3D based model or appearance based model of the gesture is used [**?**].

In this thesis, we attempt to implement the hand gesture recognition for robots with modeling, training, analyzing and recognizing gestures based on computer vision and machine learning techniques. Additionally, 3D based gesture modeling with skeletal points tracking will be used. As a result, on the one side, gestures will be used command the robot to execute certain actions and on the other side, gestures will be translated and spoken out by the robot.

We further hope to provide a platform to integrate Sign Language Translation to assist people with hearing and speech disabilities. However, further implementations and training data are needed to use this platform as a full fledged Sign Language Translator.

Keywords

Human-Robot Interaction (HRI), Computer Vision, Depth Camera, Hand Gesture, 3D hand based model, Skeleton tracking, Gesture Recognition, Sign Language Translation, Hidden Markov Model, NAO

Acknowledgements

Der Punkt Acknowledgements erlaubt es, persönliche Worte festzuhalten, wie etwa:

- Für die immer freundliche Unterstützung bei der Anfertigung dieser Arbeit danke ich insbesondere...
- Hiermit danke ich den Verfassern dieser Vorlage, für Ihre unendlichen Bemühungen, mich und meine Arbeit zu foerdern.
- Ich widme diese Arbeit

Die Acknowledgements sollte stets mit großer Sorgfalt formuliert werden. Sehr leicht kann hier viel Porzellan zerschlagen werden. Wichtige Punkte sind die vollständige Erwähnung aller wichtigen Helfer sowie das Einhalten der Reihenfolge Ihrer Wichtigkeit. Das Fehlen bzw. die Hintanstellung von Personen drückt einen scharfen Tadel aus (und sollte vermieden werden).

Contents

Chapter 1

Introduction

Huge influence of computers in society has made smart devices, an important part of our lives. Availability and affordability 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 which 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 has still been 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 computer. Usage of robots for domestic and industrial purposes has been continuously increasing. Thus in recent years, there has been a tremendous push in research toward an intuitive and easy communication with the robot through speech, gestures and facial expressions.

Tremendous progress had 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. It may not be able to differentiate between your speech, other people talking and other ambient noise, leading to transcription mix-ups and errors.

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 and hand, or hand gestures. The use of hand gestures provides an attractive alternative for Human-robot interaction than the conventional cumbersome devices.

Chapter 2

Background

2.1 Computer Vision in Robotics

Computer vision is a broad field that includes methods for acquiring, processing, analyzing and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions [**?**].

Proper vision is the utmost importance for the function of any vision based autonomous robot. 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 the hand gesture recognition using computer vision techniques for a humanoid robot named as NAO, as shown in the figure **??**. NAO is an autonomous, programmable humanoid robot developed by Aldebaran Robotics. The NAO Academics Edition was developed for universities and laboratories for research and education purposes. Table **??** shows the specification of NAO according to Aldebaran Robotics.

2.2 NAO - The Humanoid Robot

NAO is an autonomous programmable humanoid robot invented by Aldebaran Robotics. NAO Academics Edition was developed for universities and laboratories for research

and educational purposes. Follow subsections discuss briefly about the specifications of NAO as described by Aldebaran Robotics.

Table 2.1: NAO V5 specification

Height	58 centimetres (23 in)
Weight	4.3 kilograms (9.5 lb)
Battery autonomy	60 minutes (active use), 90 minutes (normal use)
Degrees of freedom	21 to 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 rangefinder 2 x IR emitters and receivers 1 x inertial board 9 x tactile sensors 8 x pressure sensors

2.2.1 Construction

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 NAO with 1.5 or more hours of autonomy, depending on usage. Additional specifications of NAO are shown in the table 2.1. —- Add more info —-

2.2.2 Motion

NAOs walking algorithm uses a simple dynamic model (linear inverse pendulum) and quadratic programming. It is stabilized using feedback from joint sensors. This makes the walking robust and resistant to small disturbances, and torso oscillations in the frontal and lateral planes are absorbed. It can walk on a variety of floor surfaces, such as carpeted, tiled, and wooden floors.

NAOs motion module is based on generalized inverse kinematics, which handles Cartesian coordinates, 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.

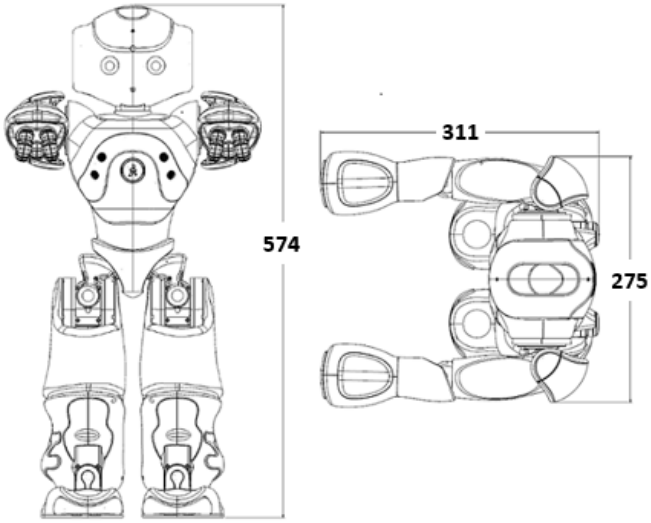

Figure 2.1: Construction of NAO

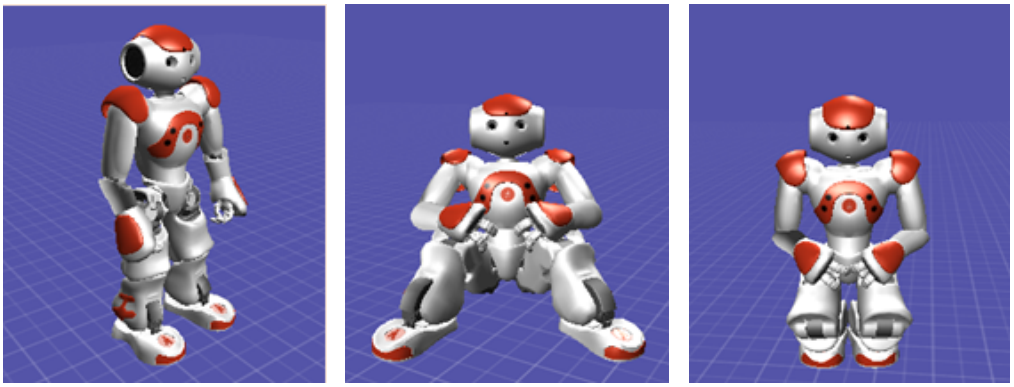

Figure 2.2: Standing, Sitting and Crouching positions of NAO using ALRobotPosture proxy

In this thesis, we used locomotion and stiffness control of Motion API to move NAO to a position in the two dimensional space. Robot Posture API was also used to make the robot go to the predefined posture such as Stand, Sit and Crouch as shown in the figure 2.2. Python code 2.2.2 shows how the robot can be moved to another position at the given normalized velocity using Motion API.

```
# To move the robot at the given normalized velocity using
   Aldebaran Motion and Posture API

import time
from naoqi import ALProxy

robotIP = "nao.local"
```

```
PORT = 9559
motionProxy = ALProxy("ALMotion", robotIP, PORT)
postureProxy = ALProxy("ALRobotPosture", robotIP, PORT)

# Wake up robot
motionProxy.wakeUp()

# Send robot to Pose Init
postureProxy.goToPosture("StandInit", 0.5)

# x - normalized, unitless, velocity along X-axis. +1 and -1
   correspond to the maximum velocity in the forward and
   backward directions, respectively.
# y - normalized, unitless, velocity along Y-axis. +1 and -1
   correspond to the maximum velocity in the left and right
   directions, respectively.
# theta - normalized, unitless, velocity around Z-axis. +1 and
   -1 correspond to the maximum velocity in the
   counterclockwise and clockwise directions, respectively.

x = 1.0
y = 0.0
theta = 0.0
frequency = 1.0
motionProxy.moveToward(x, y, theta, [["Frequency", frequency]])

# Lets make him stop after 3 seconds
time.sleep(3)
motionProxy.stopMove()

# Go to rest position
motionProxy.rest()
```

2.2.3 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.

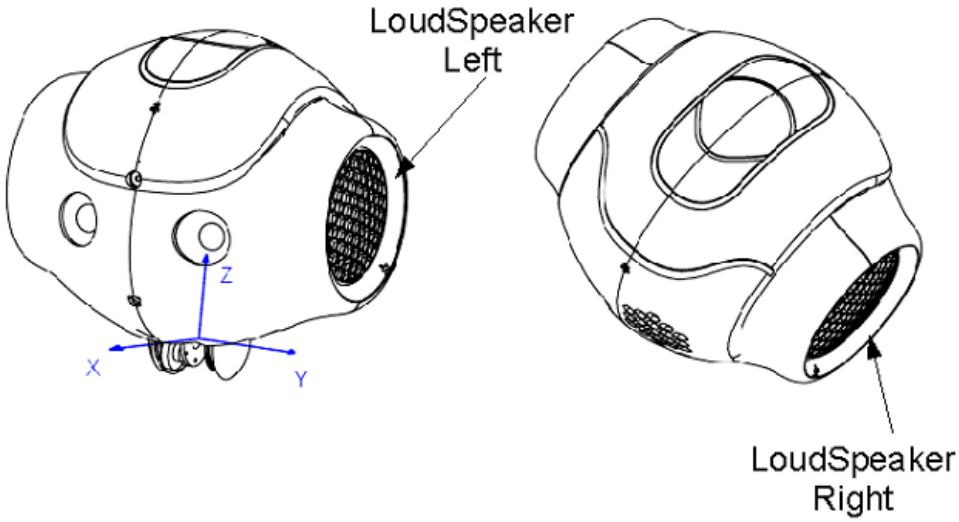

Figure 2.3: NAO Audio

In this thesis, we used Text-To-Speech API of NAO to say some words loud to communicate with the user.Python code 2.2.3 shows how NAO can say words given as strings.

```
# To say the specified string of characters in the specified
   language.

from naoqi import ALProxy

robotIP = "nao.local"
PORT = 9559
tts = ALProxy("ALTextToSpeech", robotIP, PORT)

# Set the language to english
tts.setLanguage("English")

# Say the given word
tts.say("Hello World")
```

2.2.4 Vision

Two identical video RGB cameras are located in the forehead of NAO as shown in the figure 2.4. They provide up to 1280x960 resolution at 30 frames per second. NAO contains a set of algorithms for detecting and recognizing faces and shapes.

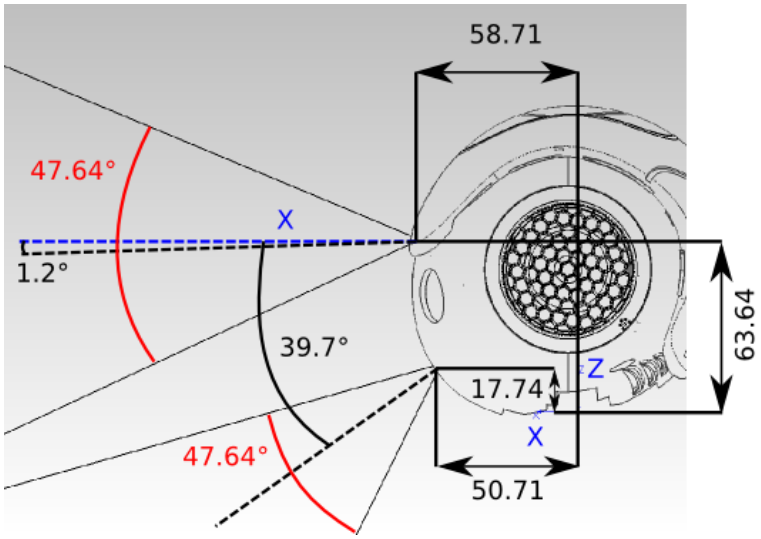

Figure 2.4: NAO Vision

Skeletal points based gesture recognition needs three dimensional data of the human bone joints. However, sensors integrated with NAO could not provide precise three dimensional data for processing heavy 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 [**?**].

Asus Xtion PRO LIVE uses infrared sensors, adaptive depth detection technology, color image sensing and audio stream to capture a user's real-time image, movement, and voice, making user tracking more precise.

Figure 2.5: Asus Xtion Pro Live

Therefore in this thesis, we attempted to use Asus Xtion PRO LIVE as an external camera that was mounted on the head of NAO as shown in the figure 2.5.

Figure 2.6: Depth Image recorded by depth camera Asus Xtion Pro Live

— this section needs revision —

2.2.5 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 Framework is the programming framework used to program Aldebaran robots. This framework allows homogeneous communication between different modules such as motion, audio, video. NAOqi executable which 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.

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

2.3 Gesture Recognition

Human hand gestures are a 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

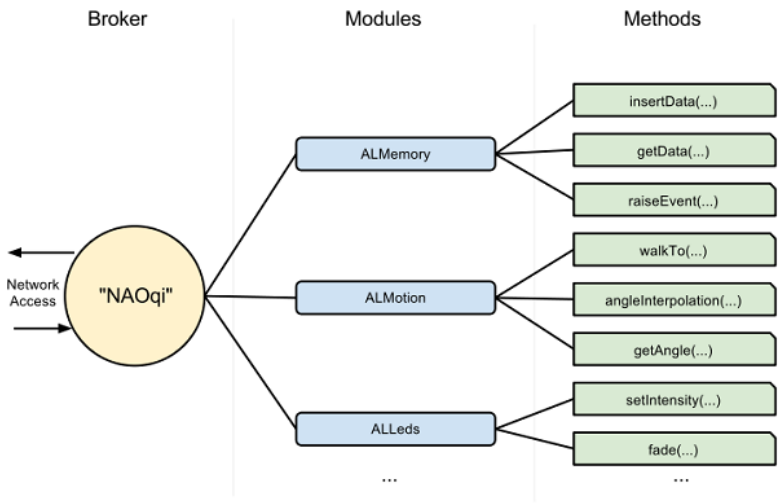

Figure 2.7: NAOqi Proxy

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 were innovated to augment the naturalness of interaction.

2.3.1 Gesture Modeling

Figure 2.8 shows various types of modeling techniques used for Gesture modeling [**?**]. 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.

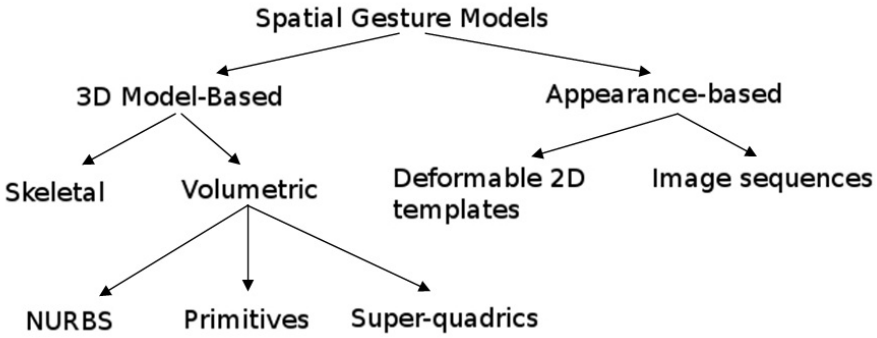

Figure 2.8: Gesture Modeling

Appearance based models don't 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 recognition program has to focus only on the significant parts of the body.

2.3.2 Gestural Taxonomy

Several alternative taxonomies have been suggested that deal with psychological aspects of gestures [**?**]. All hand/arm movements are first classified into two major classes as shown in the figure 2.9.

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.

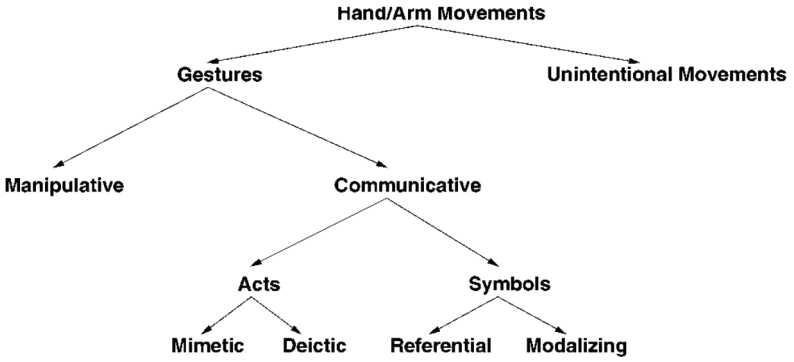

Figure 2.9: Gesture Taxonomy

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 an alphabet "O" or as an object such as wheel or as a command to turn in a circular motion.

2.3.3 Feature Extraction

Feature extraction stage is concerned with the detection of features which are used for the estimation of parameters of the chosen gestural model. In the detection process it is first necessary to localize the user.

2.3.3.1 OpenNI 2

OpenNI or Open Natural Interaction is a framework by the company PrimeSense and open source software project focused on certifying and improving interoperability of natural user interfaces and organic user interfaces for Natural Interaction (NI) devices, applications that 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 IR 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.

C++ code 2.3.3.1 shows how a depth camera such as Asus Xtion Pro can be used to retrieve depth information using OpenNI 2 framework.

```
/**
 * Basic OpenNI setup to read data from depth camera
 */


#include <stdio.h>
#include <OpenNI.h>

#define SAMPLE_READ_WAIT_TIMEOUT 2000
using namespace openni;

int main()
{
  OpenNI::initialize();
  Device depth_camera;
  depth_camera.open(ANY_DEVICE);

  VideoStream depth_stream;
  depth_stream.create(depth_camera, SENSOR_DEPTH);
  depth_stream.start();

  VideoFrameRef depth_frame;

  while (true)
  {
    int changedStreamDummy;
    VideoStream* pStream = &depth_camera;
    OpenNI::waitForAnyStream(&pStream, 1, &changedStreamDummy,
       SAMPLE_READ_WAIT_TIMEOUT);

    depth_camera.readFrame(&depth_frame);
    DepthPixel* pDepth = (DepthPixel*)depth_frame.getData();
    int middleIndex =
       (depth_frame.getHeight()+1)*depth_frame.getWidth()/2;

    printf("%8d\n", pDepth[middleIndex]);
  }
```

```
    depth_camera.stop();
    depth_camera.destroy();
    depth_camera.close();
    OpenNI::shutdown();

    return 0;
}
```

2.3.3.2 NiTE 2

PrimeSense's Natural Interaction Technology for End-user 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 that 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 2.10 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, which is the physical acquisition layer, resulting in raw sensory data from a stream of depth images.
- The next Cshaped layer is executed on the host PC represents OpenNI. OpenNI provides communication interfaces that interact with both the sensor's 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.
- The NITE Controls layer is an applicative layer that provides application framework for gesture identification and gesture-based UI controls, on top of the data that was processed by NITE Algorithms.

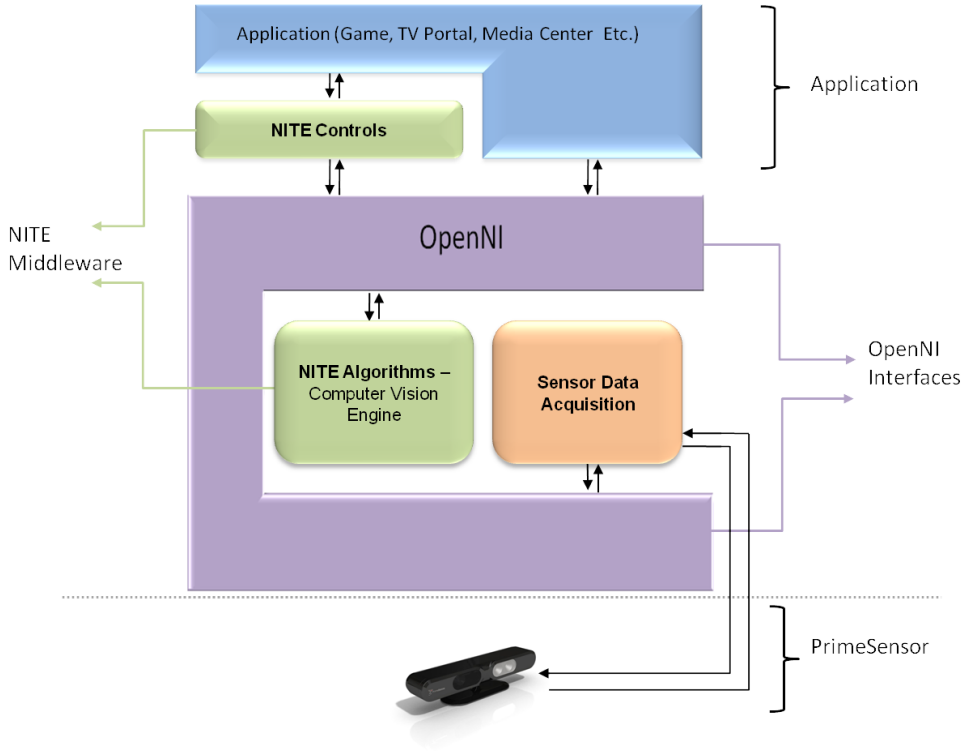

Figure 2.10: NiTE Architecture

2.3.3.3 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 a set of locations of body joints.

NiTE uses machine learning algorithms to recognize anatomical landmarks and pose of human body []. Figure 2.11 shows how NiTE tracks human skeleton from a single input depth image and a per-pixel body part distribution is inferred. 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.

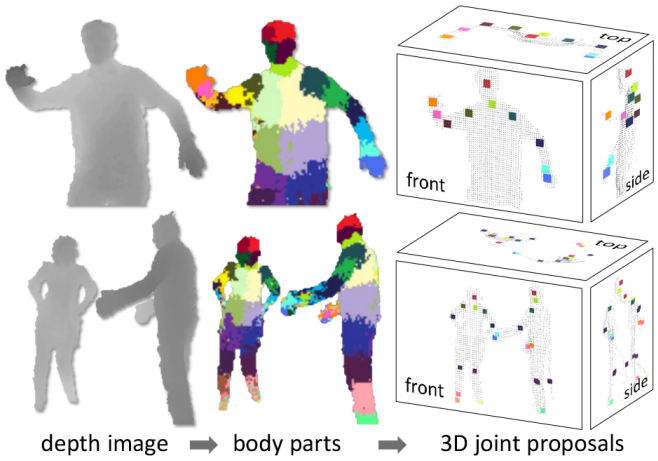

Figure 2.11: NiTE Algorithm to do real-time human pose recognition using depth images

Training In order to train the system, large collection of synthetic and real representations of human body were recorded and labeled. Each body representations was covered with several localized body part labels as show in the figure 2.12. 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.

Figure 2.12: NiTE Synthetic and real training

Feature Labeling Features are located in depth image as shown in the figure 2.13 and labeled. The yellow crosses indicates the pixel x being classified. The red circles indicate the offset pixel. In (a), the two example features give a large depth difference response. In (b), the same two features at new image locations give a much smaller response.

Classification Randomized decision forest is the classification algorithm used by NiTE to predict the probability of a pixel belonging to a body part. Randomized decision trees and forests have proven fast and effective multi-class classifiers for many tasks. Figure 2.14 shows Randomized Decision Forests. A forest is an ensemble of

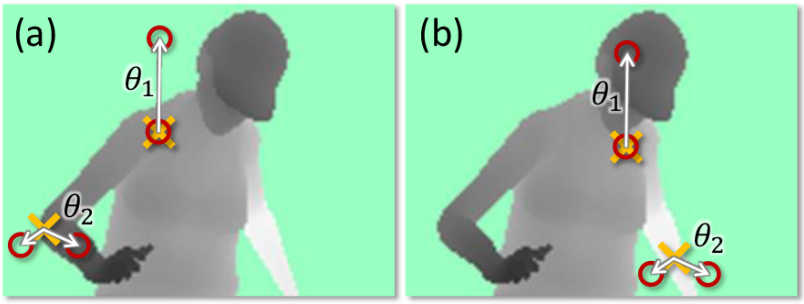

Figure 2.13: Pixels are labeled

trees. Each tree consists of split nodes (blue) and leaf nodes (green). The red arrows indicate the different paths that might be taken by different trees for a particular input.

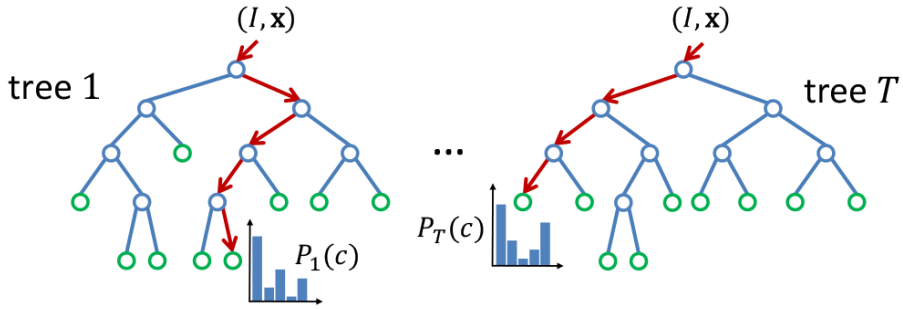

Figure 2.14: Randomized decision forest

Prediction To classify 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 to threshold τ . At the leaf node reached in tree t, 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.

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

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 a roughly even distribution across body parts. Training phase was conducted in distributed manner by training 3 trees from 1 million images on 1000 core cluster.

After predicted 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.

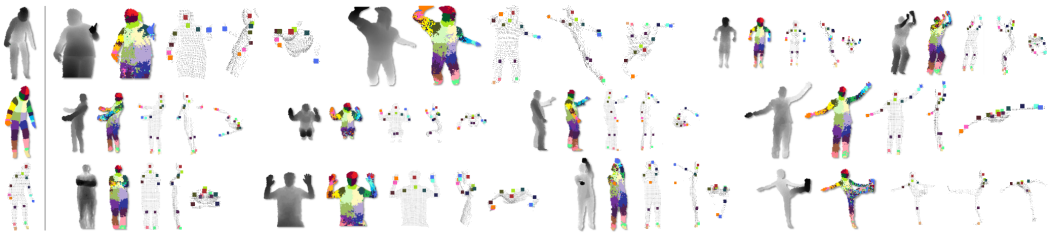

Figure 2.15: Pose Joint Proposal

Joints Proposal Figure 2.15 shows example inferences. Synthetic (top row); real (middle); failure modes (bottom). Left column: ground truth for a neutral pose as a reference. In each example we see the depth image, the inferred most likely body part labels, and the joint proposals show as front, right, and top views (overlaid on a depth point cloud). Only the most confident proposal for each joint above a fixed, shared threshold is shown.

Skeletal points Finally NiTE API returns positions and orientations of the skeleton joints as shown in the figure 2.16. As well as it returns the lengths of the body segments such as the distance between returned 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 of the, +Y points up, and +Z points in the direction of increasing depth.

Hand Tracker Even though NiTE framework can recognize full human body, in this thesis we have used only hand recognition and tracking due to computational limitation of NAO. NiTE provides an interface track only a hand in real time. In order to start tracking a hand, a focus gesture must be gesticulated. There are two supported focus gestures: click and wave. In the click gesture, you should hold your hand up, push your hand towards the sensor, and then immediately pull your hand back towards you. In the wave gesture, you should hold your 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 was touching another object or closer to another body part. Figure 2.17 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.

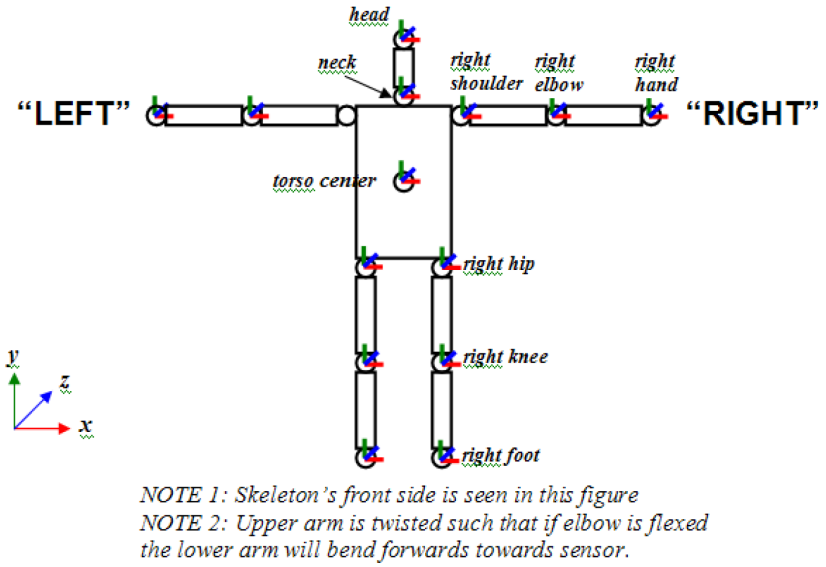

Figure 2.16: NiTE Skeletal Points

Focus gestures Focus gestures of NiTE is can be detected even after initiating the hand tracking. NITE gestures are derived from a stream of hand points which record how a hand moves through space over time. Each hand point is the real-world 3D coordinate of the center of the hand, 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 user to follow these suggestions to gain maximum efficiency from its API.

- Try to keep the hand that performs the gesture at a distance from your body.
- Your palm should be open, fingers pointing up, and face the sensor.
- The movement should not be too slow or too fast.
- WAVE movement should consist of at least 5 horizontal movements (left-right or right-left)
- CLICK movement should be long enough (at least 20 cm).
- Make sure CLICK gesture is performed towards the sensor.
- If you have difficulty to gain focus, try to stand closer to the sensor (around 2m), and make sure that your hand is inside the field of view.

Figure 2.17: NiTE Hand Tracking

Finally, C++ code 2.3.3.3 shows shows how hand tracking can be initiated using a focus gesture.

```
#include <iostream>
#include "NiTE.h"

int main(){
   nite::HandTracker handTracker;
   nite::NiTE::initialize();
   handTracker.create();
   handTracker.startGestureDetection(nite::GESTURE_WAVE);
   handTracker.startGestureDetection(nite::GESTURE_CLICK);
   nite::HandTrackerFrameRef handTrackerFrame;

   while (true)
   {
      handTracker.readFrame(&handTrackerFrame);
      const nite::Array<nite::GestureData>& gestures =
         handTrackerFrame.getGestures();

      for (int i = 0; i < gestures.getSize(); ++i)
      {
         if (gestures[i].isComplete())
         {
            printf ("Gesture Type %d \n",
               gestures[i].getType());
```

```
            nite::HandId newId;
            handTracker.startHandTracking(gestures[i].getCurrentPosition(),
               &newId);
         }
      }

      const nite::Array<nite::HandData>& hands =
         handTrackerFrame.getHands();
      for (int i = 0; i < hands.getSize(); ++i)
      {
         const nite::HandData& hand = hands[i];
         if (hand.isTracking())
         {
            printf ("Hand at x : %f, y : %f, z : %f \n",
               hand.getPosition().x, hand.getPosition().y,
               hand.getPosition().z);
         }
      }
   }

   nite::NiTE::shutdown();

   return 0;
}
```

2.3.4 Gesture Classification and Prediction

Like most other recognitions such as speech recognition and biometrics, the tasks of gesture recognition involve modeling, feature extraction, training, classification and prediction as shown in the figure 2.18. Though the alternatives such as Dynamic Programming (DP) matching algorithms have been attempted, the most successful solutions involves feature-based statistical learning algorithm. Previous sections explained how gesture is modelled and feature is extracted from raw depth images, and the following sections discuss how extracted feature inputs are trained, classified and predicted.

In this thesis, we have chosen a machine learning technique based on Adaptive Naive Bayes Classifier (ANBC) 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.

2.3.4.1 Adaptive Naive Bayes Classifier (ANBC)

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 in the maximum likelihood value that is calculated by calculating Gaussian distribution for each sample.

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 multi-dimensional 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. 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. 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). The MAP estimate classifies x as the kth class that results in the maximum a posterior probability given by the equation 2.3

$$ln\, \mathbb{N}(x|\Phi_k) \quad 1 \leq k \leq G \tag{2.4}$$

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 (i.e. if it is not any of the gestures in the model) then, unfortunately, it will be incorrectly classified against the k th gesture that gives the maximum log-likelihood value. A rejection threshold, k , 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 the Naive Bayes Classifier, 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 the adaptive online training phase the algorithm will not only perform real-time predictions on the continuous stream of input data but also 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 examples after 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 examples increases.

Pros / Cons ANBC algorithm works well for the classification of static gestures and non-temporal pattern recognition. However, The main limitation of the ANBC algo-

rithm is that, because it uses a Gaussian distribution to represent each class, it does not work well when the data you want to classify is not linearly separable. Also when ANBC is working on online training, a small number of incorrectly labelled training examples could create a loose model that becomes less effective at each update step and ultimately lead to a poor performance and accuracy.

2.3.4.2 Gesture Recognition Toolkit (GRT)

Gesture Recognition Toolkit 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 recognitions. Emphasis is placed on 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. This includes algorithms for signal processing, feature extraction and automatic gesture spotting.

In this thesis, we have chose GRT as framework to execute most of the tasks involved in a gesture recognition problem. Figure 2.18 shows that GRT provides the full fledge pipeline to achieve a real-time gesture recognition.

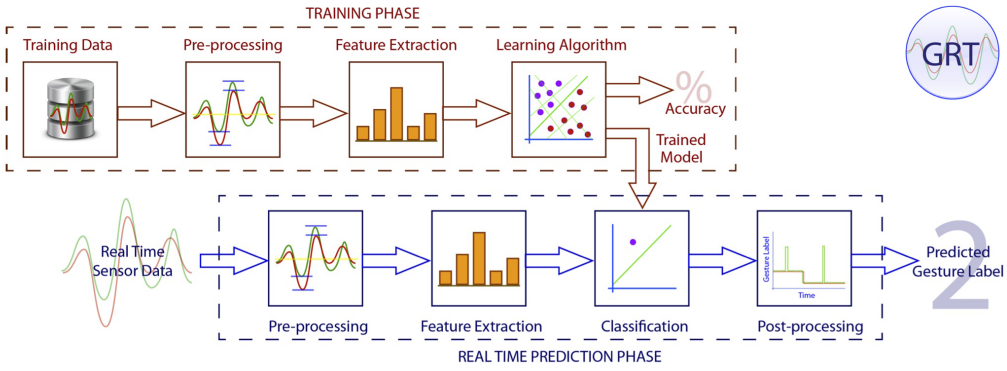

Figure 2.18: Gesture Recognition Pipeline

Pipeline GRT provides an API to reduce the need for repetitive boilerplate code to perform common functionality, such as passing data between algorithms or to preprocess 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 the type of input signal. The algorithms in each module 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. C++ code 2.3.4.2 shows the basic lines of code needed to build a gesture recognition application.

```
#include "GRT.h"
using namespace GRT;

int main (int argc, const char * argv[])
{
    GestureRecognitionPipeline pipeline;
    ANBC anbc;
    ClassificationData trainingData;

    trainingData.loadDatasetFromFile("training-data.txt")
    pipeline.setClassifier(anbc);
    pipeline.train(trainingData);

    VectorDouble inputVector(SAMPLE_DIMENSION) =
        getDataFromSensor();

    pipeline.predict(inputVector);

    UINT predictedClassLabel =
        pipeline.getPredictedClassLabel();
    double maxLikelihood = pipeline.getMaximumLikelihood();
    printf("predictedClassLabel : %d , MaximumLikelihood : %f
        \n", predictedClassLabel, maxLikelihood);

    return EXIT_SUCCESS;
}
```

ClassificationData Accurate labeling of data sets is also critical for building robust machine-learning based systems. The toolkit therefore contains extensive support for recording, labeling and managing supervised and unsupervised data sets for classifica-

tion, regression and time series analysis. ClassificationData is the data structure used for supervised learning problems and for most of the non-temporal classification algorithms. C++ code 2.3.4.2 shows the features of ClassificationData.

```
#include "GRT.h"
using namespace GRT;

int main (int argc, const char * argv[])
{
   ClassificationData trainingData;

   trainingData.setNumDimensions( 3 );
   trainingData.setDatasetName("static-hand-gesture");
   trainingData.setInfoText("Gesture Recognition For
      Human-Robot Interaction");

   UINT gestureLabel = 1;
   VectorDouble sample(3) = getDataFromSensor();
   trainingData.addSample( gestureLabel, sample );

   trainingData.saveDatasetToFile( "hri-training-dataset.txt"
      );
   trainingData.loadDatasetFromFile(
      "hri-training-dataset.txt" );

   ClassificationData testData = trainingData.partition( 80 );

   for(UINT i=0; i<trainingData.getNumSamples(); i++)
   {
      VectorDouble sampleVector = trainingData[i].getSample();
     printf("%f, %f, %f \n", sampleVector[0], sampleVector[1],
        sampleVector[2]);
   }

   trainingData.clear();

   return EXIT_SUCCESS;
}
```

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

TrainingDataRecordingTimer Important part of training is recording positive samples of gestures. Therefore, GRT provides a feature called TrainingDataRecordingTimer that takes recording time 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 that many of the algorithms are implemented with Null Rejections Thresholds. It means that these algorithms can automatically spot the difference between trained gestures and unintended gestures that can happen when the gesticulator 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 Adaptive Naive Bayes Classier and N-Dimensional Dynamic Time Warping, learn rejection thresholds from the training data, which are then used to automatically recognize valid gestures from a continuous stream of real-time data.

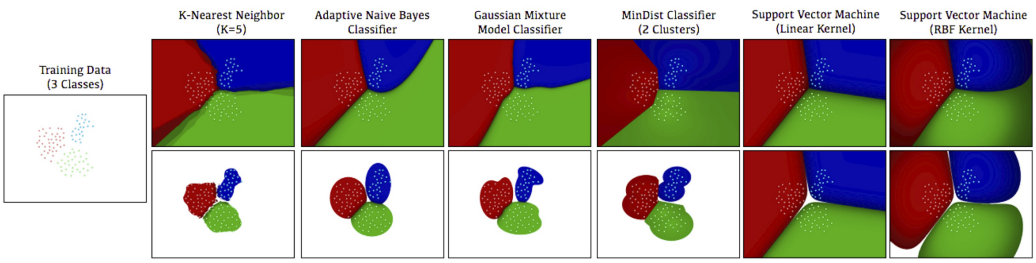

Figure 2.19: GRT Null Rejection

Figure 2.19 above 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 was 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 and a null rejection 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 different 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. Classifier's 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.

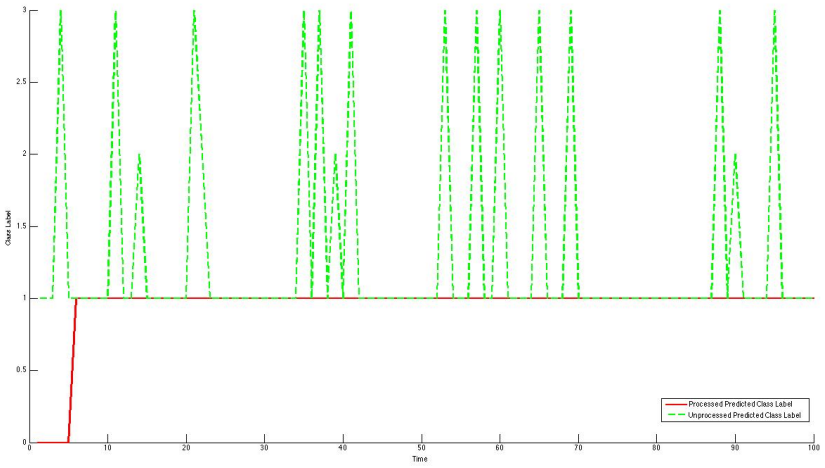

Figure 2.20: GRT Label Filter

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 2.20 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.

Class Label Change Filter It is one of the useful postprocessing module that triggers when the predicted output of a classifier changes. Figure 2.21shows that, if the output stream of a classifier was 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 only once if the user is gesticulating the same gesture for longer time duration.

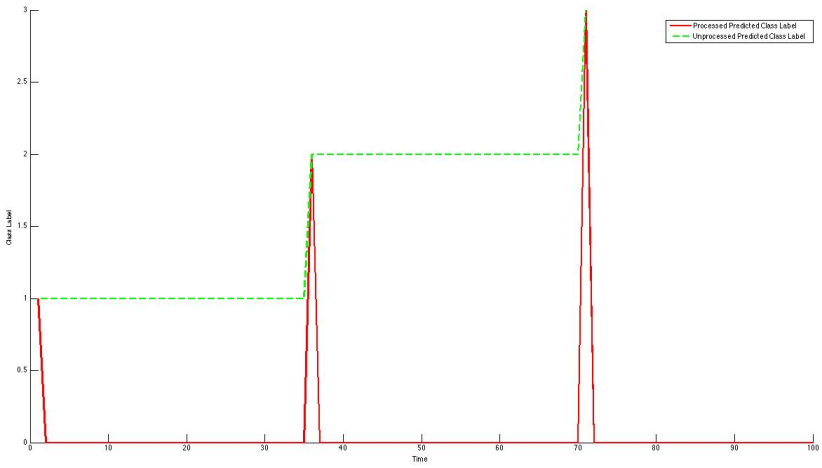

Figure 2.21: GRT Label Change Filter

GUI Figure 2.22 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 timeseries 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, save, load and test the training data and perfoms a real-time prediction for the incoming data, send output to another application.

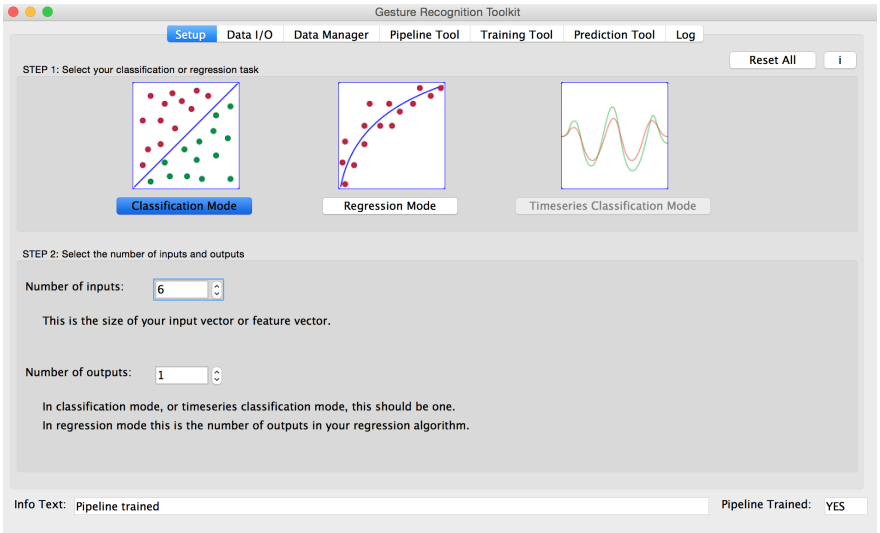

Figure 2.22: GRT GUI

Chapter 3

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.

In this thesis, we attempt to implement the feature for NAO to recognize gestures and execute predefined actions based on the gesture. NAO will be extended with an external depth camera, that will enable NAO to recognize 3D modeled gestures. This 3D camera will be mounted on the head of NAO, so that it can scan for gestures in the horizon. Additionally, skeletal points tracking algorithm with machine learning technique using Hidden Markov Models will be used to recognize the gestures. Due to the computational limitations of NAO, gesture recognition algorithm will be executed on off-board computer. With the hand gesture recognizing feature, NAO will be available to the users in two modes.

- **Command mode:** In this mode, a gesture will be recognized by NAO and related task will be executed. Even though the gesture based interaction is real time, NAO can not be interrupted or stopped by using any gesture while it is executing a task. However, other interfaces such as voice commands can be used in such situation to stop or interrupt the ongoing task execution.

- **Translation mode:** In this mode, NAO will be directly translating the meaning of the gesticulated gestures. To achieve this, text-to-speech library of NAO will be used and recognized gesture can be spoken out using the integrated loudspeaker. In future, it will allow NAO to translate a sign language to assist people with hearing and speech disabilities.

In this thesis, we planned to train NAO with few very simple gestures due to the reason that NAO has computational limitations. Gestures will involve both the hands or single hand to interact with the robot.

Chapter 4

Solution

To build an effective and easy to use hand gesture recognition system for NAO, various tools and technologies were studied during this thesis. Figure 4.1 shows the individual components which are essential parts of this thesis in implementing the goal. The main challenge is to find a solution that can integrate all these components into a robust system. However, due to the computational and compatibility limitations of NAO, we have faced problems in implementing few contemplated solutions which are described in the next section. Finally, the successful solution in achieving the goal will be discussed in the following sections.

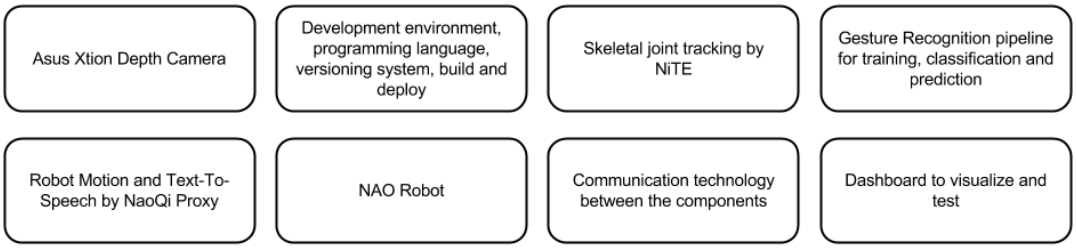

Figure 4.1: Component

4.1 Experimental Designs

4.1.1 Everything On-Board

First experiment design was 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 100 % CPU load consistently. — Show htop of NiTE cpu consumption —-

4.1.2 Extending NAO with Single Board Computer

In order to escape the computational limitation of NAO, another experimental design was contemplated, that the robot will be extended as shown in the figure 4.2 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 to be heavier and ultimately result in poor motion performances and higher power consumption.

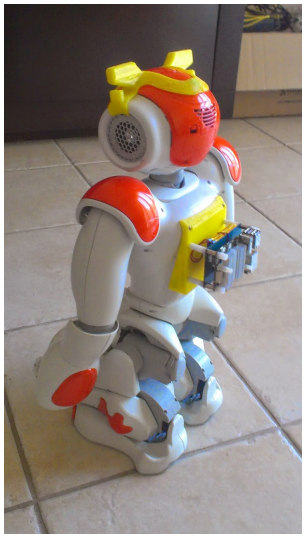

Figure 4.2: NAO Bag

4.1.3 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 will suit for other applications such as indoor navigation and localization of NAO.

4.2 Implementation

After analyzing the disadvantages of other experimental designs, the final design was chosen to build an efficient real-time hand gesture recognition for human-robot interaction using skeletal points. Figure 4.3 shows the architecture of the solution that was implemented during this thesis by grouping many components into 4 different modules which serve various 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 uses a common configuration file that contains information such as port number, host name and log path.

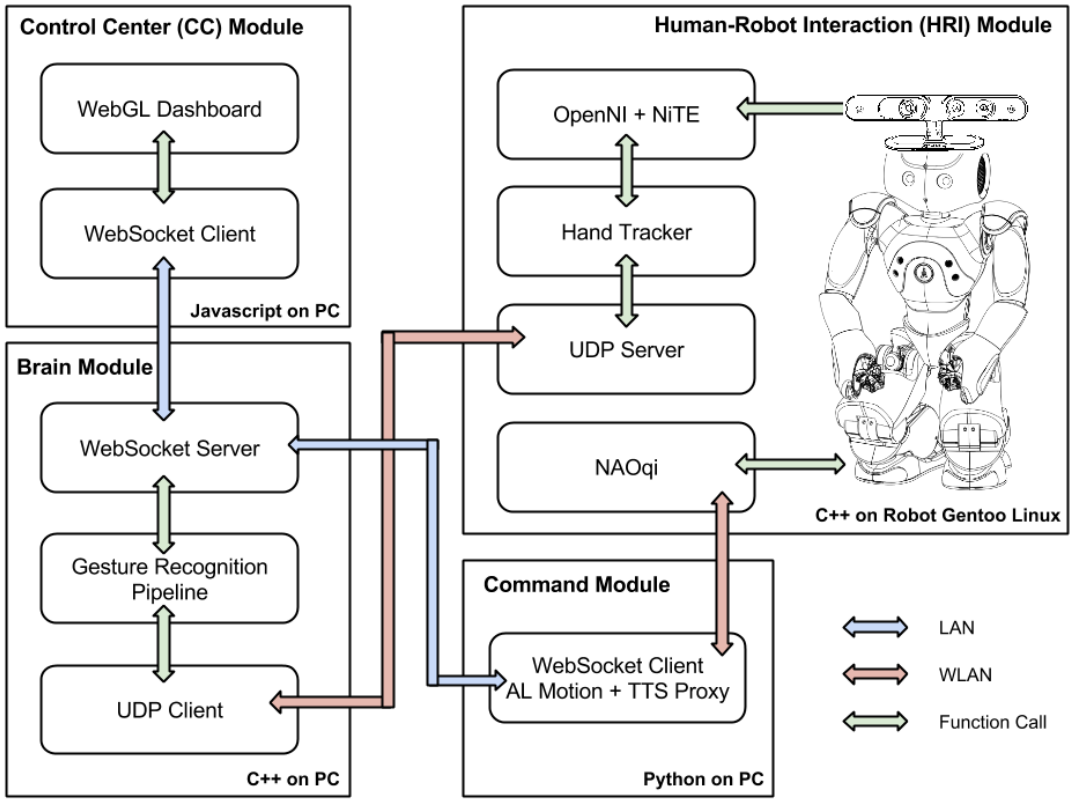

Figure 4.3: HRI Architecture

4.2.1 Human-Robot Interaction (HRI) Module

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

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. Flowchart 4.4 shows the data 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 HRI module.

4.2.1.1 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 4.3, 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 was 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.

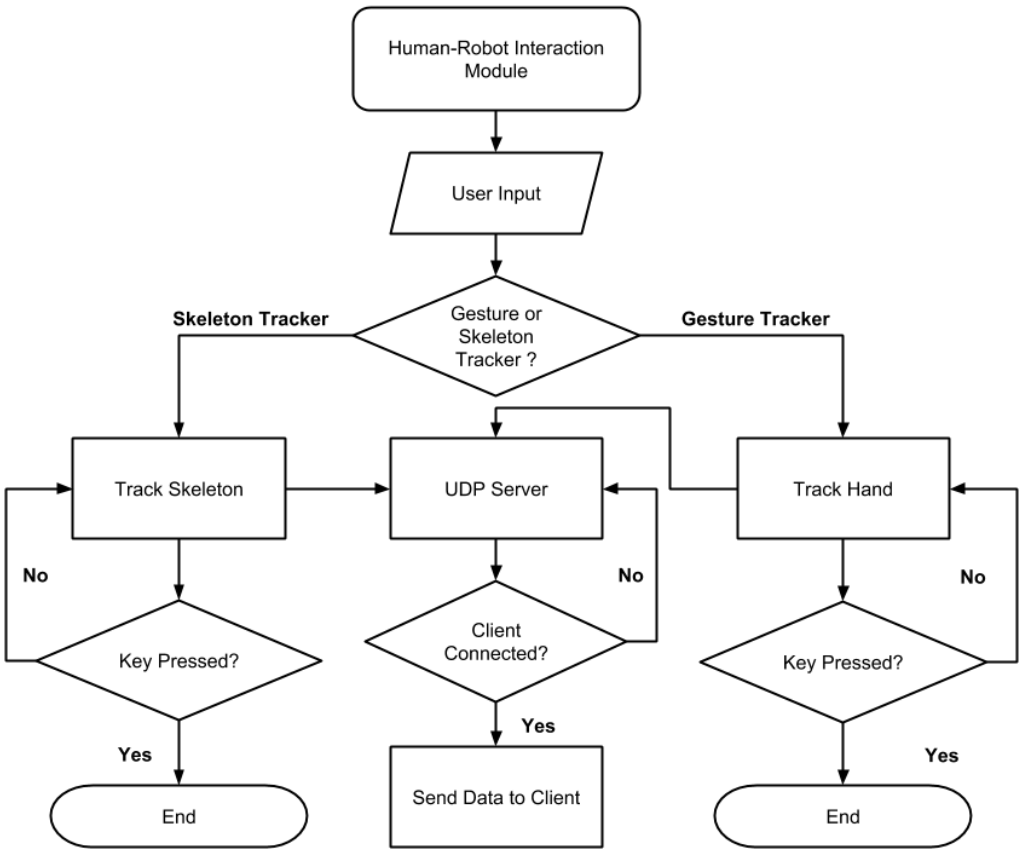

Figure 4.4: HRI Module Control Flow

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 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.

4.2.1.2 Gesture Tracker

Gesture tracker is a component of HRI module that makes use of NiTE framework to localize the hand of 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 2.3.3.2 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 Gestures such as Wave or Click detection must be initiated in order to localize the hand at first.
- nite::HandTrackerFrameRef must be read continuously for a new gesture.
- If Wave or Click 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 (Wave or Click) only in the order of right hand first and left hand second.

Four integer variables leftHand, rightHand, lastLostHand and handsSize are used to find whether tracked or hand is left or right. The logic behind is that when handsSize and lastLostHand is zero and a new hand is found, that is considered as right hand and its nite::HandData::HandId is stored in the variable rightHand. Respectively, the next hand is stored as leftHand and handsSize counter is increased. If a hand is lost or not tracking, then lastLostHand will be updated with the id of the hand that was lost. When there is a new hand and handsSize and lastLostHand are not zero, then new handId will be set to leftHand or rightHand based on lastLostHand variable.

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 4.2.1.2

```
{
  "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 4.2.1.2 to be displayed on the control center dashboard. Info messages helps us to know the status of computationally intensive 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"}
```

4.2.1.3 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 4.2.1.2, 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 4.5 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.

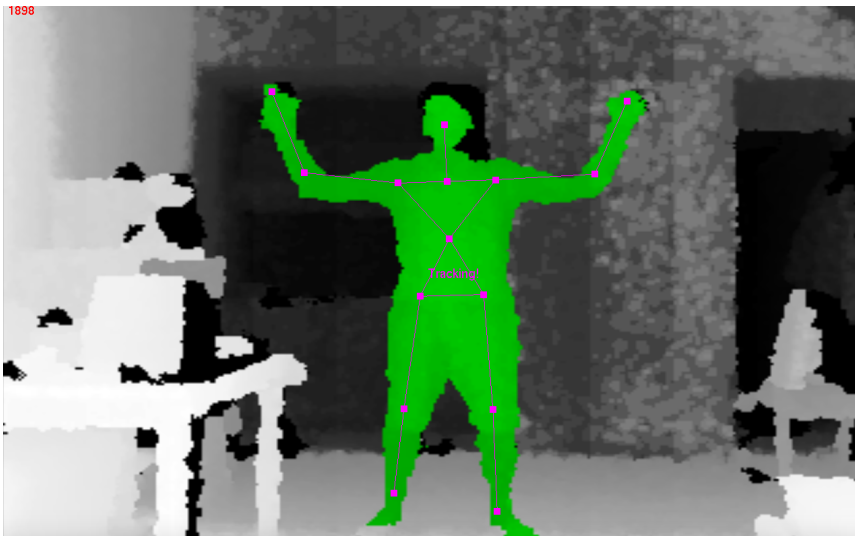

Figure 4.5: NiTE Skeleton Tracking

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 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 position data stored about the joint. The value is between 0 and 1, with increasing value indicating increasing confidence. Section 2.3.3.2 discusses extensively about the algorithm of NiTE.

Finally, Skeleton tracker serializes the C++ nite::UserData objects to JSON in strings as shown in 4.2.1.3 in order to asynchronously transfer to the client for further gesture recognition procedures.

```
{
  "HEAD": ["-274.5578", "583.2249", "1933.924"],
  "NECK": ["-286.0945", "471.8282", "1996.656"],
  "LEFT_SHOULDER": ["-399.2939", "453.2498", "1975.477"],
  "RIGHT_SHOULDER": ["-172.895", "490.4066", "2017.835"],
```

```
    "LEFT_ELBOW": ["-673.5372", "389.9277", "1973.389"],
    "RIGHT_ELBOW": ["77.3149", "437.1607", "2201.007"],
    "LEFT_HAND": ["-950.7228", "362.1895", "1930.967"],
    "RIGHT_HAND": ["351.137", "509.7826", "2453.827"],
    "TORSO": ["-258.3584", "272.1229", "2023.593"],
    "LEFT_HIP": ["-321.3845", "57.52153", "2033.549"],
    "RIGHT_HIP": ["-139.8603", "87.31343", "2067.511"],
    "LEFT_KNEE": ["-313.5818", "-344.5291", "2039.209"],
    "RIGHT_KNEE": ["-129.7786", "-280.5863", "2110.95"],
    "LEFT_FOOT": ["-341.4384", "-665.9058", "2189.055"],
    "RIGHT_FOOT": ["-172.1151", "-559.3973", "2262.547"]
}
```


4.2.2 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. Flowchart 4.6 shows the data flow of this module where the user is asked to select Prediction or Training of 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.

4.2.2.1 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 socket 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

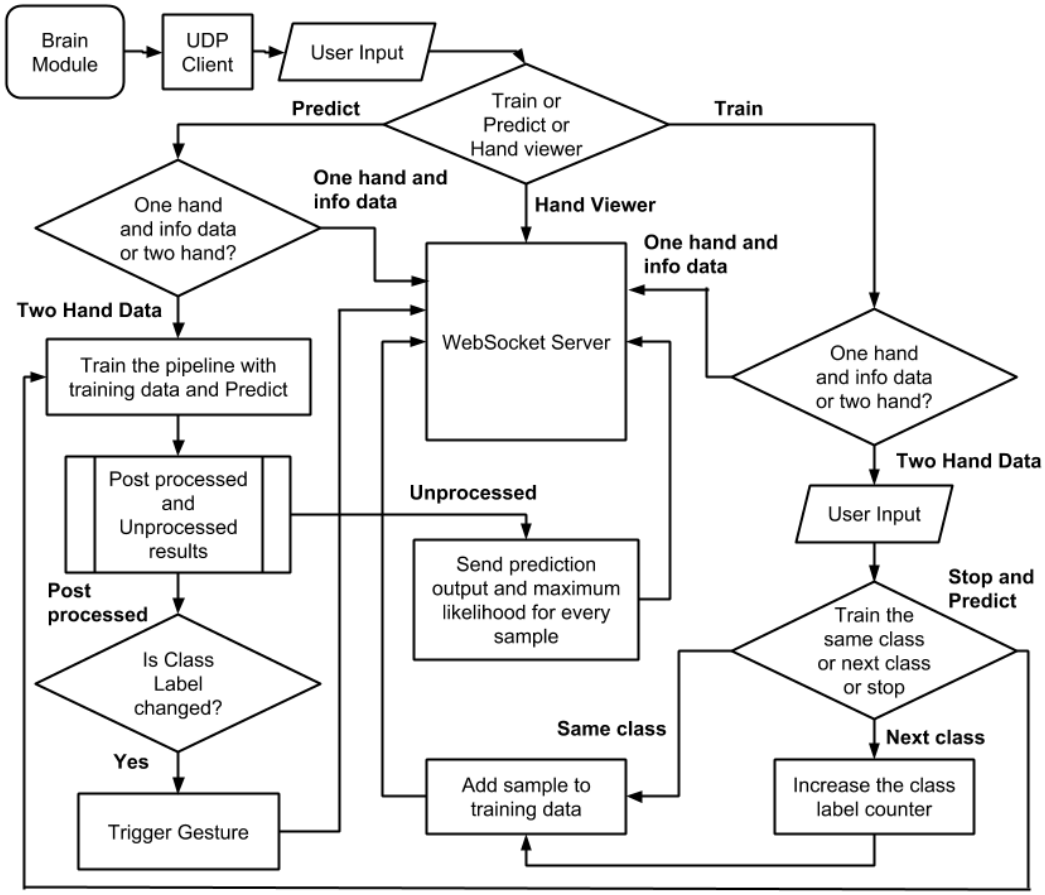

Figure 4.6: Brain Module Control Flow

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 flowchart 4.6 :

- 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

4.2.2.2 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 2.3.4.2, this component is based on the gesture recognition pipeline provided by Gesture Recognition Toolkit (GRT).

Flowchart 4.6 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 Adaptive Naive Bayes Classifier (ANBC) is chosen to be used in this thesis as described in the section 2.3.4.1. 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. In order 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 2.3.4.2, prediction output must be post processed in order to avoid false gesture 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 and 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 data has 6 dimensions. As described in the section 2.3.4.2, 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 TrainingDataRecordingTimer. 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 data 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 data 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 data 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 can 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.

4.2.2.3 WebSocket Server

WebSocket 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.

4.2.3 Control Center (CC) Module

Control Center plays an important role in this thesis. It is the eye that visualizes the internal status of the modules. It was first built in order to visually render the skeletal points of the human body that is been tracked by NiTE. Later it became one place to interact with the whole system.

CC is developed in Javascript with the help of WebGL renderer and jQuery. Everyday cloud computing is pushing computer applications to the Internet, which allows application software 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.

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 network 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 objects which rendered in WebGL. datGUI is a lightweight simple library to create GUI elements to build a dashboard in few lines of code.

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.

Figure shows how Control Center looks. UI element at left bottom in the console box that shows all the receiving data via WebSocket.

4.2.4 Command Module

4.2.5 Head Mount

4.2.6 Development Environment

4.2.7 Build and Deploy

4.3 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 Gesture Recognition Toolkit (GRT). Initially a set of simple gestures are chosen and the training data is collected for all those gestures.

4.3.1 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 4.7 shows the 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 4.3.2 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. Addi-

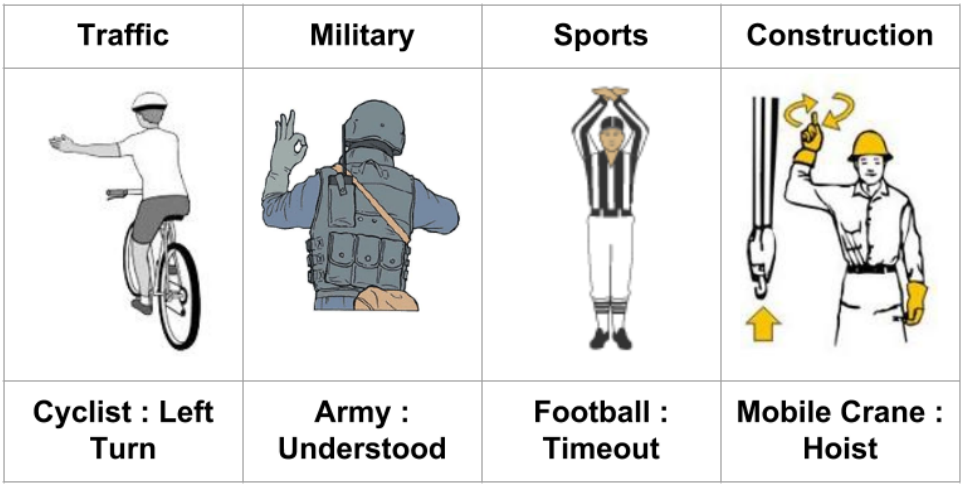

Figure 4.7: Hand Signals

tionally our system makes use of two dynamic gestures of NiTE which are used as focus gestures to gain control or start hand tracking.

Turn Left It is gesticulated as shown in the figure 4.8 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 4.9 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 4.10 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 4.10 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.

Walk It is gesticulated as shown in the figure 4.10 by holding the left and right hand up. It refers to an action that keep moving in the forward direction.

Wave It is gesticulated by holding a hand up and move it several times from left to right and back. This gesture is executed to initiate hand tracking, if no hands are tracked or tracked hand is been lost.

Figure 4.8: Turn Left Gesture

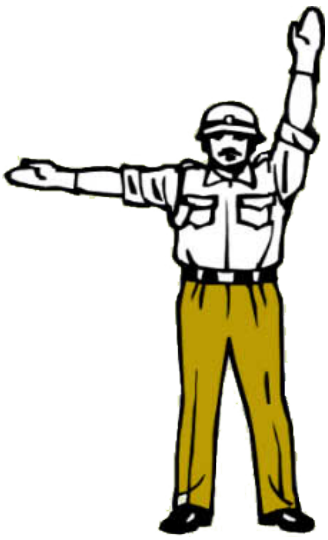

Figure 4.9: Turn Right Gesture

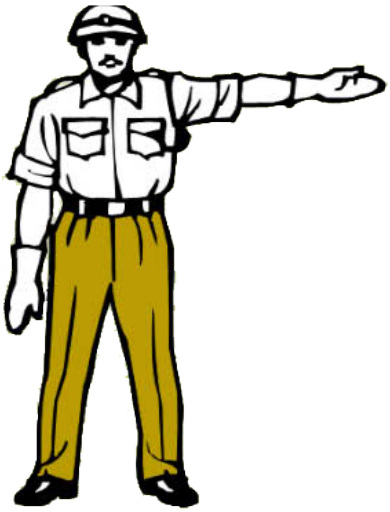

Figure 4.10: Move Left Gesture

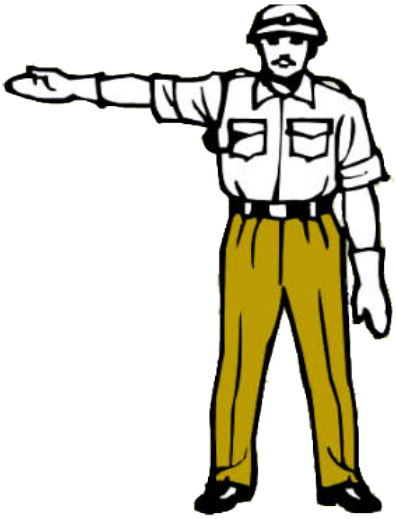

Figure 4.11: Move Right Gesture

Figure 4.12: Walk Gesture

Click It is gesticulated by holding a hand up, push the hand towards the sensor then immediately pull the hand backwards.

4.3.2 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 4.13.

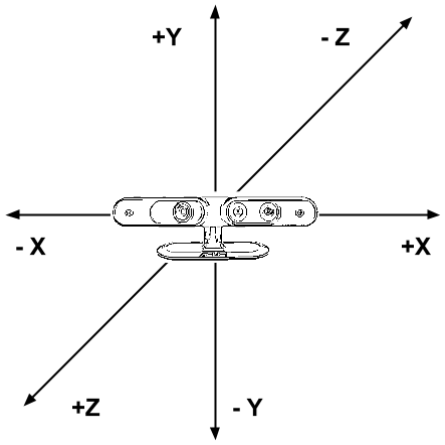

Figure 4.13: Depth Camera origin

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 another. Every gesture was 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 was gesticulated in 4 different positions as shown in the plot 4.14 and in all possible combinations that hands are kept wider or narrower as shown in the plots 4.3.2. 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.

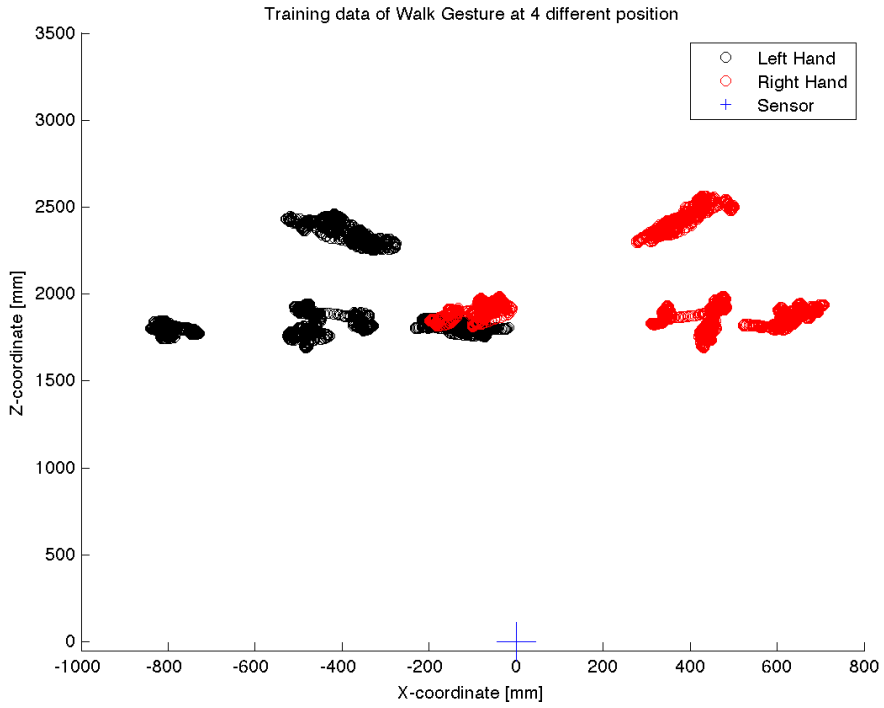

Figure 4.14: Walk Gesture in 4 different positions

ANBC is an iterative learning algorithm that improves the classification accuracy with increase in positive training data. Plot 4.3.2 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 to evaluation the accuracy of the recognition system. Furthermore, a set of non-gesture dataset was recorded in order to test the Null Rejection accuracy of the classifier.

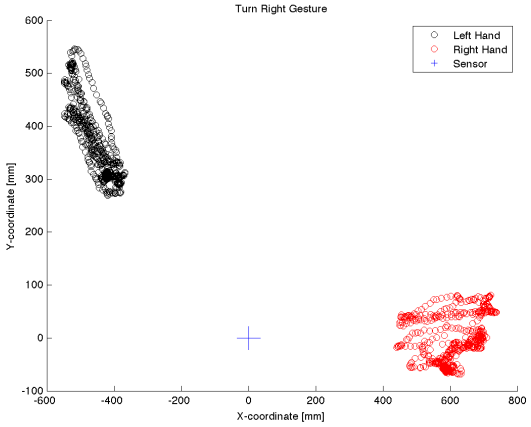

Figure 4.15: Turn Left Gesture

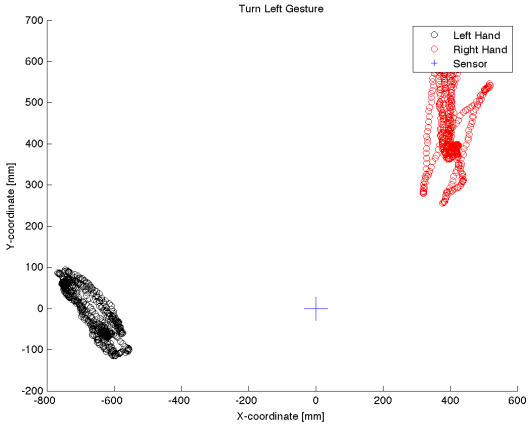

Figure 4.16: Turn Right Gesture

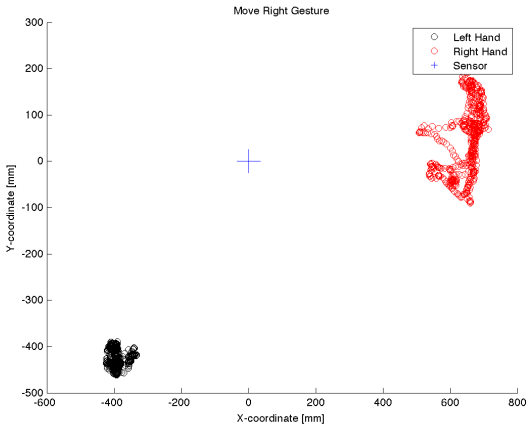

Figure 4.17: Move Left Gesture

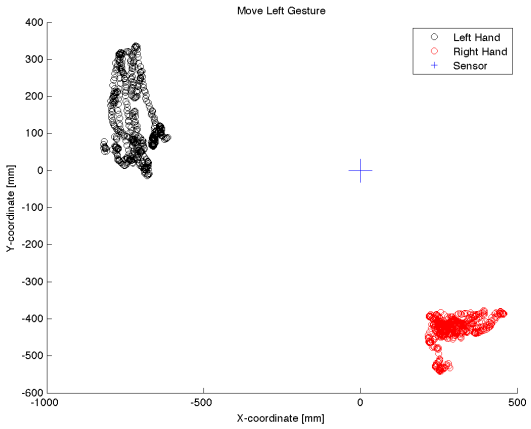

Figure 4.18: Move Right Gesture

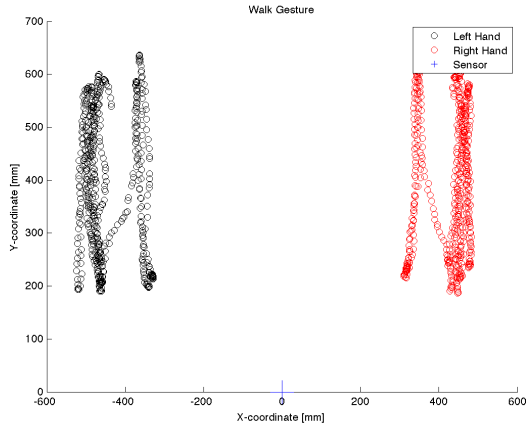

Figure 4.19: Walk Gesture

4.3.3 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 4.20 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 was gesticulated for more than one second.

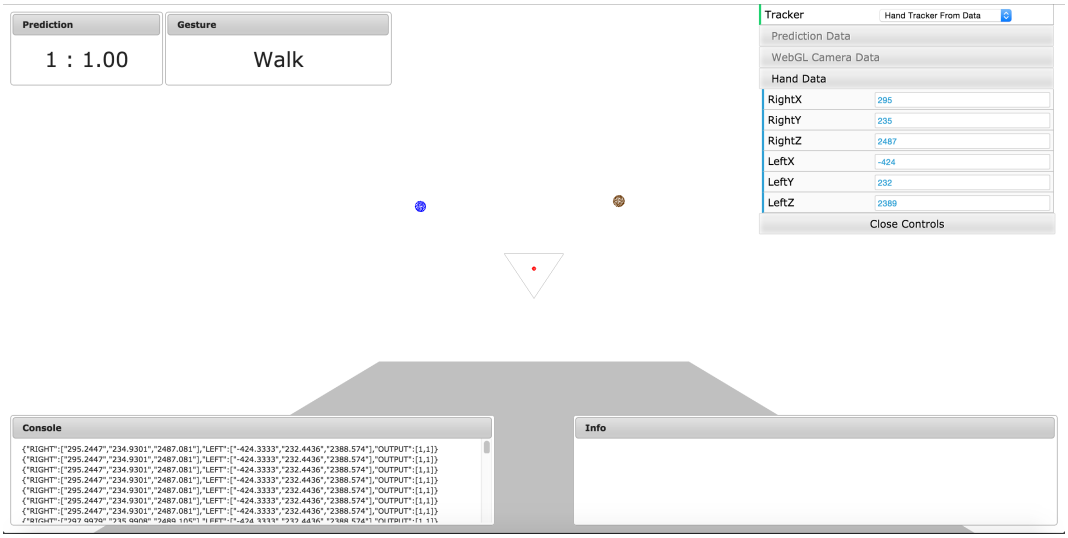

Figure 4.20: Walk Gesture

4.4 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 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 build 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.

4.4.1 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 was 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.

4.4.2 Gesture-to-Motion

This thesis was initially conceived as a hand gesture translator just to say the recognized gestures loud. To make this system more useful, Gesture-to-Motion feature was 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 was assigned a locomotion task as follows:

Walk This gesture commands the robot to walk in forward direction at a normalized maximum velocity (1.0) with the step frequency of 0.5. Robot walks approximately for 5 seconds and waits for the next command.

Turn Left This gesture commands the robot to rotate itself around z-axis in the left direction at a normalized maximum velocity (1.0) with the step frequency of 0.5. Robot walks approximately for 3 seconds and waits for the next command.

Turn Right This gesture commands the robot to rotate itself around z-axis in the right direction at a normalized maximum velocity (1.0) with the step frequency of 0.5. Robot walks approximately for 3 seconds and waits for the next command.

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

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

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

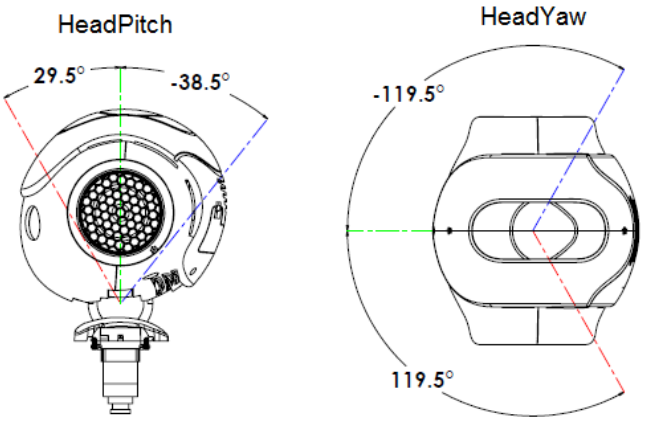

Figure 4.21: NAOs head Pitch and Yaw angle range

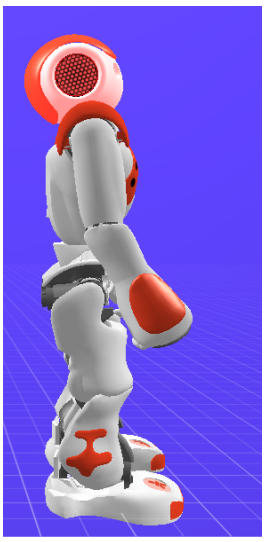

Figure 4.22: NAO with tilted head to look at the user

4.4.3 Head Position

As described in the section 4.3.2, collection of training data for each gesture was carried out in 4 different positions in front of the robot. During this phase robot was set to standing position where the height of the robot is 58 cm. Figure **??** 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 was set to -18.0 degrees and yaw was set to 0.0 degree as shown in the figure **??**. At this angle, field of view of the depth camera was 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 4.14.

4.4.4 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 were 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.

Chapter 5

Results

5.1

Bibliography

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
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