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 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

2.1.1 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 —-

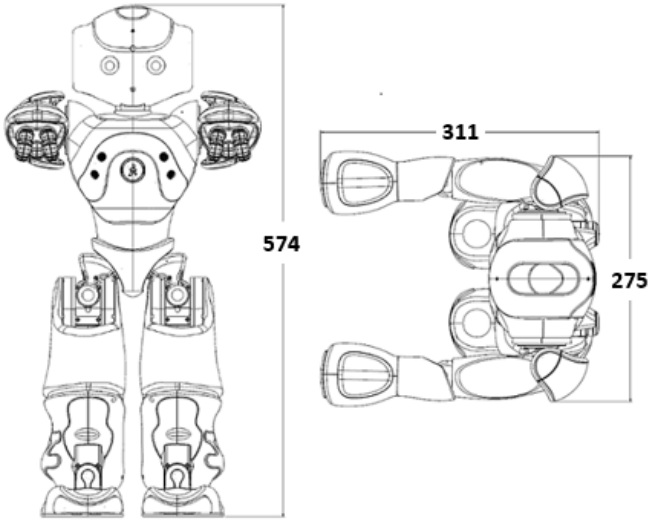

Figure 2.1: Construction of NAO

2.1.2 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.

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 2.1.2 shows how the robot can be moved to another position at the given normalized velocity using Motion API.

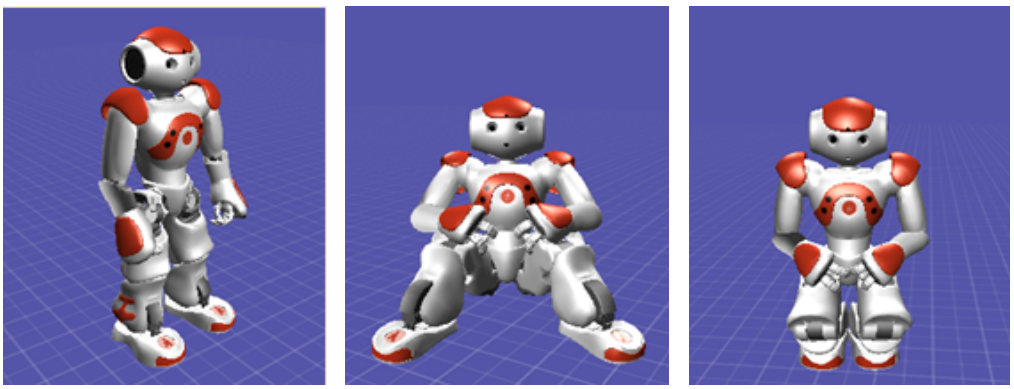

Figure 2.2: Standing, Sitting and Crouching positions of NAO using ALRobotPosture proxy

```
# 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()
```

2.1.3 Audio

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

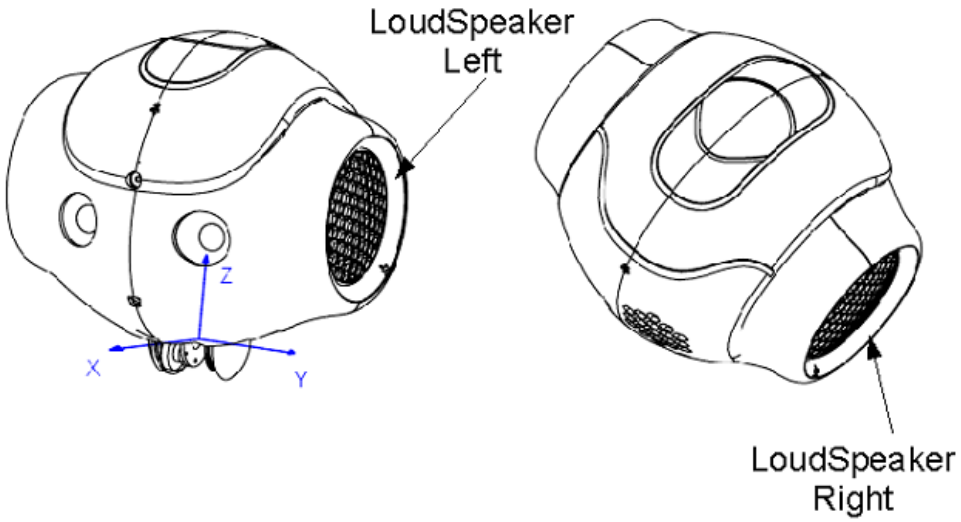

Figure 2.3: NAO Audio

In this thesis, we used Text-To-Speech API of NAO to say some words loud to communicate with the user.Python code 2.1.3 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")
```

2.1.4 Vision

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.

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.

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.

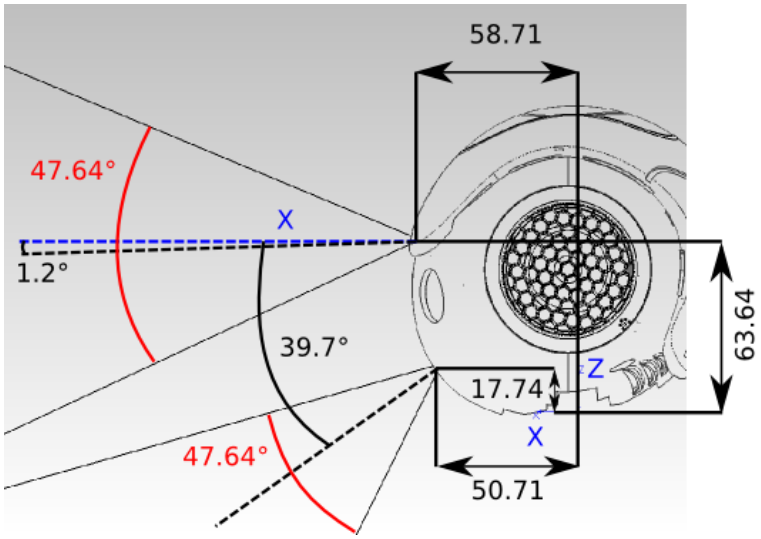

Figure 2.4: NAO Vision

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.

— this section needs revision —

2.1.5 Computing

NAO is equipped with Intel ATOM 1.6 GHz CPU in the head that runs a 32 bit Gentoo Linux to support Aldebarans proprietary middleware (NAOqi). NAOqi 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.

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

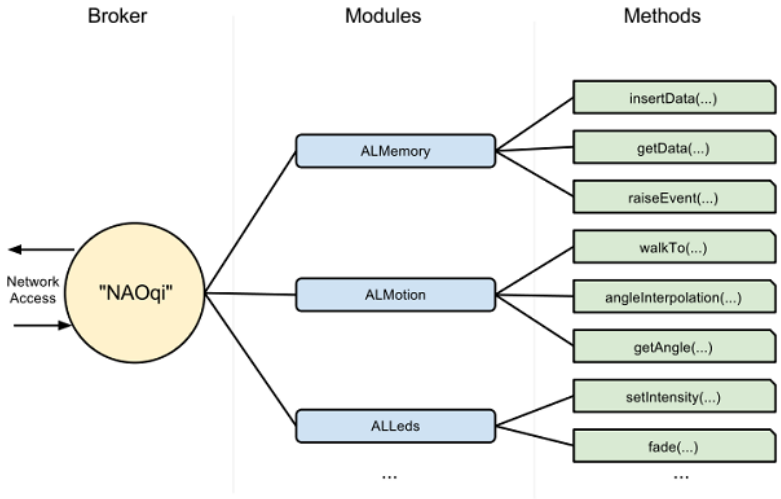

Figure 2.7: NAOqi Proxy

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.

2.2 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

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.

2.2.1 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.

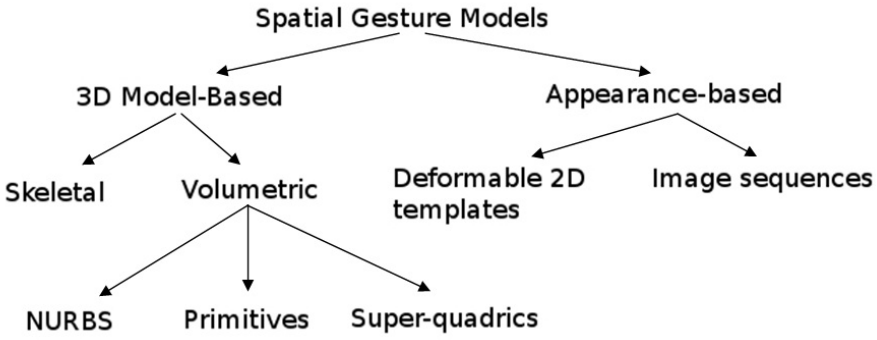

Figure 2.8: Gesture Modeling

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.

2.2.2 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.

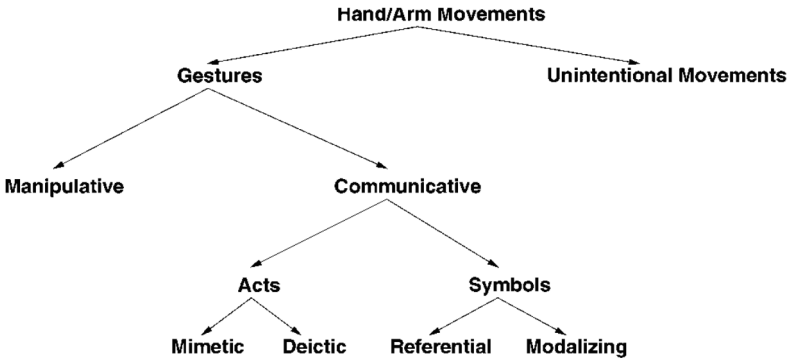

Figure 2.9: Gesture Taxonomy

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

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

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

2.2.3 Feature Extraction

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

2.2.3.1 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 2.2.3.1 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;
}
```

2.2.3.2 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.

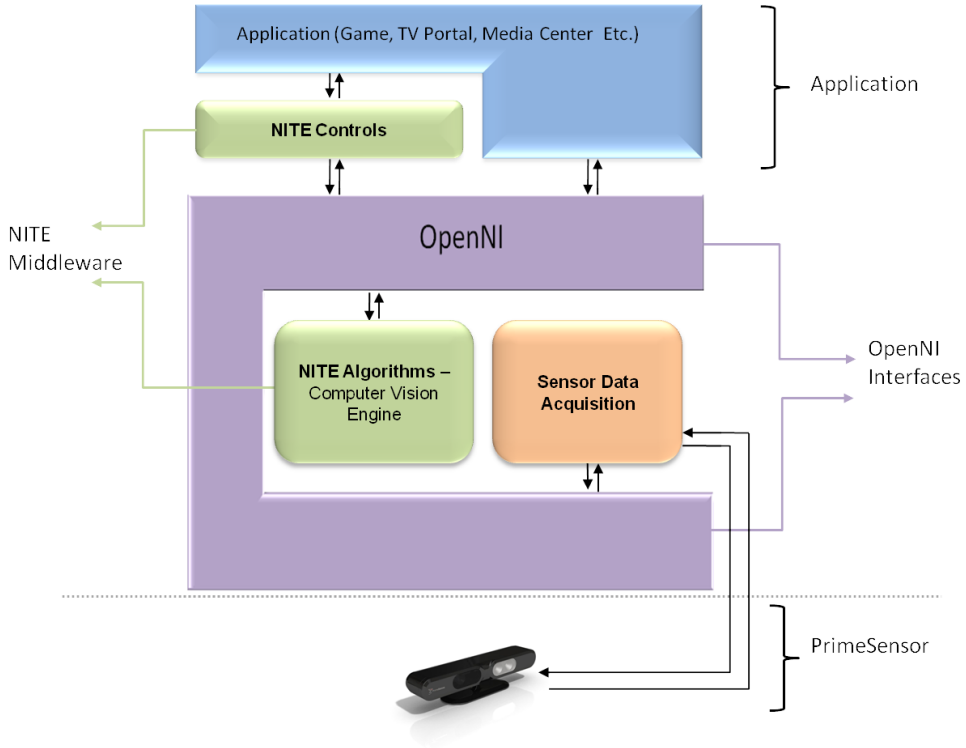

Figure 2.10: NiTE Architecture

- 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.

2.2.3.3 Skeletal Points Tracking Algorithm

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

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

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.

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.

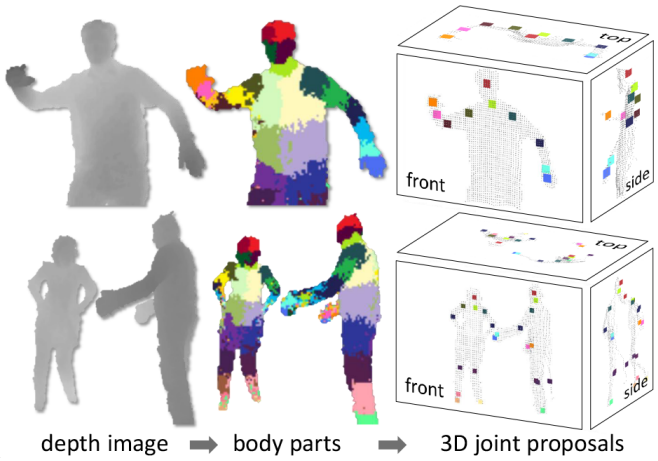

Figure 2.11: NiTE Algorithm to do real-time human pose recognition using depth images

Figure 2.12: NiTE Synthetic and real training

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 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.

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.

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

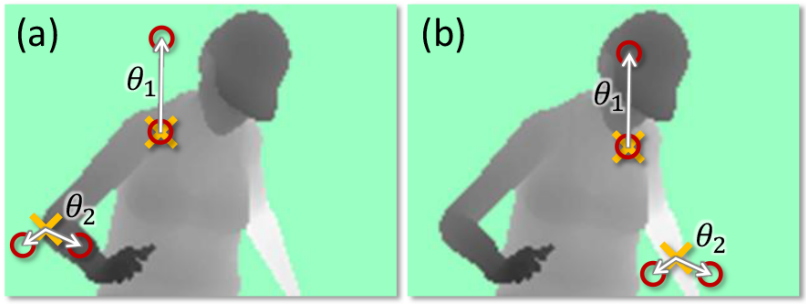

Figure 2.13: Pixels are labeled

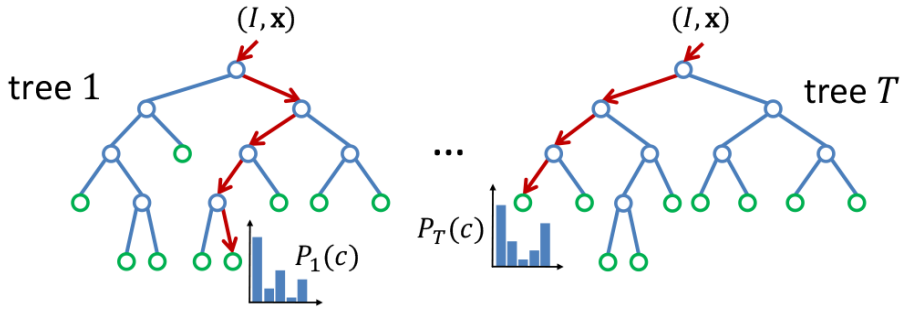

Figure 2.14: Randomized decision forest

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.

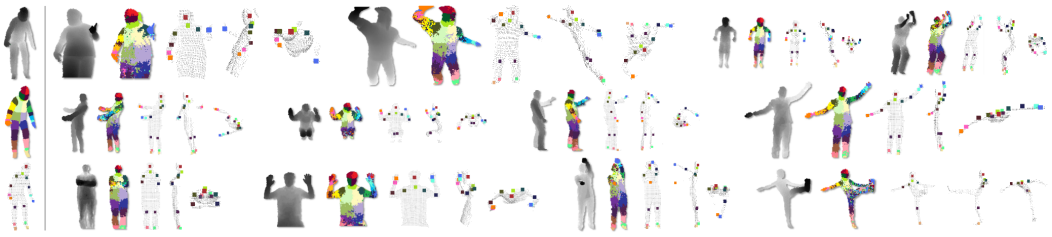

Figure 2.15: Pose Joint Proposal

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.

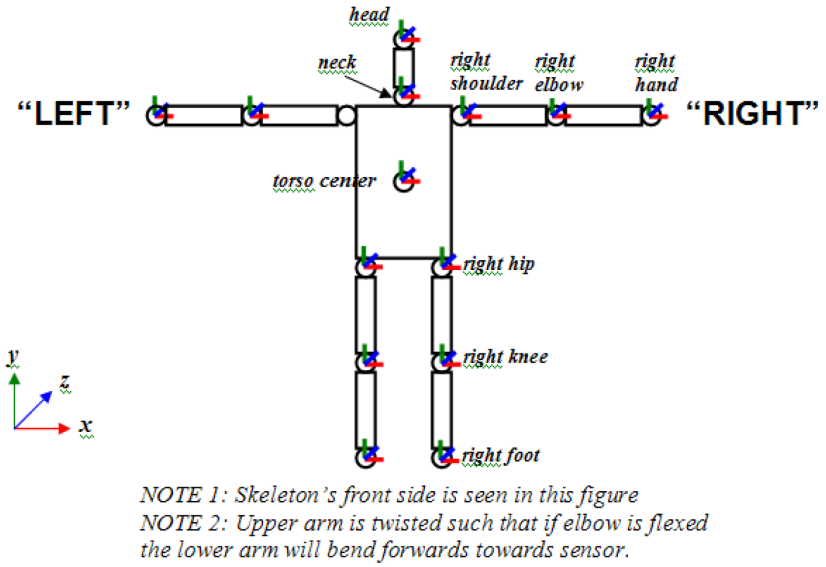

Figure 2.16: NiTE Skeletal Points

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.

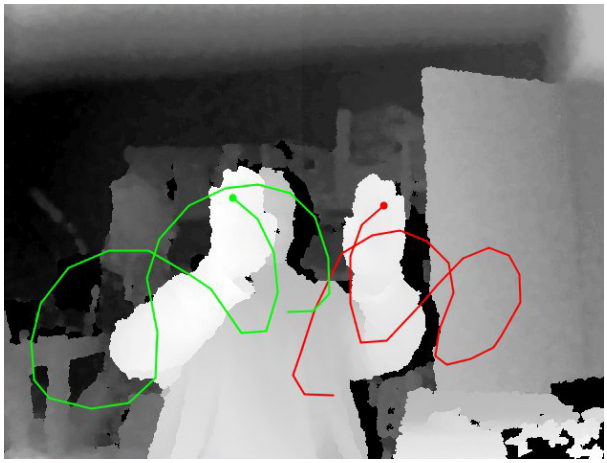

Figure 2.17: NiTE Hand Tracking

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.

Finally, C++ code 2.2.3.3 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;
}
```

2.2.4 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.

2.2.4.1 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 algorithm 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.

2.2.4.2 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.

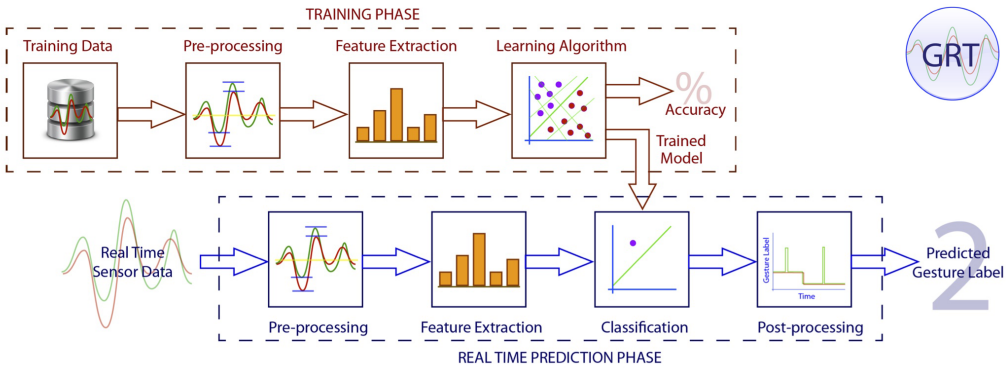

Figure 2.18: Gesture Recognition Pipeline

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 2.2.4.2 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 classification, regression and time series analysis. ClassificationData is the data structure used for supervised learning problems and for most of the non-temporal classification algorithms. C++ code 2.2.4.2 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.

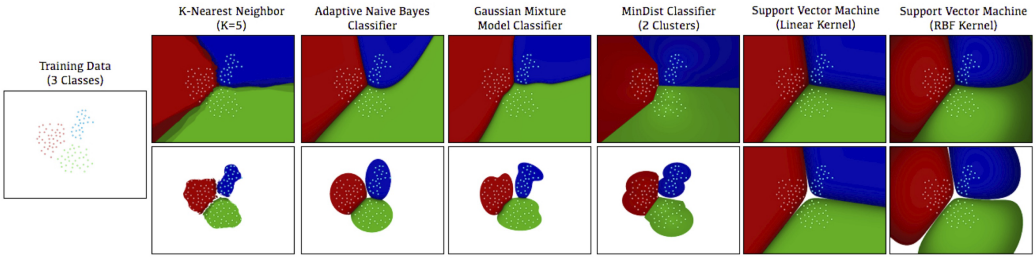

Figure 2.19: GRT Null Rejection

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.

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

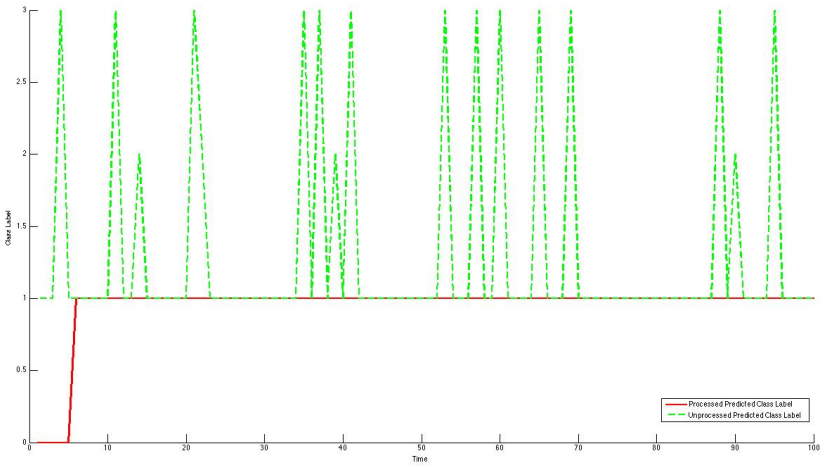

Figure 2.20: GRT Label Filter

size of the class labels buffer is set by the buffer size parameter. If there is more than one type of class label in the buffer then the class label with the maximum number of instances will be output. If the maximum number of instances for any class label in the buffer is less than the minimum count parameter then the Class Label Filter will output the default null rejection class label of 0.

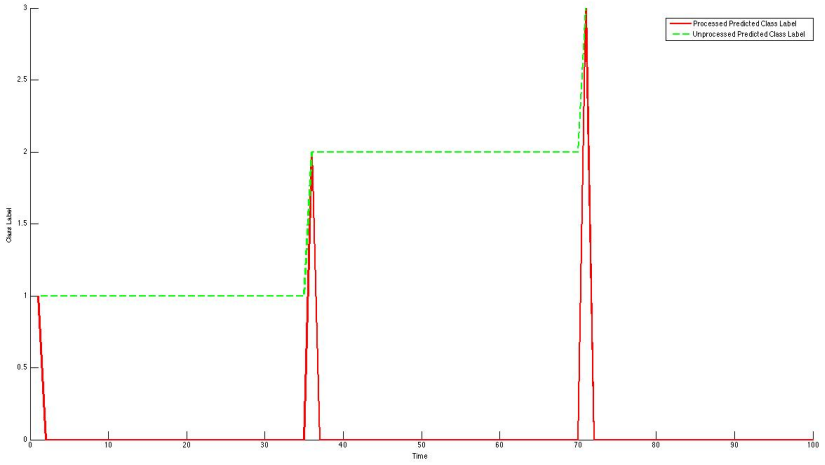

Figure 2.21: GRT Label Change Filter

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.

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.

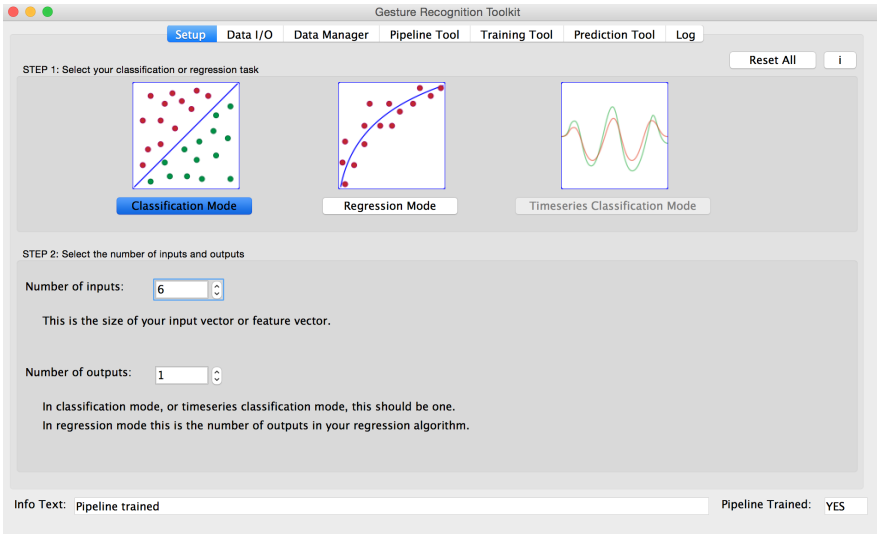

Figure 2.22: GRT GUI

2.3 Related Work

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.

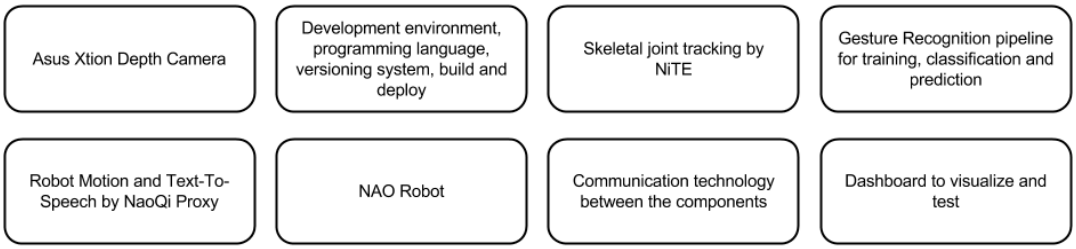

Figure 4.1: Component

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.

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.

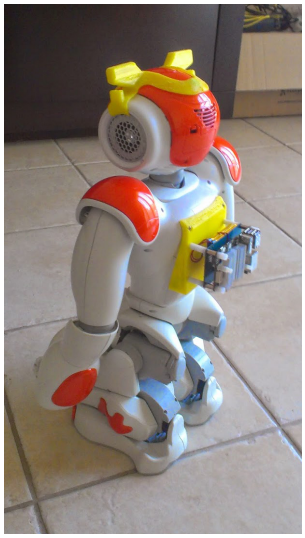

Figure 4.2: NAO Bag

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.

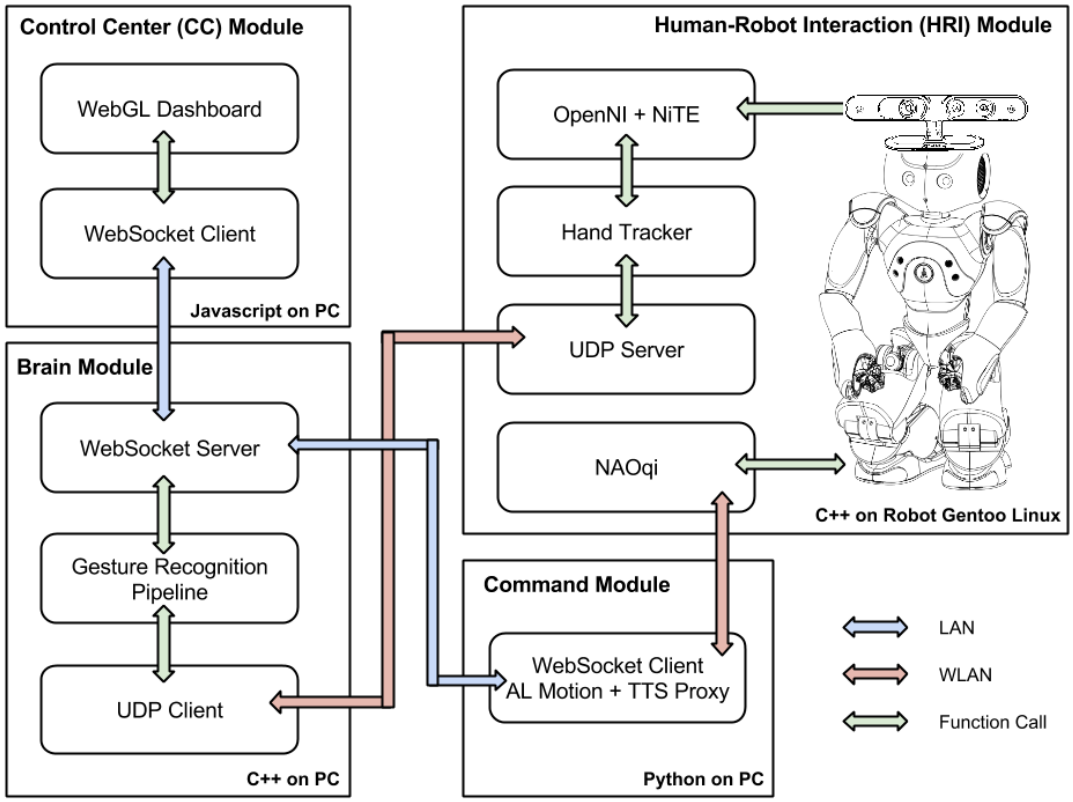

Figure 4.3: HRI Architecture

4.2.1 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 for the purpose of visually render the skeletal points of the human body that is being tracked by NiTE. Later, it became one place to interact with the whole system.

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

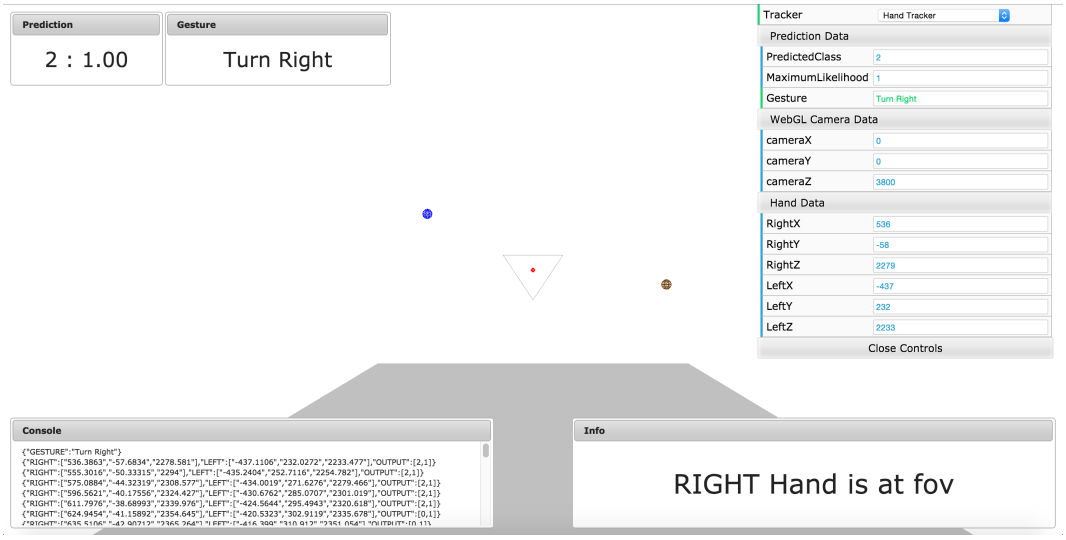

Figure 4.4: CC Hand

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.

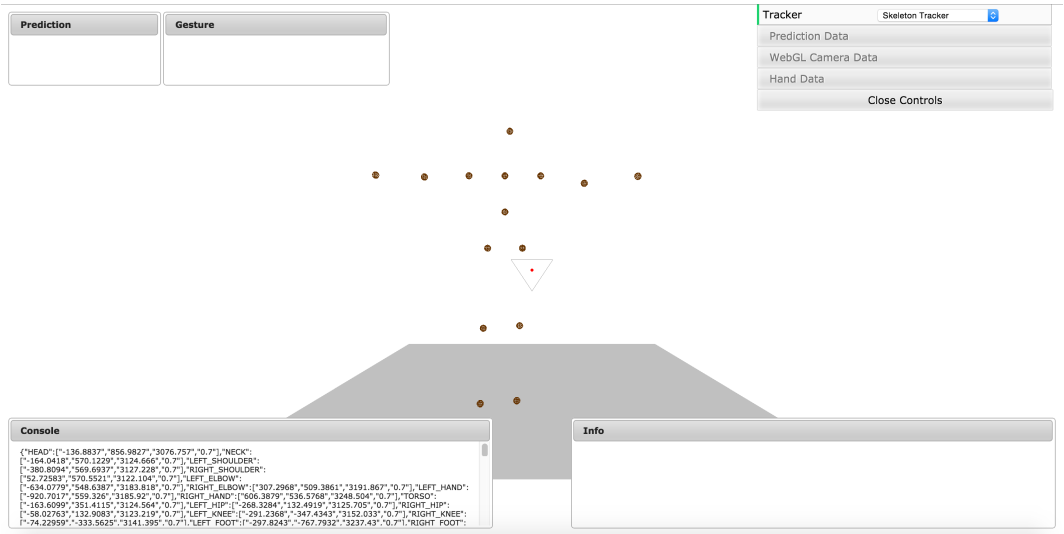

Figure 4.5: CC Skeleton

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

replay the positions of joints by storing them to a file and selecting Hand Tracker From Data option in the GUI. View automatically iterates through all the objects in the array and renders them at 60 fps.

User Interface (UI) Figure 4.4 the dashboard of the Control Center. Console box is an UI element that shows all the incoming data via WebSocket. It allows us to scroll through the data, if there is a necessity to cross check the data. Right bottom shows Info box which is created for the purpose of showing all intercommunication messages among all the modules. For example, "RIGHT Hand is at FOV" is an info message triggered by NiTE to inform that hand is closer to the field of view (FOV) and it may lose the hand. Left top corner displays two UI elements which are meant to show the prediction result for every input sample and recognized gesture that is triggered only after gesticulating it for more than one second. Furthermore, top right corner of the dashboard reveals more internal variables such as WebGL camera positions and real time hand in 3 dimensional Cartesian coordinates. CC can not only render hand joints, but also complete human skeleton with 15 skeletal point which are being tracked as shown in the figure 4.5. It also allows us to save tracking data to a json file and replay them by choosing appropriate mode from the drop down list on the top right corner.

4.2.2 Command Module

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

Command module is developed in Python with WebSocket and NAOqi libraries. Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

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

NAOqi NAOqi is the main software that runs on the robot and controls it. NAOqi SDK is the programming framework used to program Aldebaran robots. It allows homogeneous communication between different modules such as motion, audio, video. Python implementation of NAOqi is the simplest interface to NAO, since it allows us to run our code both on our computer or directly on the robot without compiling it to any executable. Hence, Python programming language is chosen to build this module.

ALProxy It is an object of NAOqi SDK that gives us access to all the methods or the module that are available in the robot. Via ALProxy, Command module makes use of three Aldebaran modules such as ALMotion, ALRobotPosture and ALTextToSpeech. Once the command module is started, it creates an ALProxy to the IP of NAO that is defined in the common configuration file. Thereafter, each proxy is ready to send the command to the local or the remote NAOqi.

ALMotion It provides methods which facilitate making the robot move. It contains four major groups of methods for controlling the following :

- Joint stiffness (basically motor On-Off)
- Joint position (interpolation, reactive control)
- Walk (distance and velocity control, world position and so on)
- Robot effector in the Cartesian space (inverse kinematics, whole body constraints)

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

ALRobotPosture It allows us to make the robot go to different predefined postures such as Crouch, LyingBack, LyingBelly, Sit, SitRelax, Stand, StandInit, StandZero. A posture for a robot is a (unique) configuration of its joints and of inertial sensor. Command module uses ALRobotPosture to make the robot stand up by calling robotPostureProxy.goToPosture("Stand", 0.5). This functionality is the very first interaction with the robot, if the robot is still in sleep mode.

ALTextToSpeech It allows the robot to speak. It sends commands to a text-to-speech engine, and authorizes also voice customization. The result of the synthesis is sent to the robots loudspeakers.

4.2.3 Head Mount

4.2.4 Development Environment

4.2.5 Build and Deploy