January 2015

Interface System for NAO Robots

Daniel Patrick Haggerty
Worcester Polytechnic Institute

Kshitij Thapa
Worcester Polytechnic Institute

WPI

Interface System for NAO Robots

A Major Qualifying Project Submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science

Submitted By:

Daniel P. Haggerty

Kshitij “Ken” Thapa

On-Site Liason:

Dr. Wang

Project Advisors:

Prof. Rong

Prof. Huang

Sponsoring Agency:

Shanghai University

Project Information

Joint-Project with 3 Shanghai University Students:

Neil King

Vicky

Hanqing

December 8, 2014

Authorship

The academic work of this project will consist of 3/3 unit of work from October 26th, 2014 – December 20th, 2014. Daniel Haggerty and Kshitij Thapa will take the credit for the work done on this project.

The writers for the paper are Daniel Haggerty and Kshitij Thapa. All parts were written and reviewed by both authors in equal parts.

Abstract

The goal of this project is to access NAO's strengths, implement entertainment application, and create more interactive experience through the vast array of sensors and capabilities available on NAO robots. NAO robots were designed for entertainment and educational purposes. The implementation for the project was realized through Choregraphe and C++ software. The project furthered the use of sensors and feedback on the robots to design a more interactive experience with the robots.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The NAO robot is an intelligent humanoid robot made by Aldebaran robotics. The robot's primary purpose is human interaction. With many capabilities and powerful customizable functions, this robot is an excellent platform for applications involving interaction and entertainment.

The NAO and its aesthetically pleasing appearance are great for entertainment. There are many possibilities but few implementations of them. This project creates a specific entertainment application for in-home interaction or for presentation. The main feature of the application created is a dance performed by the robot. The robot dances to the popular Chinese song 小苹果, little apple. The robot plays the song on the speakers and goes through the dance. This dance is popular and exciting, and the user of this application can dance along or simply watch the robot break out its moves. This is a more specific purpose for the many capabilities of NAO.

The project aims to use the vast capabilities of the NAO for specific purposes. The main goals are interaction and entertainment. Because the robot is made for interaction, an application that utilizes and advances these capabilities is very suitable. Using custom code written to access the robots sensors and motors allows much more control of the abilities of the robot that go beyond the basic functions provided for the standard user. The cute appearance of the robot coupled with its lights, speakers, and voice make it a great platform for entertainment. The robot comes with several pre-programmed short dances and games. A big performance and entertainment tailored to each user can add to these to make a well-rounded library of applications.

The objectives of the interaction portion of the project were an introduction and facial recognition. The robot should not simply perform for the user but it should interact with the user.

Using C++, access to the robots hand sensors and motors should be established. The robot extends its hand and waits for the user to shake. The robots motors then shake the user's hand, and the NAO introduces itself. Using C++ to accomplish this allows direct access to the motors and sensors of the robot and allows control of these without going through the more limited functions provided by Aldebaran. A facial recognition aspect of the application allows the entertainment to be tailored to each user. Aldebaran provides a way to store a face into the robots memory. When asked to tell a joke or a similar command, the robot can make decisions based on which user is giving the commands.

The dance is the main focus of the application. This dance can be interactive or simply a show. The user tells the robot to perform the dance, and the robot will play the song on the speakers while dancing along. When finished, the robot bows to its audience. This dance utilizes the motors of the robots in a much more extensive exploration of their capabilities. The dance pushes the motors of the robot to its limits with very quick motions reaching the extremes of its ranges of motion. The balance of the robot during these quick motions is easily upset and must be taken into consideration.

NAO is a powerful platform with many entertainment and interactive possibilities. By using C++ to gain a much deeper control of the robot and its capabilities, interaction with the user can be achieved. By providing a popular dance, amusing entertainment will be built into the application. This application should advance the utilization of NAOs capabilities and provide interactive entertainment for NAO users.

2. Background

This section covers the essential background information required for the project. This background information focuses primarily on the NAO robots. Most of the information presented in this background section is derived from the Aldebaran website for NAO robots.

2.1 What is a NAO?

The NAO robot is 573mm tall when standing upright. It includes two loudspeakers, 25 motors, two video cameras, four microphones, and over 80 LEDs[1]. Research into the components used in this project was done in order to better understand how the robot works and the capabilities of the components that would be used.

The NAO robot has many different sensors, motors, and joints. There are 14 touch sensors, located on the head, chest, hands, and feet. Some of these are capacitive while others are simple on/off switches. These sensors respond when being pushed, and, with the exception of the chest button, are not utilized unless specified in a particular application. The chest button is used to turn the robot on and off, as well as make the robot say its IP address, store a position, etc. depending on what state the robot is in. The reactions to any of these sensors being pushed are specified within applications and can range from a vocal reaction to one involving movement from the robot[2].

2.1.1 Sensors and Joints

In total, there are 25 motors controlling the movement of the NAO. All movement is controlled by these motors. Three different types of motors are utilized in the NAO robots. Type

[1] (Aldebaran-Robotics, 2012, p. Technical Overview)
[2] (Aldebaran-Robotics, 2012, p. Contact and tactile sensors)

1 is used in the legs, type 2 in the hands, and the third type is used in the arms and head[3]. The table below shows the specifications of each motor type, from the data sheet provided by Aldebaran Robotics:

Table 1: Specifications of Each Motor Type

	Motor type 1	**Motor type 2**	**Motor type 3**
Model	22NT82213P	17N88208E	16GT83210E
No load speed	8 300 rpm ±10%	8 400 rpm ±12%	10 700 rpm ±10%
Stall torque	68 mNm ±8%	9,4 mNm ±8%	14,3 mNm±8%
Nominal torque	16.1 mNm	4.9 mNm	6.2 mNm

These motors control each of the joints on the robots. There are two types of speed reduction ratios for each motor type, shown in the table provided by the Aldebaran Robotics data sheet:

Table 2: Speed Reduction Ratios

	Motor type 1	**Motor type 2**	**Motor type 3**
Type A	201.3	50.61	150.27
Type B	130.85	36.24	173.22

Each motor uses one of these speed reductions, depending on its type and position on the robot. The motors become hot if held in one place for too long, and it is important to turn off

[3] (Aldebaran-Robotics, 2012, p. Motors)

the motors whenever they are not being used. There is a simple way to do this in Choregraphe. The robot will say “motor hot” as a warning when the motors are becoming too hot.

The joints are shown in the picture below. The motors can all be controlled using the motion widget in Choregraphe, and the values of each motor will be shown. The picture on the right shows the motion widget, with the boxes used to edit the position of each motor on the right hand.

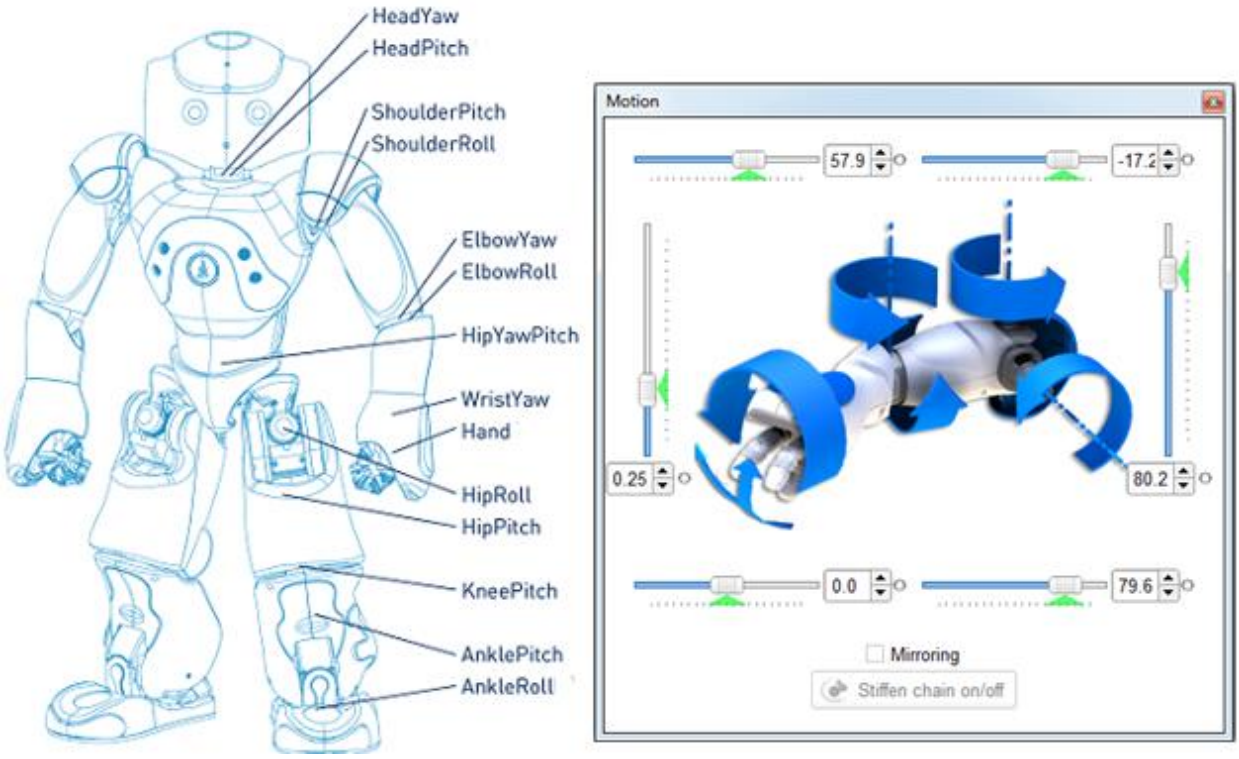

Figure 1: (Left) Labels of Each Joint and (Right) Motor Values for Each Joint

There are 14 different joints shown here. The other 11 joints are the arm and leg joints for the right side, controlling the shoulder, elbow, wrist, hand, hip, knee, and ankle. HeadYaw, HeadPitch, and HipYawPitch are the only three motors that do not have both a left and right side. The arms and legs can be moved individually, or be mirrored by selecting a box in the motion widget. The joints can be controlled by either entering the desired value or by moving the

slider until the joint has reached the desired position. If you are connected to a robot with Choregraphe, changing the value on the motion widget will change the physical robots joints real-time. According to the datasheet provided by Aldebaran, the position of the joints is determined using magnetic rotary encoders. These detect the position of the joints by utilizing Hall-effect sensor technology, and return the position of the joints with 12-bit precision. Hall-effect technology means that the position is determined by a sensor which can detect how far away the permanent magnet is. With these sensors, the position of NAOs joints can be determined with approximately 0.1˚ precision[4].

There are more than 80 LEDs on NAO. These LEDs are located on the top of the head, eyes, ears, chest, and feet. These LEDs are used to communicate to the user the status of the robot and also used to light up the robot when a sensor is pressed or when an application utilizes them. The messages communicated through the LEDs are important and should be taken seriously. The chest button lights up during boot and when the NAO is running. A blue light during boot indicates a firmware update has been requested, and a green steady light indicates that the boot process is stuck. Once the NAO is running the chest light indicates the state of the battery charge, or it can indicate that NAOqi is not running. There are also messages that the ears can indicate during bootup and upgrade. During animation mode, the LEDs on the eyes indicate whether the robot is listening, storing, or has accepted a position[5].

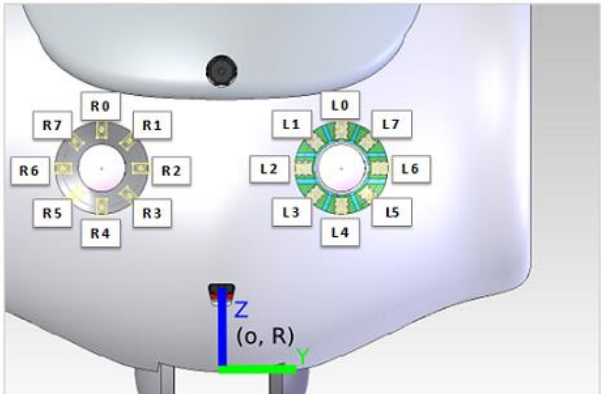

Figure 2: Eye LEDS

The NAO is equipped with two identical video cameras. These cameras are used to locate certain objects and recognize things like faces, goals, and balls. The Learn Face function, as well as other pre-existing functions in

[4] Ibid.
[5] (Aldebaran-Robotics, 2012, p. LEDs)

Choregraphe, utilizes the video cameras. The cameras can be used to record a video, take a picture, or allow the robot to follow a specific object.

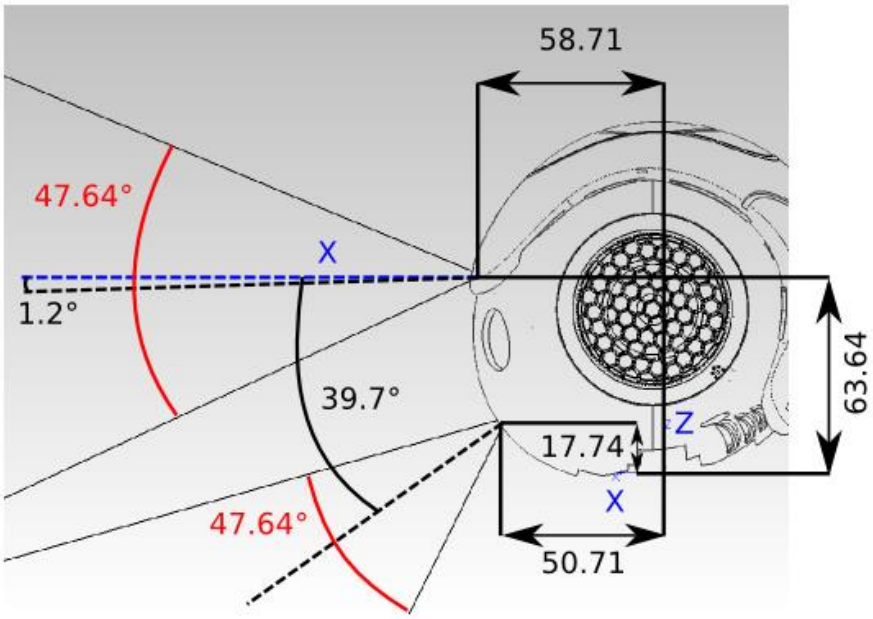

Figure 3: NAO Camera Angles

Each robot is also equipped with four microphones. These microphones are located on each ear and the front and back of the head. The microphones can be used for sound locating as well as speech recognition. The speech recognition is a pre-existing function provided in Choregraphe which can recognize speech and translate it to text. This can be used to accept commands given to the robot[6].

[6] (Aldebaran-Robotics, 2012, p. Microphones)

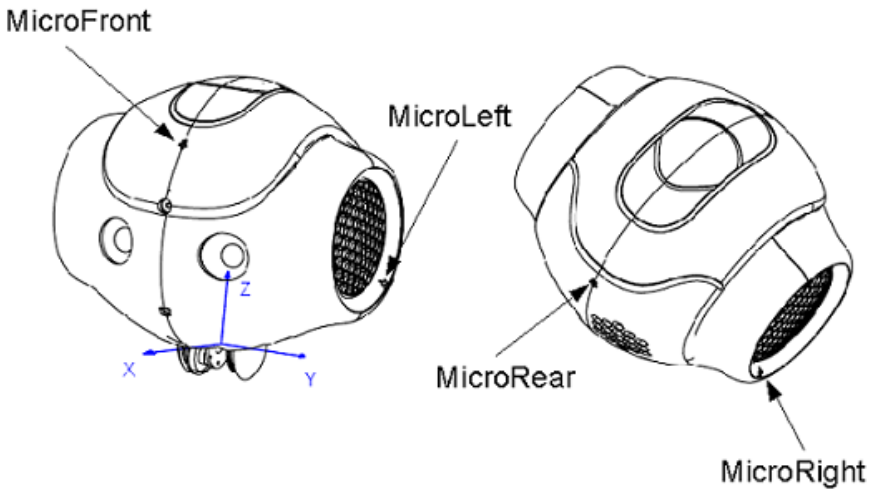

Figure 4: Locations of NAO Microphones

The available languages for the NAO text to speech and speech recognition are: Arabic, Chinese, Czech, Dutch, Danish, English, Finnish, French, German, Italian, Japanese, Korean, Polish, Portuguese, Spanish, Swedish, Russian and Turkish. When utilizing the text to speech function, the language selected is the language the text is written in, however, the robot will speak in the system language of the computer controlling NAOqi[7].

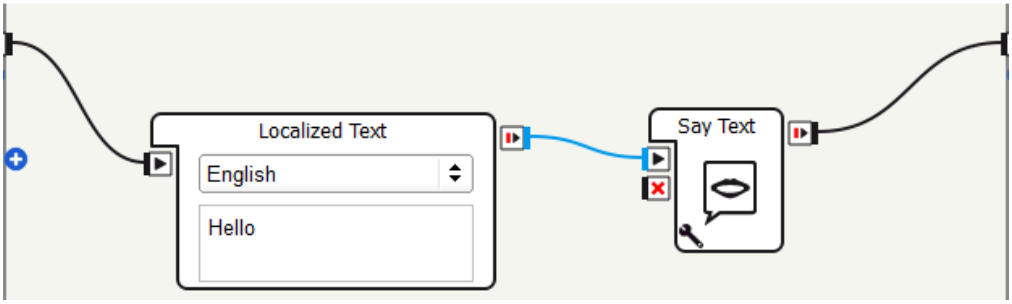

Figure 5: Choregraphe Implementation of Text to Speech

[7] (Aldebaran-Robotics, 2012, p. Available languages)

2.2 Software Programming

Nao robots offer a vast array of functions and capabilities that can be utilized as per user's will. In order to organize the sensors and hardware, the NAO comes with embedded software and desktop software. Embedded software is the program running on the motherboard of the NAOs and allows autonomous behaviors. Desktop software is the one written by the user, allows new behavior to run on robot remotely.

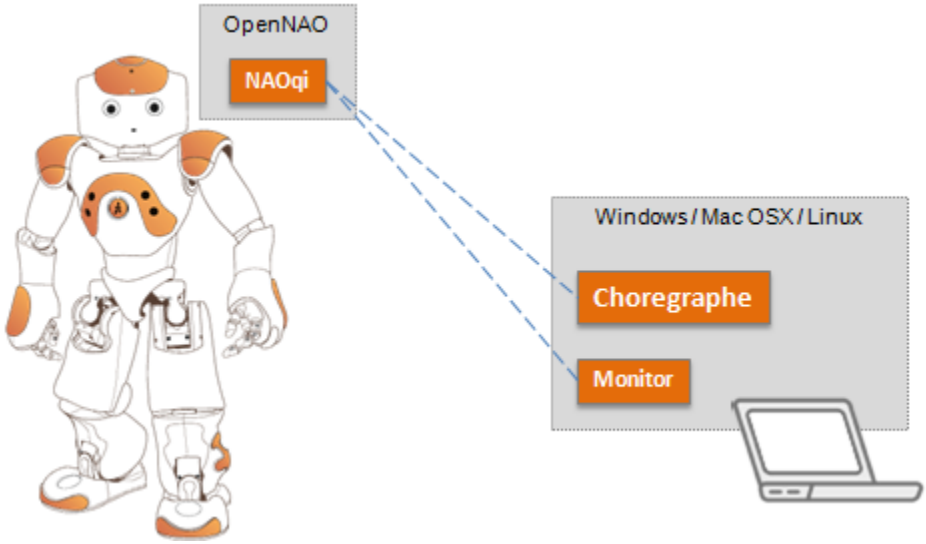

Figure 6: Embedded and Desktop Software

In order to run autonomous behavior on NAO, Aldebaran has developed custom operating system for the robot known as OpenNAO. It is a linux based operating system and as such retains linux platform for navigation and running code. Running on top of OpenNAO there is NAOqi, the main development software for the robots. All the modules and behavior that come with NAO is built upon using tools provided by the NAOqi[8].

[8] (Aldebaran-Robotics, 2012, p. NAOqi Framework)

Aside from autonomous behavior available natively through a new NAO, there are ways to program them through different platforms on computers. NAO have vast array of sensors and capabilities which can be utilized by a programmer. In order to make use of these functionalities, Aldebaran offers a few robust platforms. Some of the platforms utilized in this project are:

1. Choregraphe: platform specialized for NAO.
2. Labview: Visual programming platform.
3. C++ Software Development Kit (SDK): Written programming platform.

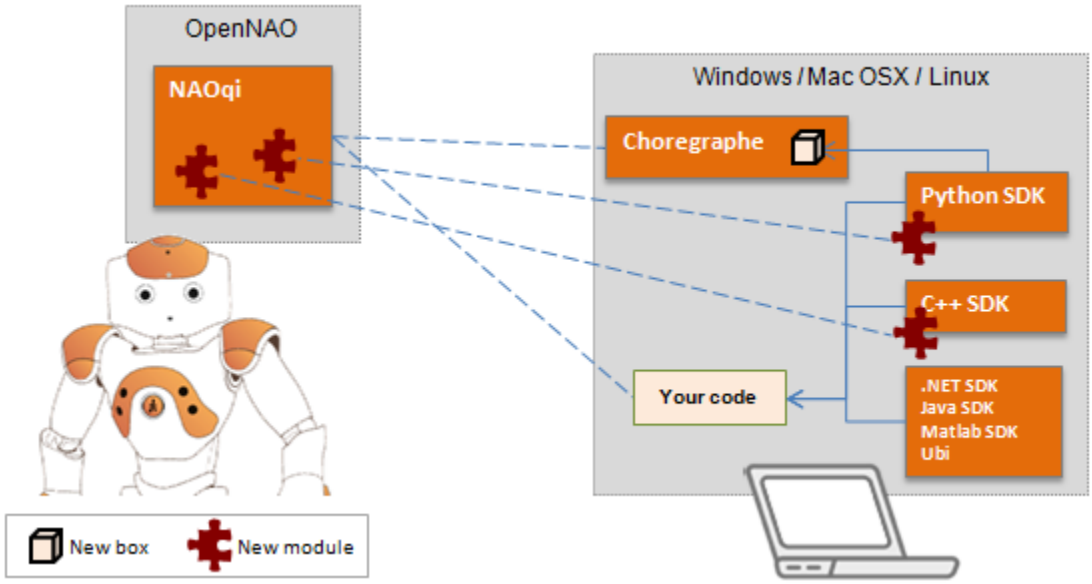

Figure 7: NAO Programming Platforms

These platforms provide an interface with NAO, making it simpler to control the robots. However they each have their specialty and limitation and as a result are suited for different tasks and users[9]. This section will cover what each of the above platforms brings to the table for programming NAOs.

[9] (Aldebaran-Robotics, 2012, p. SDKs)

2.2.1 NAOqi

The main software build on top of the OpenNAO operating system is NAOqi, developed by Aldebaran. It provides the framework with which to control the robots. NAOqi, along with OpenNAO, take care of translating user code to one with which NAO's can be controlled. While interfacing with the robot, NAOqi takes care of tasks including interfacing with sensors, sending the correct data to the joints of the robots, and providing communication with these features to the user.

NAOqi is built to address the common robotics needs which include: parallelism, resources, synchronization, and events. Parallelism refers to the ability of NAOqi to manage lot of data through breaking down a program into smaller parts and processing them at the same time. This is often used in modern processors to achieve much higher data processing speeds than serial processing. NAOqi provides wide variety of tools such as modules which can be accessed through different platforms on the computer. These are the resources provided by NAOqi. Synchronization is when tasks assigned to the robots are started at a pre-defined time as opposed to whenever the task is requested. This is done through internal clock in the NAOqi and provides more predictable timing as well as program behavior in comparison to asynchronous timing. Lastly events are internal flags that are raised whenever a given behavior occurs on the robot. This makes it easier for programmer to find such a behavior without having to look through data out of various sensors and joints.

In addition to addressing robotics needs, NAOqi also allows standardized communication between audio, motion, and video modules, standardized programming, and homogenous information sharing. This allows the programmer to develop the software on different operating systems as long as they are compiled using a compatible compiler. NAOqi also provides Application Programming Interface (API) for many different programming languages, but with

most robust support given to C++ and Python. Finally, NAOqi keeps track of all the functions available in different modules and provides their location[10].

Many times user does not directly interact with NAOqi. There are platforms which make the communication between NAOqi and user much more fluid, like Choregraphe. Others allow user more direct control over NAOqi like the C++ SDK.

2.2.2 Choregraphe

Choregraphe is a custom platform made by Aldebaran in order to program NAO with ease. It allows the user to create animations and behaviors which can be tested on a simulated robot as well as on a real one. In Choregraphe, it is also possible to create complex behavior such as interaction with people, dance, IR communication, etc. Choregraphe has access to all the modules provided by NAOqi and as such provides vast array of capabilities to the programmer.

[10] ibid

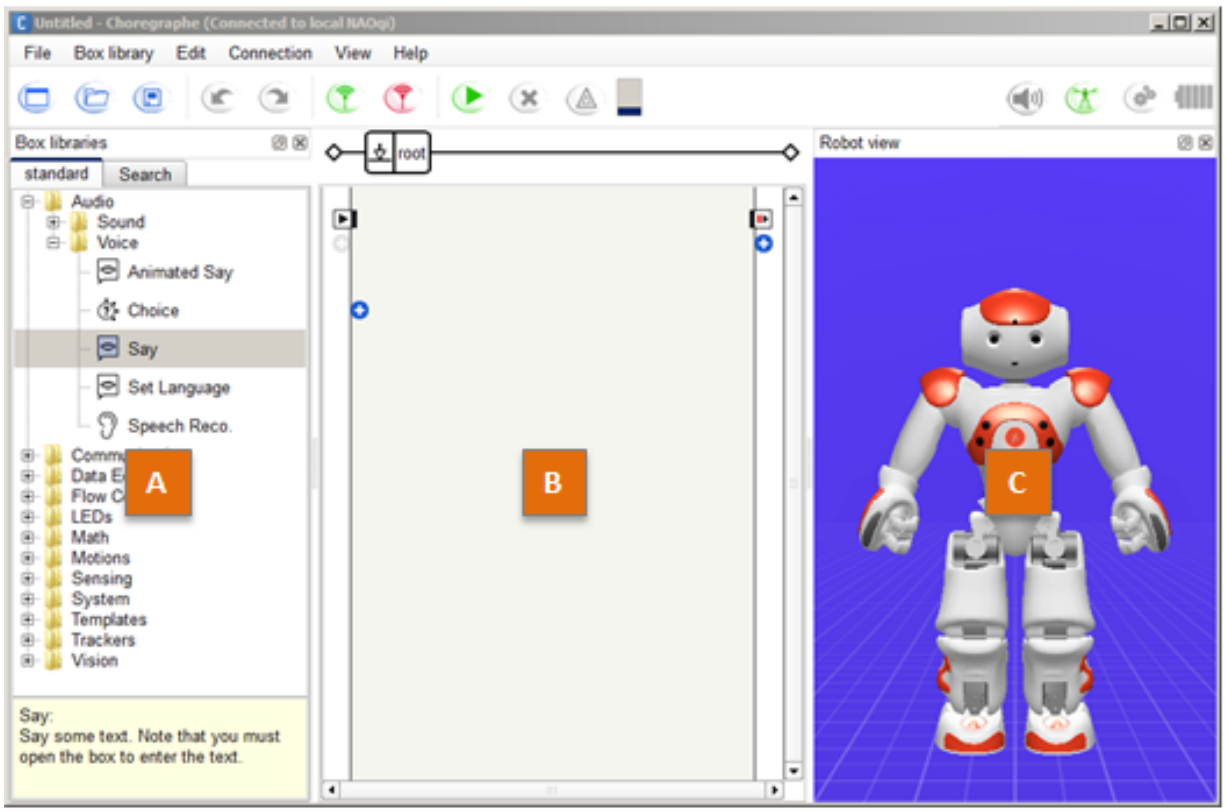

Figure 8: Choregraphe Overview. A – Box libraries panel. B – Flow diagram Panel. C – Robot View.

The figure above shows the basic panels available in Choregraphe. The first panel has a collection of standard and advance box libraries containing behaviors which can be utilized to make simple and advance behaviors on the robots. To begin, the user has to drag a function from the box libraries panel to the flow diagram panel. The functions shown on the box libraries panel are pre-programmed by Aldebaran for ease of use. These functions utilize many of the motors and sensors present on the robot. In addition, user is able to customize and change the given functions to fit their need.

Functions come in different categories ranging from simple movement to complex vision recognition. Functions can be modified or combined to create new features on the robot. For

each of these functions, there is a matching python code. The skeleton for python is similar for all functions and can be modified by the user. As a result, Choregraphe is graphical representation of python used to control the robots.

2.2.3 LabVIEW

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is a graphical design platform that made by National Instruments which allows for visual development environment. Labview is most commonly used for things such as acquiring data from connected instruments, controlling a device, and automating tasks on different operating systems. LabVIEW provides a graphical interface for programming[11].

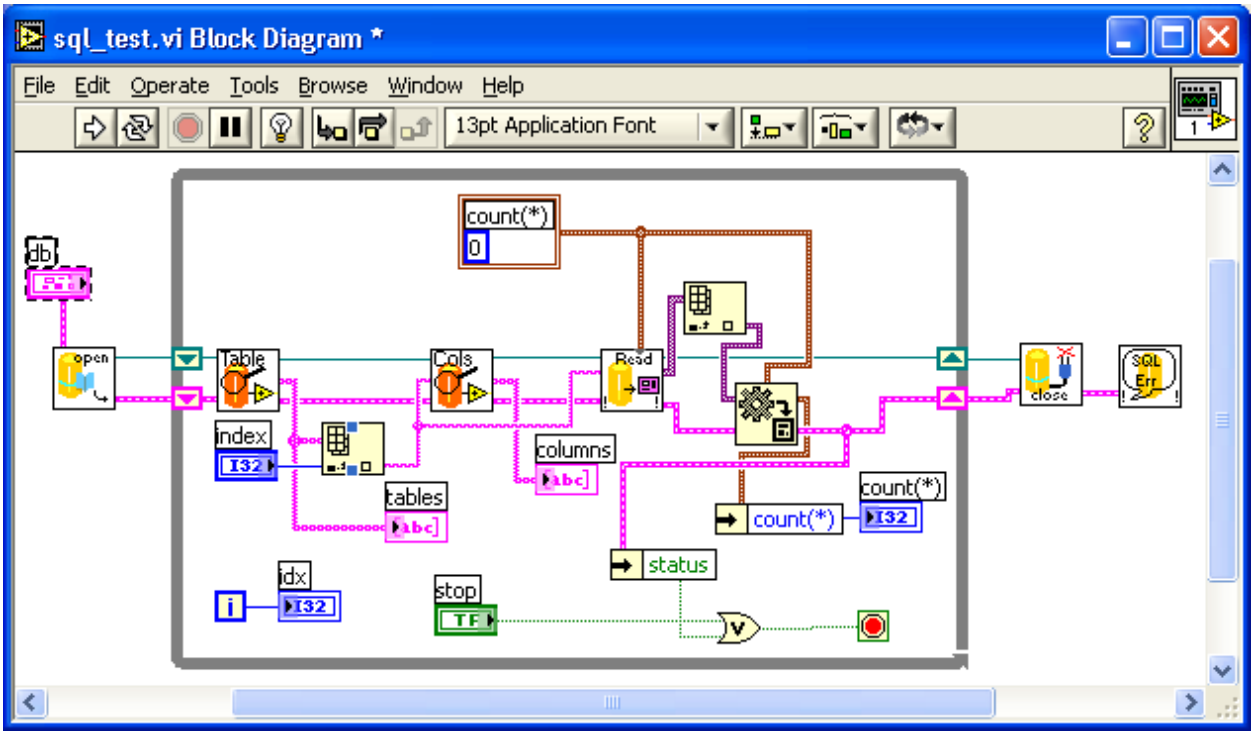

Figure 9: Example of Labview Function[12]

[11] (National Instruments, 2014)
[12] http://sql-lv.sourceforge.net/new_sql_LV.png

There are many similarities between Choregraphe and LabVIEW which results in overlapping functionalities. Both provide graphical interface with which user can customize their programming. LabVIEW provides a function block similar to Choregraphe, which is able to perform a specific task. These blocks have inputs with which the user can control the outcome from the functional block. As a result, LabVIEW is simple to use yet highly versatile. Since it is designed for functions other than just NAO robots, external functions within LabVIEW give users the ability to enhance their code. As a result, LabVIEW includes some functionality not readily available on Choregraphe.

2.2.4 C++ SDK

NAOqi offers many different ways to write code and run it on the NAO robots. The two most supported languages for developers are C++ and python. All existing API, sets of tools and routines for NAO are available for use in the C++ Software Development Kit (SDK). This allows any developer wide customizability over the tasks they wish to run on the robots. C++ is different from Choregraphe or LabVIEW as it does not have a graphical interface for users. As a result, C++ SDK is set for developers that prefer to bypass the graphical interface.

C++ SDK is versatile and available for use on different systems. Developers can choose to setup their environment in Windows or Linux. Both options are similar, however Linux allows the user to compile code that runs directly on the robot. When using Windows, the user must manually execute the code and select the correct IP for their robot. As a result, Linux provides users with more functionality.

The most difficult aspect of the C++ SDK's usage is the intricacies involved in setting up a working environment. The initial setup for the SDK requires the correct version of several

different programs. A coding platform needed for Windows is Microsoft Visual Studios which allows C++ programs to compile for later use. However, libraries used to compile programs for NAO are not native to Visual Studios. CMAKE is such a program that fetches the correct libraries and their dependencies. Designed specifically for NAO SDK development, qi-build is also a required program. Qi-build enhances the capabilities of CMAKE, making it easier for CMAKE to open and link libraries when required. Qibuild uses some scripting which requires Python to be downloaded as well. Finally, with all these programs set up, NAO C++ SDK can be installed on the computer and be ready for use[13].

[13] (Aldebaran-Robotics, 2012, p. NAO C++ Development)

3. Methodology

This project's goal is to find entertainment application for the vast array of sensors and capabilities available on NAO robots. NAO's were made for entertainment and educational purposes and as such furthering the entertainment aspect of them seemed appropriate. With the goal in mind, three objectives were drawn:

1. Accessing the strengths and shortcomings of NAO robots.
2. Implementing an entertainment application.
3. Creating libraries of complex implementation.

The methods behind these objectives will be covered in following sections.

3.1 Feature Evaluation

The capabilities of the robot were assessed before work began on the application. This allowed a better understanding of the robot and its strengths. There were five main functionalities tested. Each was tested by the entire team after a method for determining the capabilities was agreed upon. The results were recorded in the lab notebook and then put into the report under the results section. The team performed these tests in a controlled lab environment, and used the same robot for each trial in order to keep results uniform. Several robots were used throughout the project, all the results are confirmed and not due to any specific robot shortcomings or malfunctions.

The first test was simply commanding the robot to stand and sit through Chorepgraphe software's voice command. The next was to assess the robots face recognition function. This was done through the face recognition method available on Choregraphe. The method was run on several different faces and the results ability to record and recognition those faces was

tested. The third test was having the robot locate a sound and turning toward it. With sound locating, three different distances were tested in 180° field of view in from of the robot. These distances were same for face recognition as well and are as follows:

Table 3: Face Recognition and Sound Locating Distances

Distance (Feet)
1-2 Near
2-4 Mid
4-6 Far

. Fourth, the robot tracked a red ball. This is an innate feature of the NAO's allowing them to track either a face or a red ball. Tracking was done using a softball sized red ball and continued for 5 minutes. If the robot lost track of the ball, test was deemed a failure. Lastly, the robot was given voice commands and its ability to recognize the words was assessed. The three words used for testing were: sit, stand, and joke. Two of these words are similar to each other so as to determine how difficult it is to differentiate between similar sounding words. Only one voice operator was used for this task so as to control the variables. Robot was made to repeat what the understood word was. Each test was repeated and the results were recorded.

3.2 Dance Animation

To animate the dance, the timeline function on Choregraphe was used. The timeline function allows the user to store specific positions the robot is in and put them on a timeline animation. There are a few different methods to animate the robots and store the positions. It is

possible to relax the robots joints, which allows them to be put into any position by physically moving the robot. Once in the desired position, a keyframe can be created by either pushing the chest button, then saying “store position” or by saving the position onto the timeline using the keyboard shortcuts. Next the timing was be addressed by dragging the keyframes to specific points on the timeline. Each frame represents a certain amount of time, depending on the frame rate which can be specified by the user. The entire timeline shows each specified position and when played, the robot will move between positions at whichever rate is necessary in order to be in the correct position by the next keyframe. This means that the user is responsible for not putting the keyframes too close together, or else the robot will try to move too quickly and lose balance.

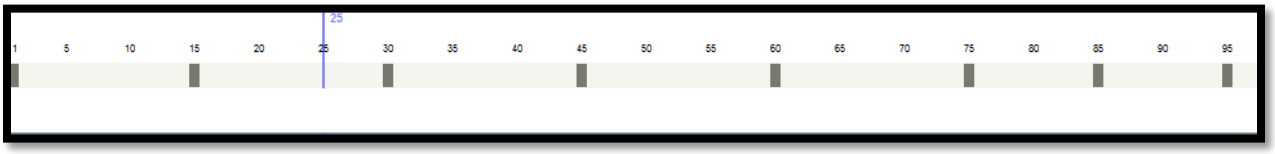

Figure 10: Timeline

The motor positions for each joint can be edited on the timeline itself, in the timeline editor. The timeline editor shows the transitions between each keyframe with the values for

each motor on a timeline graph.

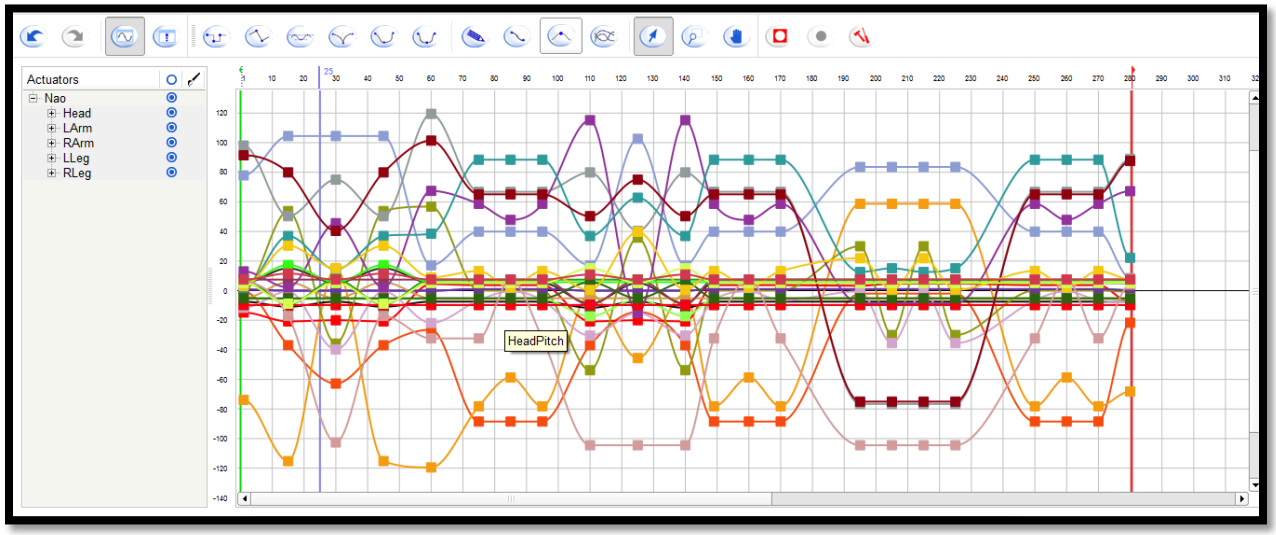

Figure 11: Timeline Editor

The motor positions at each keyframe can be found by using the timeline editor or simply moving the mouse over the keyframe in question. These positions can be used as a reference for a later keyframe, copied into another keyframe or edited for more precise control of the robot.

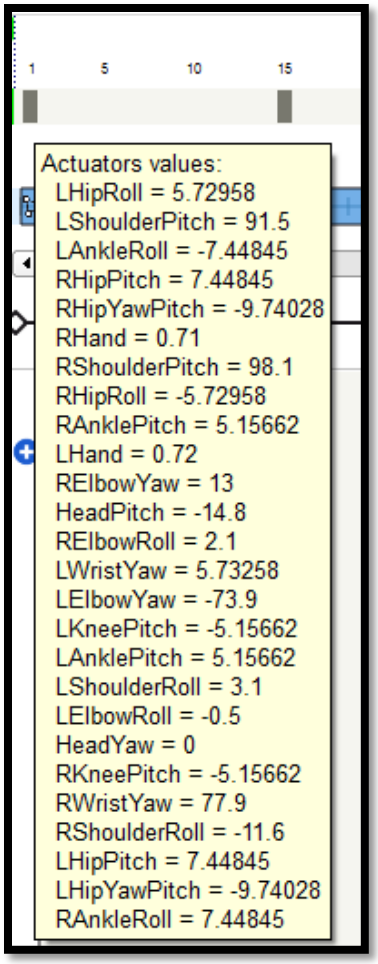

Figure 12: Motor Position at Keyframe 1

Once the dance was chosen, a video of the dance was selected in order to have a reference the entire team agreed upon. The video was split into five parts and each team member animated a specific part. The parts were animated and then combined afterwards to form a complete dance. First, the hands were animated. The hands were the easiest to animate because the robot could be standing still while this was done. Next the legs were animated. This was difficult, because the robots tend to fall over. Due to the robots being so unstable, there were a variety of methods to complete the dance. Many of the dance moves involved the dance

standing on one leg and lifting the other up. This was impossible for the robots to do so there was compromise. The motion widget tool and labview were essential for this portion.

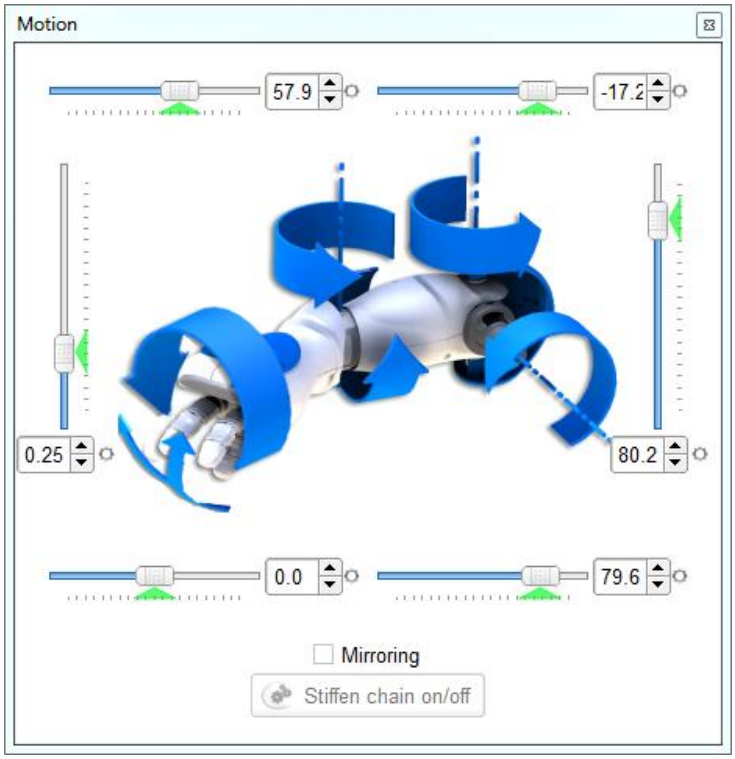

Figure 13: Motion Widget Tool

To use the motion widget tool, a connection to the robot must be established, and a specific part of the robot selected. The values for each motor are then shown and can be changed to make the robot move. One method for moving the robots legs without causing it to be unstable was to write down the motor values of a stable position, and copy those into the frame. Another was to use the motion widget tool to slowly move the robot to a position and then save that in the frame. The robot had to be moved slowly but because the dance was done at a higher speed, it could still become unstable from moving positions too fast and many problems were encountered.

The final dance was the combined animations from each team member. Because each team member animated slightly differently, combining the parts was a process. First, the five parts needed to be placed one after another. Then the parts needed to be analyzed, making sure the end of each part flowed into the beginning of the next part without any issues or awkward pauses. The next part was to adjust the speed of each animation to match each other. Some of the team members used slower animations which looked odd when combined with the faster ones. The entire dance was combined and the speeds adjusted to make one fluid dance. Finally, the dance only took about half of the time the song did. The dance was repeated and adjusted so that the entire thing took exactly as long as the song did, and the robots could run through the entire thing without any issues.

3.3 Advanced Feature

So far, the programs already present for use on NAO robots were used to finish different tasks. With the third objective, intention was to be directly able to use the sensors available on the robots to create a new helpful module. In order to achieve this, attention was focused on the lack luster introduction that is available on the NAO robots, particularly the handshake module.

The handshake program given by Aldebaran consists of only one simple motion: raising right arm up. There is no shaking motion or feedback from the user. After looking at the configuration of the sensors on NAO, capabilities were found in order to add missing pieces to the handshake module and make it more intuitive experience. Motion and feedback will be added on to create a new handshaking program. This program can be loaded on to the robots at their startup so that whenever the robots are turned on, simply touching one of the designated sensors will set off the handshaking procedure. Thus, this program would have to written in C++ (or Python), which are the only two languages supported for startup program use.

After the setup was completed, the focus of the project turned to writing the actual program. There are three sensors located on each hand of NAO robot which will be utilized to gather feedback from the user. These sensors are shown in the figure below:

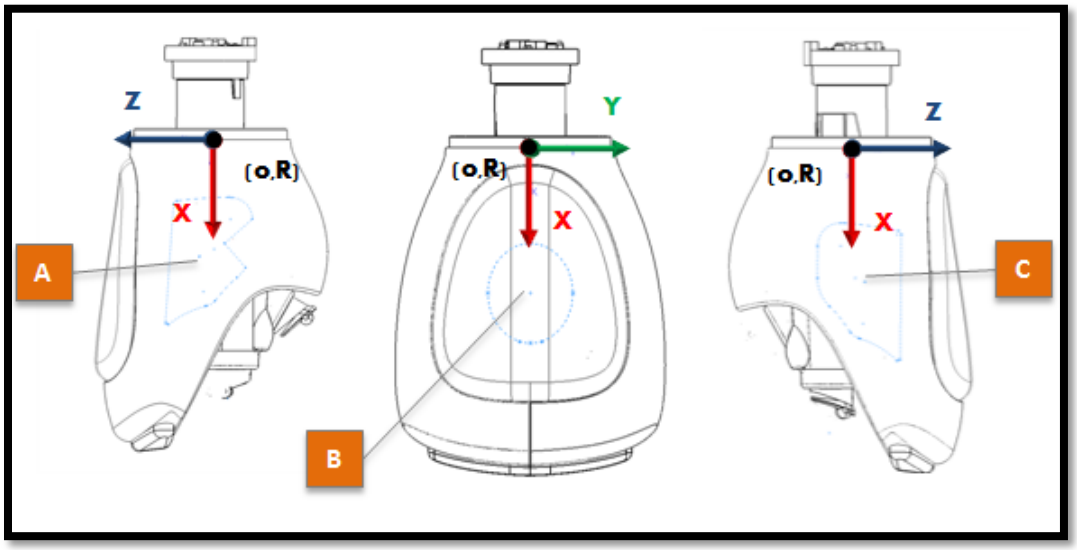

Figure 14: Hand Sensors

These sensors can send back touch data to the robot which reactions can be based on. Based on these reactions, native motions are added to the robots to complete the handshaking procedure. The program would start from a touch on the tactile sensors on the head. At this point the NAO robot will raise his hand and wait for the user to grab its hand. Completion of this step will be determined by polling the three sensors on the hands of NAO. Once sensors register a touch, the hand shaking motion will be put into effect. All in all, the motion and feedback are based upon the procedures used by humans to interact with each other.

4. Results and Recommendation

In this chapter, results from the each section described in methodology are listed. Along with results, recommendations are given where appropriate. First the evaluation of features on the NAO is shown. Then the dance animation created as the entertainment application is presented. Finally, the advanced module created for NAO is explained.

4.1 Feature Evaluation

After testing many of the robots features, we found their strengths and their shortcomings. The results of our tests are shown in the table below. A green score indicates that the robot performed nearly perfectly, and a yellow score indicates the robot did not quite meet our expectations on the given task.

Table 4: Results for Basic Function Test

Task	Score
Sitting and Standing	
Face Recognition	
Sound Location	
Red Ball Movement Tracking	
Word Recognition	
Custom Functions	

This table shows each tasks respective score. The robot was exceptional when commanded to sit or stand. The face recognition and sound location were both tasks that the robot was able to accomplish, however there were some limitations. The robot was able to track a red ball without any problem. When recognizing words, the robot sometimes confused similar words, and executed the wrong command. The custom functions ran on the robot without any problems.

The first test was very simple, used to learn more about the software and how the robot responds. The test was run using Choregraphe connected to the robot. The function “Stand” was run, as well as the function “Sit”. The robot should be able to stand or sit from any position. Because the robot was initially sitting, “Stand” was run first, and then “Sit”. In order to thoroughly test the function, “Stand” and then “Sit” were run once more as shown in the figure below.

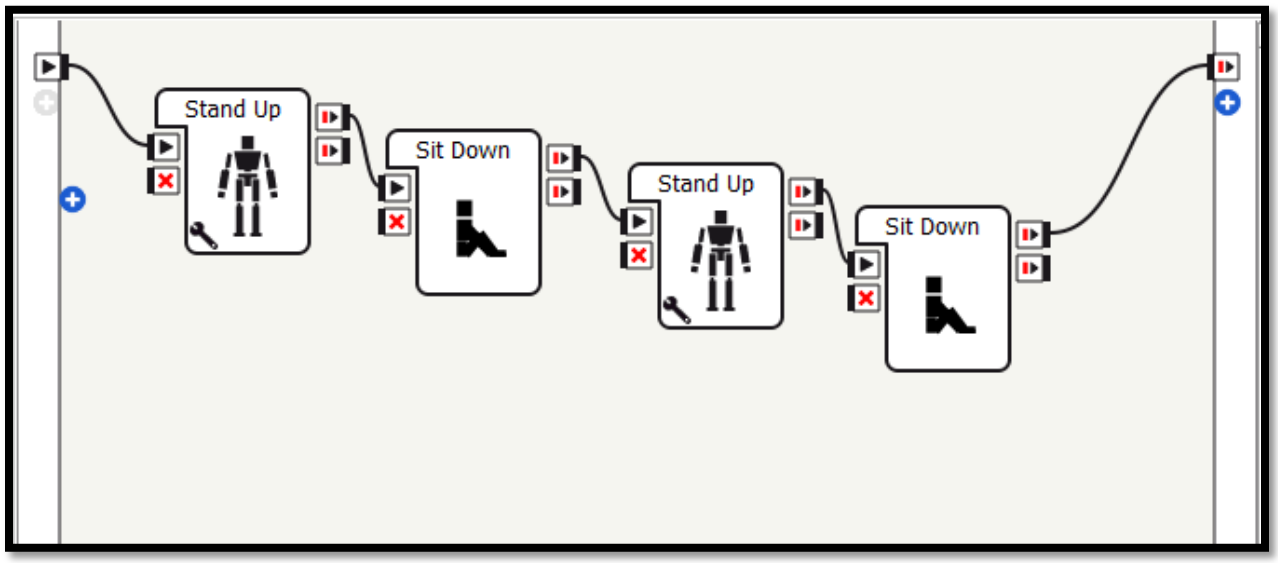

Figure 15: Sitting and Standing Test

The robot went through the steps shown in the picture, several times. Each time robot received a passing mark if it was able to successfully move to the position requested.

The next test was for facial recognition. The robot was asked to store several faces, using the method shown below.

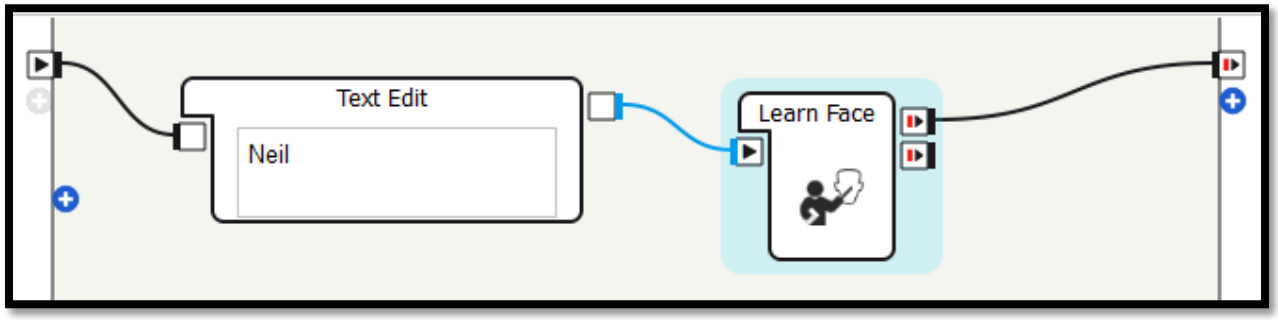

Figure 16: Method to Store Team Members Faces

After each team member stored their faces with the robot, the face recognition was tested. The method for testing this is shown here.

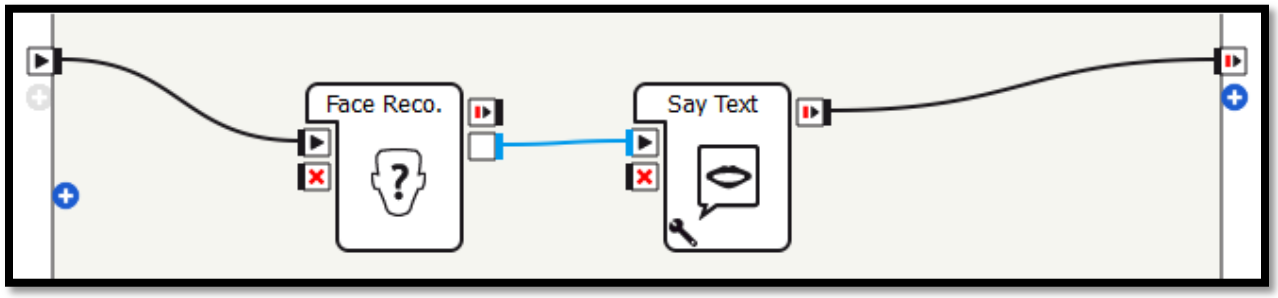

Figure 17: Face Recognition Test

The setup is very simple; each team member has their own name associated with their face. When the robot recognizes the face of a team member, it will say their name. The test was done by running this program for each team member, at different distances from the robot. The first trial was done close to the cameras of the robot, approximately one to two feet from them. The next trial was done approximately two to four feet away from the robot, and the last test

done about four to six feet away. Each time, the robot was given a successful score if it could recognize the face within 30 seconds, and a failing score if not.

The next test performed was sound locating. Once Choregraphe was connected to the robot, the sound tracking function was run.

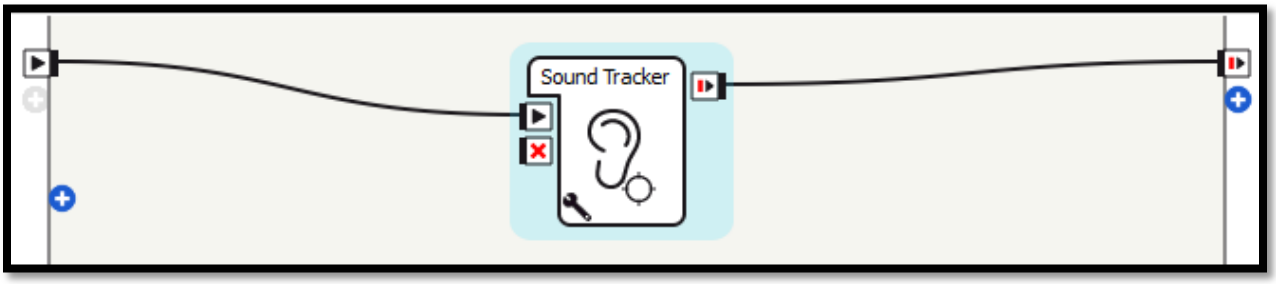

Figure 18: Sound Tracking Test Setup

Once this test was set up, the robot was placed in the middle of the room. Each team member stood close to the robot, within one or two feet. The robot should look at the sound produced. The team members clapped in turn, and the robots ability to look in the right direction was assessed. Then the members backed up to around two to four feet, and repeated the test. A third trial at four to six feet from the robot was done last. All of the results were recorded and taken into consideration when determining the best application for the robot.

The next test was for the tracking of a red ball. The robot should be able to follow a red ball, looking at it and moving toward it if it moves away.

The function was run and the robot was placed in the middle of the room. The red ball was held up about three feet from the robots cameras, and the robot looked at it. Then the ball was moved around, and the robot kept track of it by moving its head to follow the position. When the ball was moved too far in front of the robot, the robot moved forward toward it to keep it in sight. The robot did very well and always kept track of the ball.

Lastly, the ability of the robot to recognize simple commands was tested. The robot was set up to listen, and then would execute the command given.

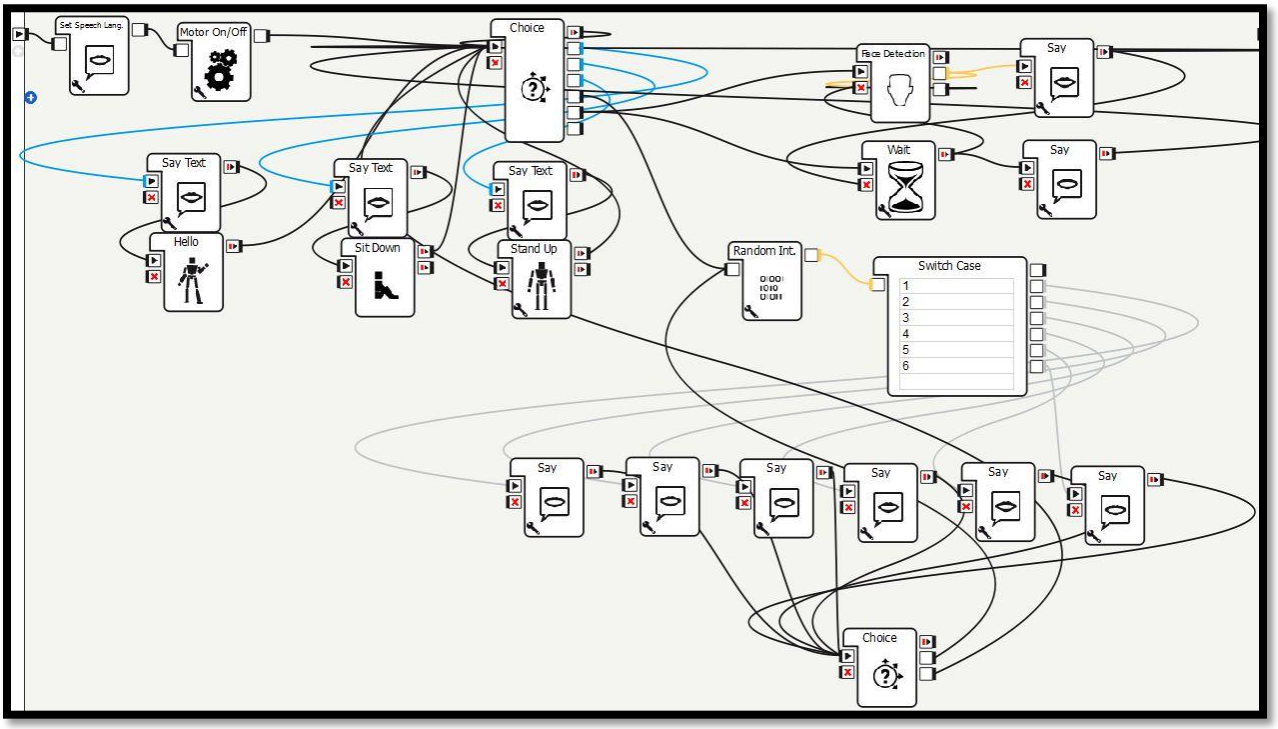

Figure 19: Voice Command Setup

The robot was given three different commands it should be able to recognize. When asked to sit, stand, or tell a joke, it would respond appropriately. Each command was said clearly and loudly, and the response of the robot was recorded.

The face recognition scored yellow because the robot would not recognize a face if it was too far away. The results of the trials are shown below.

Table 5: Face Recognition Distance vs Result

Distance	Score
1-2 Feet	
2-4 Feet	
4-6 Feet	

The robot was able to recognize a face, as long as it was not too close to the robot's cameras. The total score given was yellow, because the robot not only had difficulty recognizing a face too close, but it also took too long to find the face. The robot also had trouble if the face was not very still. When running these tests, the subject placed their face directly in front of the camera and held still until the robot recognized them, or 30 seconds had passed without recognition. The detailed results of the trials are located below.

Table 6: Face Recognition Results per person

Team Member	Distance (feet)	Time (s)	Score
Neil	**1 – 2**	> 30	Fail
	2 – 4	8	Pass
	4 – 6	12	Pass
Vicky	**1 – 2**	> 30	Fail

	2 – 4	13	Pass
	4 – 6	11	Pass
Qonny	**1 – 2**	27	Pass
	2 – 4	11	Pass
	4 – 6	15	Pass
Ken	**1 – 2**	> 30	Fail
	2 – 4	> 30	Fail
	4 – 6	17	Pass
Daniel	**1 – 2**	> 30	Fail
	2 – 4	4	Pass
	4 – 6	12	Pass

The results of the face recognition show that the robot is able to recognize a face most of the time, so long as the face is not too close to the camera. The robot was able to recognize a face 1-2 feet away within 30 seconds once, but it was still 27 seconds before it managed to. The robot was also not able to recognize the face within 4-6 feet once, but every other time, it was able to successfully.

The next task attempted was sound location. The built-in function allows the robot to turn its head toward a sound. The results of the trials are shown below.

Table 7: Sound Location Results

Distance	Score
1-2 Feet	
2-4 Feet	
4-6 Feet	

When the sound was produced very close to the robots microphones, the robot was easily able to turn its head toward the sound. However, at a further distance, the robot was not as adept at recognizing the sound. An echo or ambient noise would sometimes draw its attention, so the overall score given was yellow.

When the robot was given voice commands, we assessed its ability to recognize the words and recorded the results. These were simple one-word commands. The commands sit, stand, and joke were used, each being repeated three times.

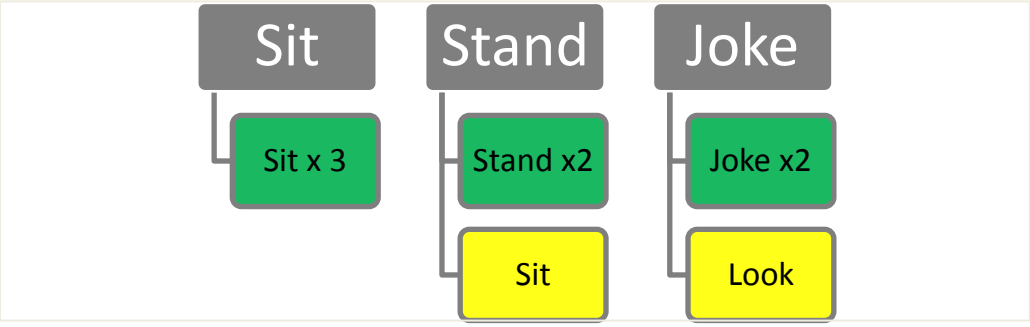

Figure 20: Words Recorded for Voice Comprehension

When asked to sit, the robot was able to correctly interpret the command all three times. However, when the command “Stand” was given, the robot only understood two of the three times. The third time, the robot interpreted the command as “Sit”. Finally, the robot was asked to

tell a joke, using the command “Joke”. Again, two times it was able to interpret this command correctly, but the third time, the robot misinterpreted the command as “Look”, another command built-in to its vocabulary. Thus, the following scores were given.

Table 8: Voice Command Results

Command	Score
Sit	
Stand	
Joke	

Sit was the only command correctly recognized all three times, and so it is the only green score. The rest were misinterpreted one out of three times and are given a score of yellow. The word recognition is given an overall score of yellow, however it should be noted that the robot was able to understand the commands correctly most of the time.

4.2 Dance Animation

The song used for the dance is three minutes, 20 seconds in length. This means that the dance moves had to be repeated approximately two times each. The original dance repeats the moves many more times, however our robots are not able to perform the moves nearly as fast as a human would. The dance was synchronized so that each different animation would flow without any awkward transitions between them. There were 150 animations used in total.

Figure 21: One Frame from Dance Animation

The original dance was divided into 5 parts so to as divide the work evenly between 5 team members. These five parts were extracted from the following video: http://v.youku.com/v_show/id_XNzI4NzQ0NDU2.html?from=y1.2-3-95.3.2-1.1-1-1-1

As a result of being a clear cut dance as well as an instructive one, this video fit the need for the robot's own dance. This dance included repeated section with fairly distinctive dance parts. This video was separated into five different dances at the following parts:

Dance one = 0:05 – 0:16

Dance two = 0:17 – 0:24

Dance three = 0:30 – 0:40

Dance four = 0:48 – 1:03

Dance five = 1:03 – 1:17

After each dance was completed, the different parts were put together. The [insert figure] shows the completed choregrahe file for the dance. The file itself contains several parts. These parts are preparation, parallel functions, dance repeats, and post-preparation. These parts are important in order to made improve robot safety and entertainment experience.

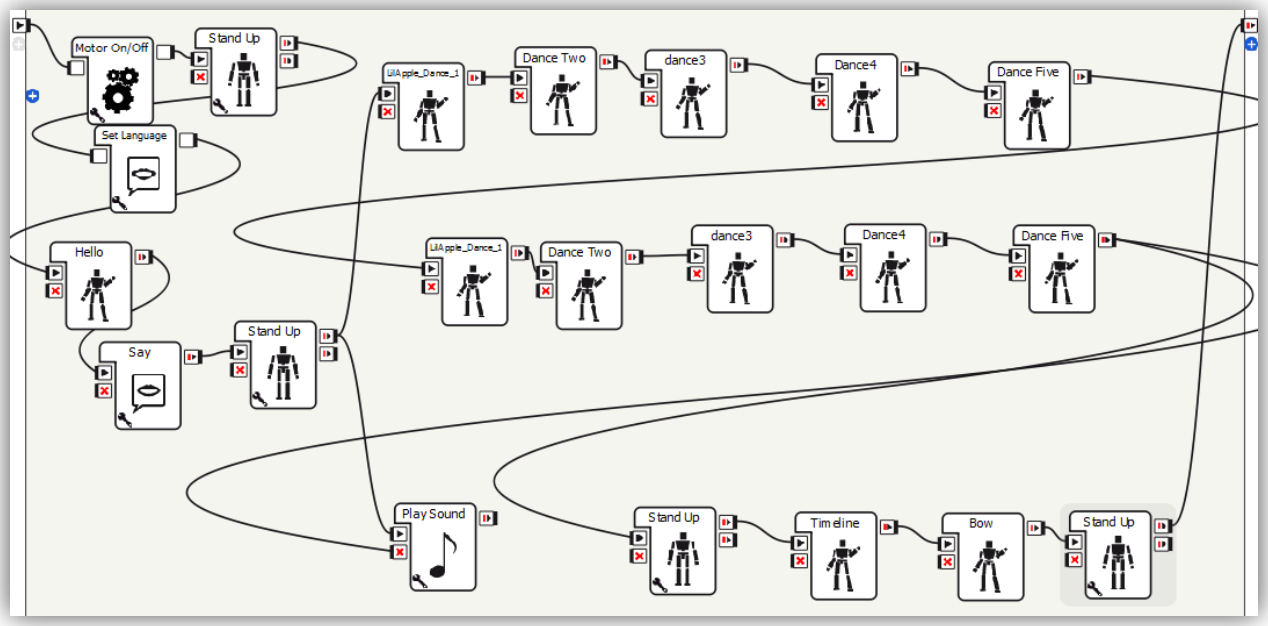

Figure 22: Final Dance Choregraphe File

The first part of the final file is preparation. This part makes sure everything is correctly setup for the dance to begin. Without preparation, the robots could start in awkward position, resulting in failure of the dance or damage to the robots. In addition, motors are properly turned on and correct language set in this section. Finally, a custom introduction is also input for better introduction. Second part of the final dance is parallel function. This is the music that plays alongside dance animation thus giving a more inclusive experience. These parts are shown in Figure 23 and Figure 24 respectively.

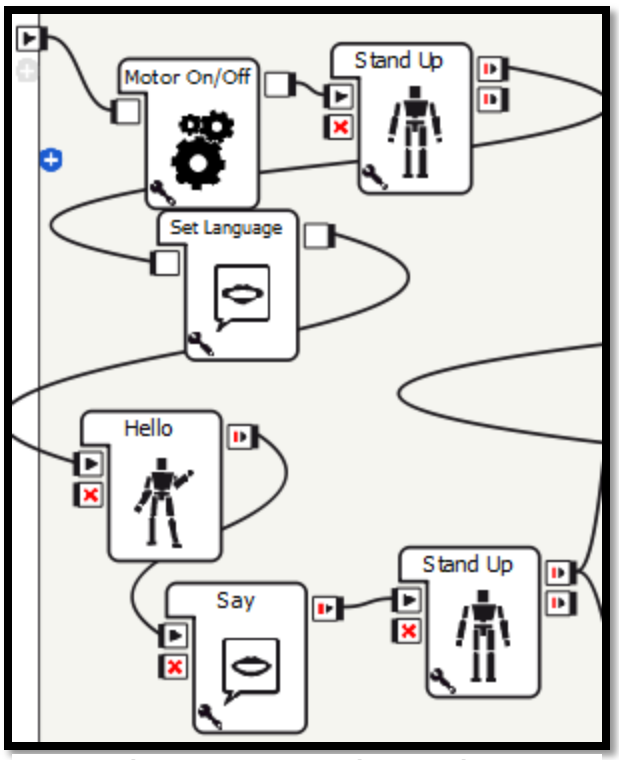

Figure 23: Preparation Section

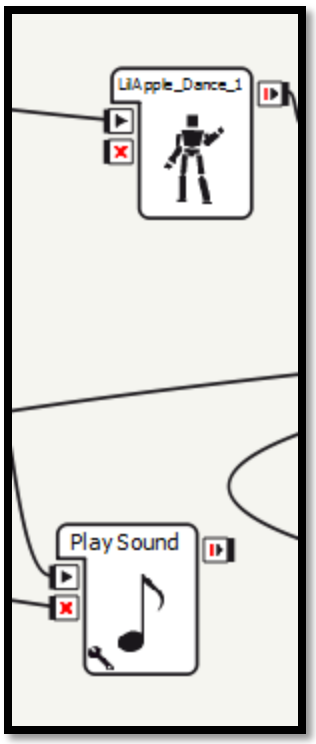

Figure 24: Parallel Function Section

The final two parts of the dance are dance repeats and post-preparation. First the dance is 3 minutes long but contains two overall repeated parts. That is after minute 1:17 the dance repeats itself in another minute and a half long cycle. As a result, this is reflected by the robots as well. The original five animation parts of the dance are repeated twice. Post-preparation section is similar to preparation. It includes ways to properly turn off the motors of the robot and put the robot in a safe and resting position. This ensures no sudden fall from ending position and is generally a safe practice. Finally a concluding bow and statements are also inserted into this section in order to make an custom exit. These two sections are shown in Figure 25: Dance Repeats Section and Figure 26: Post-Preparation Section.

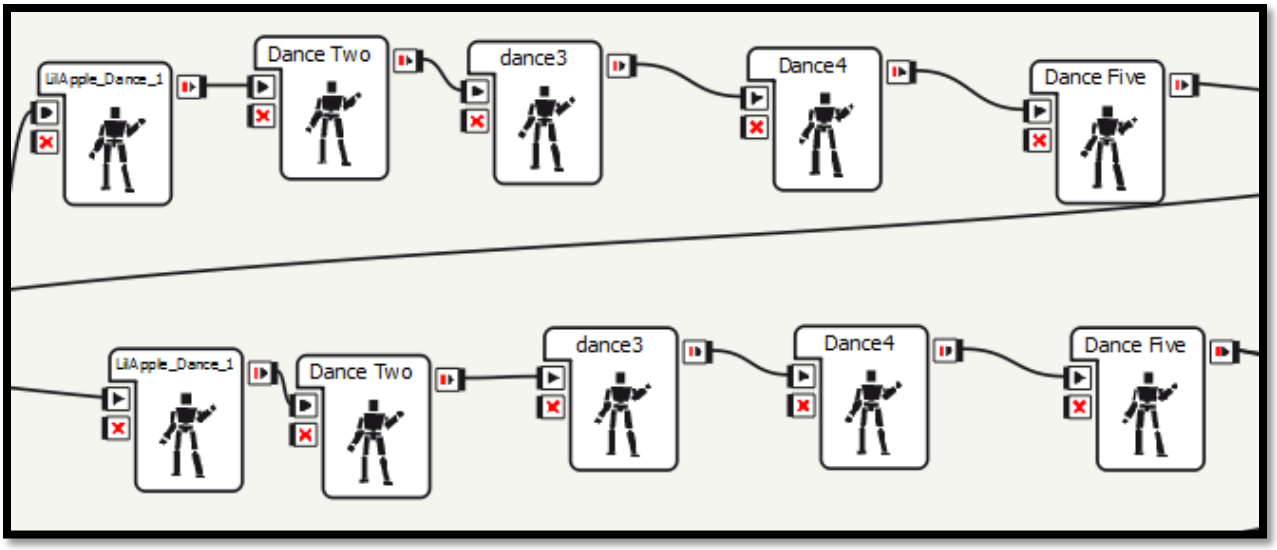

Figure 25: Dance Repeats Section

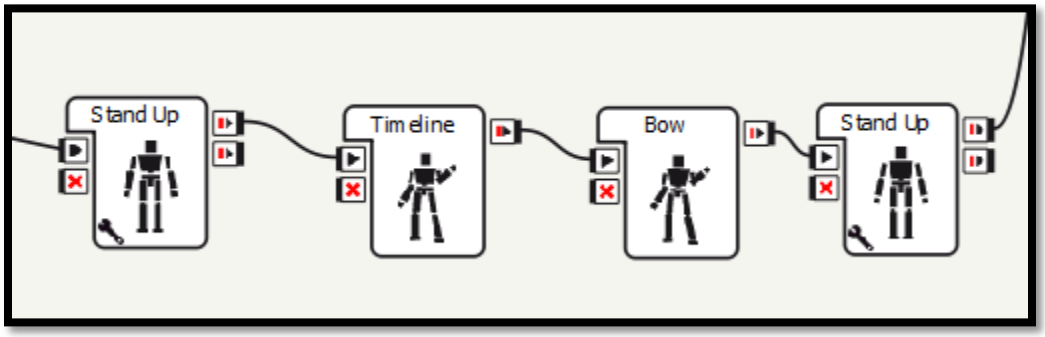

Figure 26: Post-Preparation Section

4.3 Handshake Module

Before the process to write code in C++ was started, the C++ environment for NAO had to be setup. This required procuring and installing several programs for the C++ SDK. The steps for setup are complicated and cumbersome. There are 5 programs required to setup the C++ SDK:

- Visual Studios 10 – 32 bit.
- CMake 2.8.10 – 32 bit.

- Qibuild 1.14.2
- Python 2.7
- NAOqi-sdk 1.14.2

These were all the functional programs that were required for Version 1.12 NAO robots. While Aldebaran provides tutorials for such program setup, there is a severe lack of detail for the setup process. To make matters worse, the troubleshooting in the setup process is not easily accessible. As limited time was available, when setting up the C++ SDK, the process was nearly abandoned after one week's effort. After installing all the required programs and running test functions, an error was detected. The libraries provided for NAO were not able to be linked for a program. As a result, compilation failed.

This caused considerable delay and many solutions were attempted. It was possible that the wrong version of windows was used in these programs thus using a Virtual machine with the correct windows was attempted. However, there would be no internet connection through Wi-Fi available on Virtual Box which is essential for communication with the robots. Second attempted solution based on earlier assumptions was to run Windows 32 bit on a USB or external hard drive. However USB was not viable and an external hard drive would be too expensive. The final failed solution was to write the code directly on the robot. The robots themselves did not have a compiler and as such running any non-tested code would create unwanted complications. After searching for several days, it was found that the CMake version was incorrect (not listed on Aldebaran tutorials). The correct version was listed earlier.

After initial setup, the handshake module was written entirely in C++ using the libraries provided by Aldebaran. These libraries have complete set of instructions that encompass the different sensors and movements NAO robots are capable of using. All of the programs written for NAO's are done using these libraries. Thus, the handshake module utilized content of these libraries to accomplish a new module for NAO.

The procedural steps involved in setting up the handshake procedures are simple. However it is the intricate details of correct usage and situational differences that make the program a challenge. There are several rules that must be followed before a program can properly function on the robots. These rules are there to make sure that program does not cause preventable crashes on the robots. These rules are different for different applications and it takes time to learn to utilize them properly. These rules can be seen i

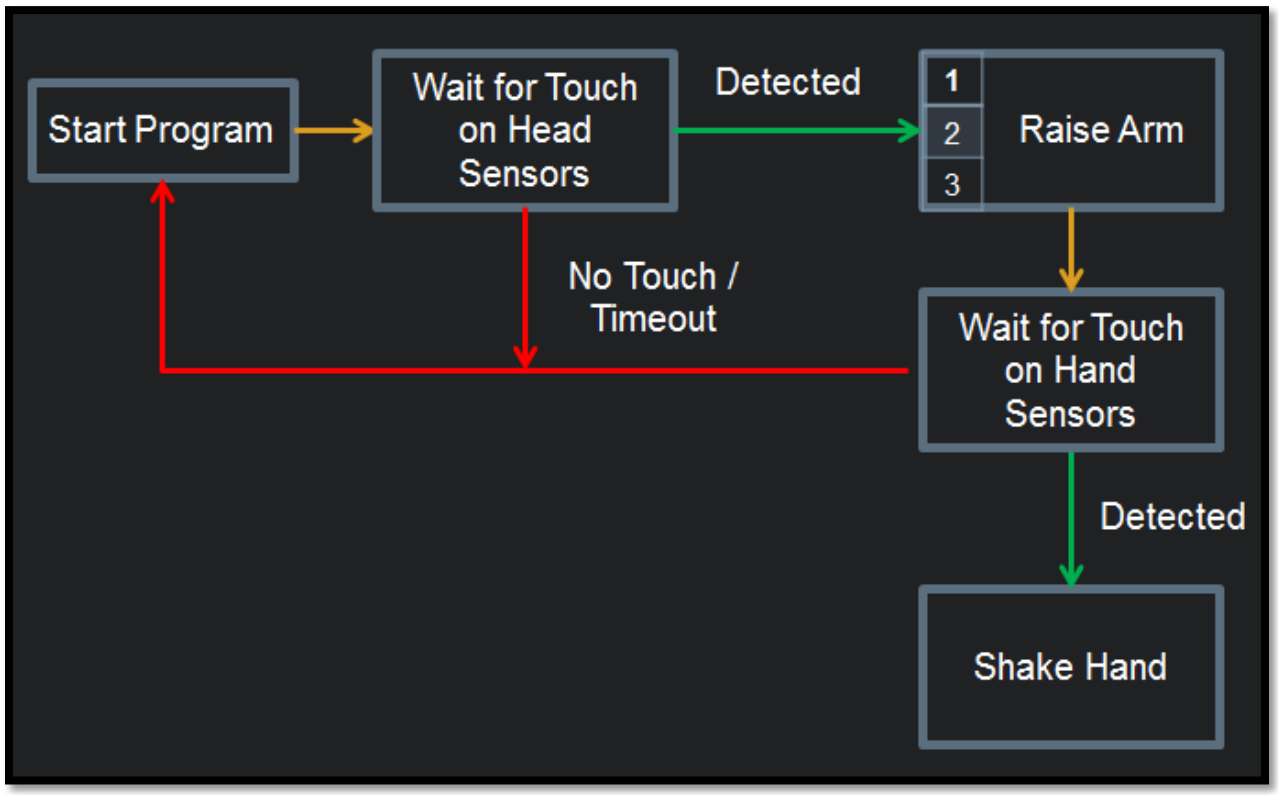

Figure 27: Handshake Flowchart

Before the process to write code in C++ was started, the C++ environment for NAO had to be setup. This required procuring and installing several programs for the C++ SDK. The steps for setup are complicated and cumbersome. There are 5 programs required to setup the C++ SDK:

- Visual Studios 10 – 32 bit.

- CMake 2.8.10 – 32 bit.
- Qibuild 1.14.2
- Python 2.7
- NAOqi-sdk 1.14.2

These were all the functional programs that were required for Version 1.12 NAO robots. While Aldebaran provides tutorials for such program setup, there is a severe lack of detail for the setup process. To make matters worse, the troubleshooting in the setup process is not easily accessible. As limited time was available, when setting up the C++ SDK, the process was nearly abandoned after one week's effort. After installing all the required programs and running test functions, an error was detected. The libraries provided for NAO were not able to be linked for a program. As a result, compilation failed.

This caused considerable delay and many solutions were attempted. It was possible that the wrong version of windows was used in these programs thus using a Virtual machine with the correct windows was attempted. However, there would be no internet connection through Wi-Fi available on Virtual Box which is essential for communication with the robots. Second attempted solution based on earlier assumptions was to run Windows 32 bit on a USB or external hard drive. However USB was not viable and an external hard drive would be too expensive. The final failed solution was to write the code directly on the robot. The robots themselves did not have a compiler and as such running any non-tested code would create unwanted complications. After searching for several days, it was found that the CMake version was incorrect (not listed on Aldebaran tutorials). The correct version was listed earlier.

In order to build an intuitive handshaking procedure, it needed some feedback from the user. These two stages of feedback are shown in the figure above. When the program begins, it will not begin the motions of the handshake until user touches one of the three tactile sensors on the robots head, or if a pre-determined wait period expiries. Once these sensors are touched the raising arm procedure is set into motion. This motion raises the arm of the robot to predetermined height. At that point, the robot waits for a touch is detected on one of the three hand sensors. The hand-shaking motion is not dependent on which of the three hand sensors is touched, as all three are unreliable. The following table shows the average time it takes for sensors on head and hand of the robots to register. It is important to note that after 2 minutes, attempts to get a signal from the head or hand sensors are ceased.

Table 9: Average time for Sensors to Register

	Average Time (Max 120 mins)
Head Sensors	0.5 seconds
Hand Sensors	52 seconds

These hand data from hand sensors reflect a grim reality where tactile hand sensors for most of the robots are non-functional. This proves quite a challenge when trying to design a system based on feedback from the user. As a result, a failsafe is necessary so as to relieve the system when no-feedback is received. These fail-safe are set in forms of 20 and 10 second timeout for head and hand sensors respectively. The 10 second is recommended time after which the program will be restarted or move to completion depending on which user desires (this feature is currently set to restart the program). The 20 second timeout for head sensors is

recommended because during that period the robot will mention that user needs to press one of its head sensors to start the program. In addition, three sensors on head correspond to different levels to which the NAO robots will raise their arm which will also need to be mentioned by the robot. These arm levels and their angle with respect to 0° (arms pointing straight down to the ground) are listed below the Table 10.

Table 10: Three Different Arm level

Sensor Level	Arm Level	Angle of Arm (Degrees)
Front Head	Low	0°
Middle Head	Medium	-25°
Back Head	High	-50°

With these features, it is important to explain the main working parts of the C++ program. This program is controlled by the main function, which takes in the robot's IP address as an input argument. At this point it begins an infinitely looping function call with multiple working parts. Initially, this looping function begins the program and checks the three tactile sensors on the NAO robot's head using the check_touch(). If no touch is detected, it will begin the program again after a 10 second wait period. If a touch is detected, then it will call raise_arm() function to raise the NAO robot's arm to one of the three selected levels. After this, it will check the tactile touch on the either left or right hand by calling check_hand() function. If no touch is detected, it will again restart the program. Otherwise, it will move forward with hand shaking using shake_hand() function.

In order to check the correct sensors, check_touch() function requires four arguments: the name of the three head sensors, and the robot's IP. The names of the head sensors give

the function the correct sensors to poll for. Polling occurs through the function accessing the NAO robot's memory for a given sensor. This memory bank can be accessed only if the IP address of the robot-in-interest is known. The function check_touch polls for all of these sensors sequentially, with whichever sensor is sent as first input polled first. As a result, if the user presses all three sensors at the same time, only data from the first sensor will be recorded.

The second function called by the main program is raise_arm(), which first makes sure robot is not in an unwanted position by making it stand up. Function raise_arm() then sets an end position for all six of NAO robot's arm joints. Along with end positions, either left or right arm needs to be set, as well as the speed with which to reach the end position (1 being max and 0 min). Arm height is different based on the input for raise_arm() which can be specified as low, medium, or high.

The next function used by the main program is the check_hand(). It uses polling methods and memory reading similar to the check_touch() function from above. In that sense, the check_hand() function requires the desired arm side and IP address. From there, it checks all three sensors on the desired arm sequentially. If signal is detected on any of the sensors on the hand, the outcome is to proceed with the shaking motion. This is done with the use of the shake_hand() function. It accepts arm side, arm height, and NAO robot's IP address as arguments. With this, it uses methods similar to raise_arm() function to set two different end position for shaking hand. These two end positions are then repeated two times to get the desired hand shaking motion.

This finishes up how the hand shaking program works. It is based on retrieving data from NAO robot's memory, and accessing of sensor data in real time, and controlling individual joints at desired speed to gain an acceptable outcome. The overall length of the program is roughly 450 lines. This program has been tested on multiple robots, with nearly identical results, except

when it comes to the reliability of the hand sensors. Many times the hand sensors simply failed to register a signal leading to shortened programs.

5. Bibliography

Aldebaran-Robotics. (2012). *Available languages.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/family/robots/languages.html

Aldebaran-Robotics. (2012). *Contact and Tactile Sensors.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/family/robots/contact-sensors_robot.html

Aldebaran-Robotics. (2012). *LEDs.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/family/robots/leds_robot.html

Aldebaran-Robotics. (2012). *Microphones.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/family/robots/microphone_robot.html

Aldebaran-Robotics. (2012). *NAO Linux C++ Development Tutorial Part 1.* Retrieved December 2014, from Aldebaran Community: https://community.aldebaran-robotics.com/resources/tutorial/Cplusplus-Tutorial-Part1/

Aldebaran-Robotics. (2012). *NAO Motors.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/family/robots/motors_robot.html

Aldebaran-Robotics. (2012). *NAO Technical Overview.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/index.html

Aldebaran-Robotics. (2012). *NAOqi Framework.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/dev/naoqi/index.html

Aldebaran-Robotics. (2012). *SDKs.* Retrieved December 2014, from NAO Software 1.14.5 documentation: http://doc.aldebaran.com/1-14/dev/sdk.html

National Instruments. (2014). *One Platform.* Retrieved December 2014, from NI Labview: http://www.ni.com/labview/why/