NAO Robot Demos for STEM Robotics Education
Colorado School of Mines

Client: Dr. Hao Zhang

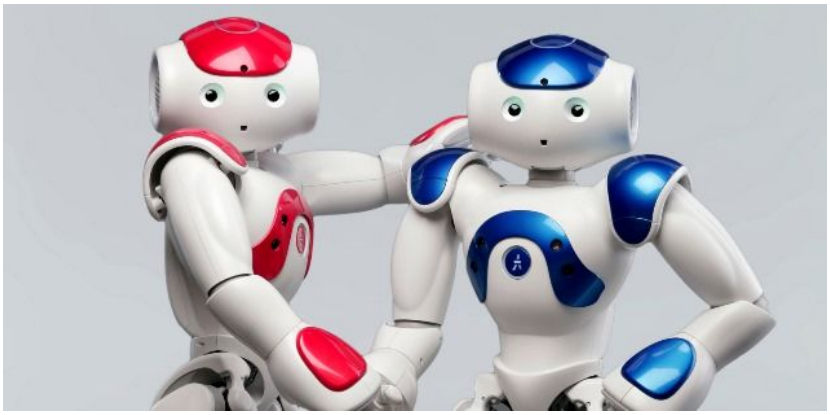
Project Team: John Spielvogel
Austin Leo
Zach Smialek
Jacob Maerli

Contents

1.0 Introduction

1.1 Product Vision

The objective for this project is to design and implement functional software to interact with and demonstrate the capabilities of Aldebaran NAO robots. The demonstrations will be targeted towards middle school and high school students to promote STEM robotics education and to expose the general public to the potentials of robotics.



http://www.robotlab.com/hs-fs/hub/314265/file-1052596453-jpg

The vision for this project was to incorporate games that the student could play with the NAO robot. The goal of these games would be to help get the next generation of brilliant minds excited for robotics and hopefully spark curiosity in at least a few of them. The product was envisioned to include three demonstrations which would demonstrate the capabilities of the NAO robot. One demonstration was to exhibit the physical mobility. The second would show the potentials of utilizing computer vision. The third would present a situation in which the player and the robot would be involved in an active dialogue.

2.0 Project Requirements

2.1 Functional Requirements

The functional requirements for this project were divided into two sets of deliverables. The first contained three demonstrations that the client requested to

be implemented including: a dance, tracking and kicking a red ball, and a dialogue between a person and the robot. If time allowed, the second deliverable requested was to interface the NAO robot with a Microsoft Kinect and port existing code for a Simon Says game over to the NAO robot. In order to meet all project specifications a video demonstrating the deliverables must be included.

2.2 Non-Functional Requirements

The non-functional requirements for this project entailed the upkeep of good coding practices, utilizing either the Robot Operating System (ROS) or Choregraphe, and developing the project in either Python or C++. In order to produce satisfiable code, the code is to be completely documented and implementations must be developed in an extensible and modifiable manner which can be understood by future developers who may extend the demonstrations.

2.3 Potential Project Risks

Developing software for a highly mobile piece of hardware such as the NAO robot must be done with the utmost care. NAO's anatomy contains many joints and motors controlling every component. If programming is not done in small incremental steps the robot can easily damage himself or others. There are several components which prove to be much more fragile than the rest including the back of NAO's head as well as his fingers. In order to safely work with NAO it is advised that there is at least one person spotting NAO while another executes code or that NAO is suspended via a baby harness.

Figure 1: Broken Finger

3.0 System Architecture

3.1 Core Components

The main software running on and giving control over the NAO robot is called “NAOqi”. The NAOqi Framework provides developers with the framework needed to program the NAO robot. The framework is cross-language and cross-platform allowing for the creation of distributable software applications. This means that code developed for one Aldebaran robot will be executable on another robot as the application programming interface (API) calls remain the same.

In addition to utilizing the core software and framework provided with the NAO robot, the system architecture includes the three core demonstrations in the form of Python scripts developed by the team. These scripts are individually packaged and consist of a single Python file each, excluding the Trivia Game demonstration whose implementation is more complex.

3.2 System Package

The system package in this project is a server client model where the robot is the server and the client is a computer that runs the scripts. Once the script is run on the local machine, the broker allows communication across the network and the ability to call the modules of the API (Figure 2). The system package defines the local machine as the client and the server as the NAO robot.

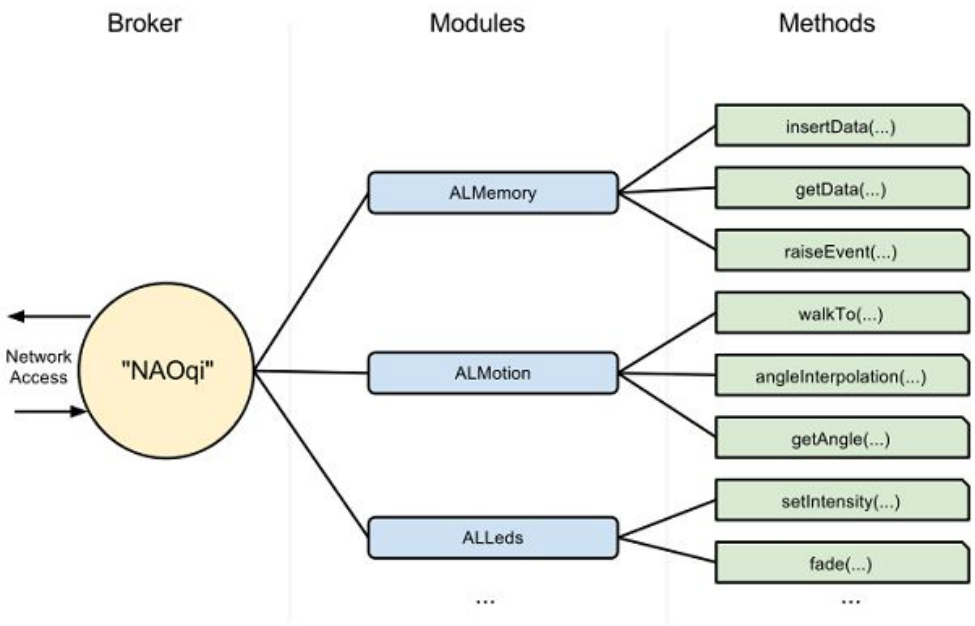


Figure 2: Interaction with NAOqi Services

3.3 Use of Software Package

This section is intended to aid the user running the scripts. The first things that has to happen is the changing the IP of the robot. The robot network interface is dynamic host configuration protocol (DHCP) so the IP will vary. The IP is set either by getting passed into main or using the argparse, the port should not have to change. The next step is to adding the NAOqi directory to the PYTHONPATH, see the **install guide** for more details. After the path has been set and network setting has be alter the script can be ran through the terminal and the robot will begin to execute the script.

4.0 Technical Design

4.1 Trivia Game

The trivia game is designed to be playable either solely by a single individual or with a group of players and proves to be a fun way to increase your knowledge. The game begins by having the NAO robot ask a couple of game

initializing questions such as “How many players will be playing in this game?”. Following initialization of the game, NAO begins iterating over the user defined number of questions which are pulled at random from a database. NAO reads the question out loud followed by the multiple choice options A, B, C, and D. After a question is read and multiple choice options are given, NAO will iterate over the players in the game asking each one for their final answer. Each player is asked to confirm their answer, if unconfirmed, NAO will ask for a new answer to the question. Once all players have supplied an answer to the active question, NAO will say what the correct answer is for the current question. This process is repeated until the user defined number of rounds has been reached. At the end, NAO states every player’s score and determines the victor.

The trivia game is designed using three classes as well as the help of the provided NAOqi API. The API is utilized for the creation of the broker. The broker is used to create the proxies that allows the developer to access specific services such as the speech recognition engine. The Game class contains the majority of the programs logic. The Game class’s constructor first executes a method which greets the players and asks some game initializing questions such as “How many rounds do you want your game to last?”. Within this method, a proxy is created to the Animated Speech service, which NAO uses to speak out loud while providing basic contextual based animations. After the game’s parameters have been set (number of players and rounds), the next method uses the prebuilt database (discussed below) to populate a localized list of questions with their respective multiple choice options and answer. After the question bank is initialized, the program enters the main game loop which it does not exit out of until the game has reached the user defined number of rounds to play. Within a single iteration of the game loop, the program randomly picks a question from the question bank. Reads the question and its associated answers out loud and then loops over all the players asking them for their answers. The program creates a proxy to the Speech Recognition service which allows NAO to understand specific keywords

that have been added to his vocabulary list. Upon turning speech recognition on, NAO's eyes begin flickering blue (to notify the user it is listening). Once NAO hears a word he/she understands the WordRecognized event is triggered and the is_recognized() callback method executes. It is very important that speech recognition is turned off immediately after user input has been accepted since NAO's own voice can trigger recognized words.

Due to the trivia game being one of the more complex demonstrations, it would be best to abstract out functionality into multiple classes in order to maintain readable, maintainable and extendable code. In addition to the Game class, there is the BuildDatabase class and the ASRDialog class (Figure 3). The BuildDatabase class handles the initialization of the database as well as importing records into the database from a pre-existing CSV (comma delimited file). The ASRDialog helper class sets up a proxy which provides access to the speech recognition service. This class also provides helper functions which provide an easy way to turn off and turn on the speech recognition service from anywhere in the project.

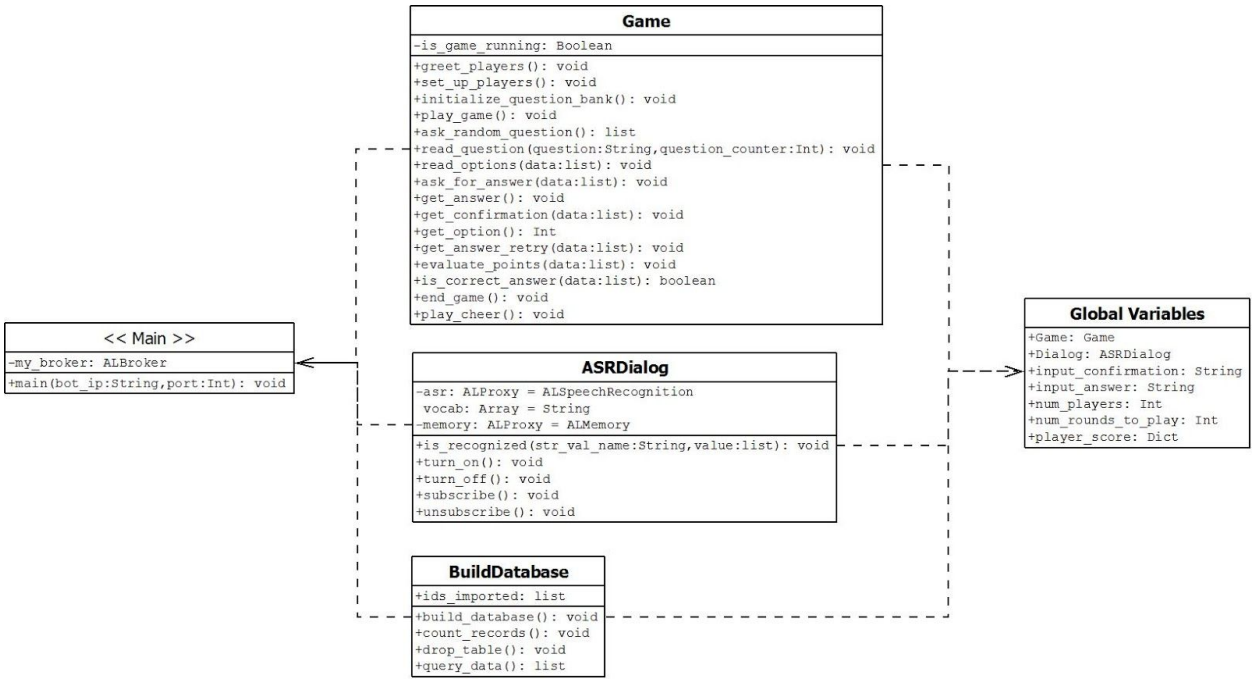


Figure 3: UML of Trivia Game

4.1.1 Database

The database is a simple table with multiple choice answers, questions, and the real answer (Figure 4). The decision to use a database over reading in a comma delimited file for certain reasons. An excel/csv file is used to store the data in order to make it easier for future developers to easily add or update questions. In addition, a simple query of all the required information is returned in an easily referenceable list. The limitation of this design decision is that it is dependent on the structure of the comma delimited file. The question, answer and multiple choice answers needs to be in a specific cell in the file.

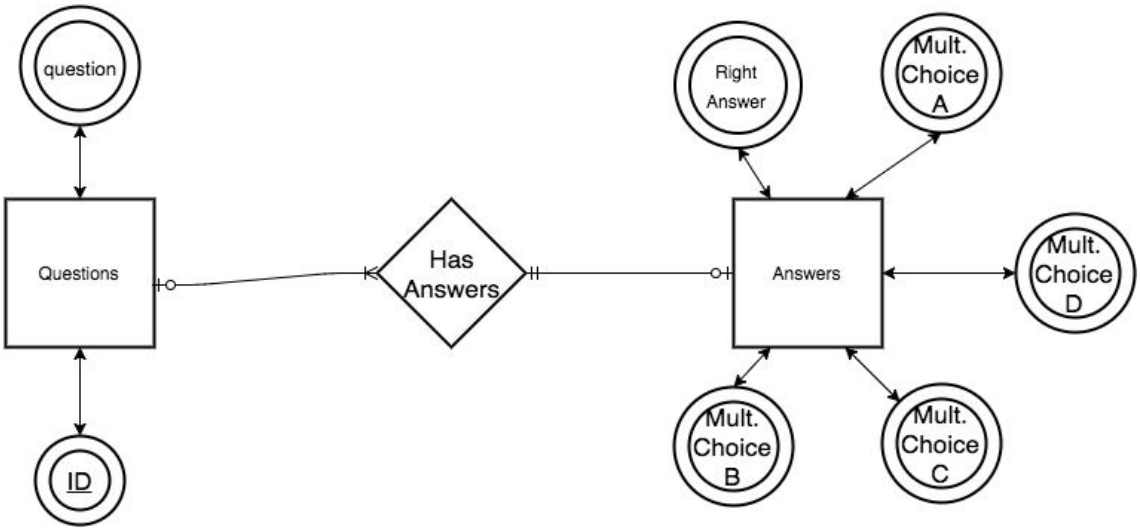


Figure 4: Entity-Relationship Diagram

4.2 Red Ball Tracker

This demonstration illustrates the robot's ability to track an object. The robot defaults to using the upper camera which has trouble keeping objects in his field of view when NAO approaches them. By setting the lower camera as the active camera NAO had a more accurate alignment with the ball. A limitation of the tracker is it requires the diameter of the red ball to be hard coded into the script prior to execution.

Once the red ball is found in the environment, a vector is calculated describing where the ball is in the real space and NAO's relative position to the ball. The robot starts to move to the ball after a relative position is set with the frame of reference to the torso the robot. Once the relative coordinate position is satisfied the kick is started. The kick was programmed manually by adjusting each joints/motors angle. After the kick has been completed, if the ball is still in view the robot will go for another kick.

It is unclear from the documentation how the tracking is implemented in the API. There are a few common ways object tracking can occur, common target representation and localization algorithms. To popular flavors are mean-shift tracking and contour tracking. Since the robot has the concept of what a red ball looks like it mostly implements common target representation methods.

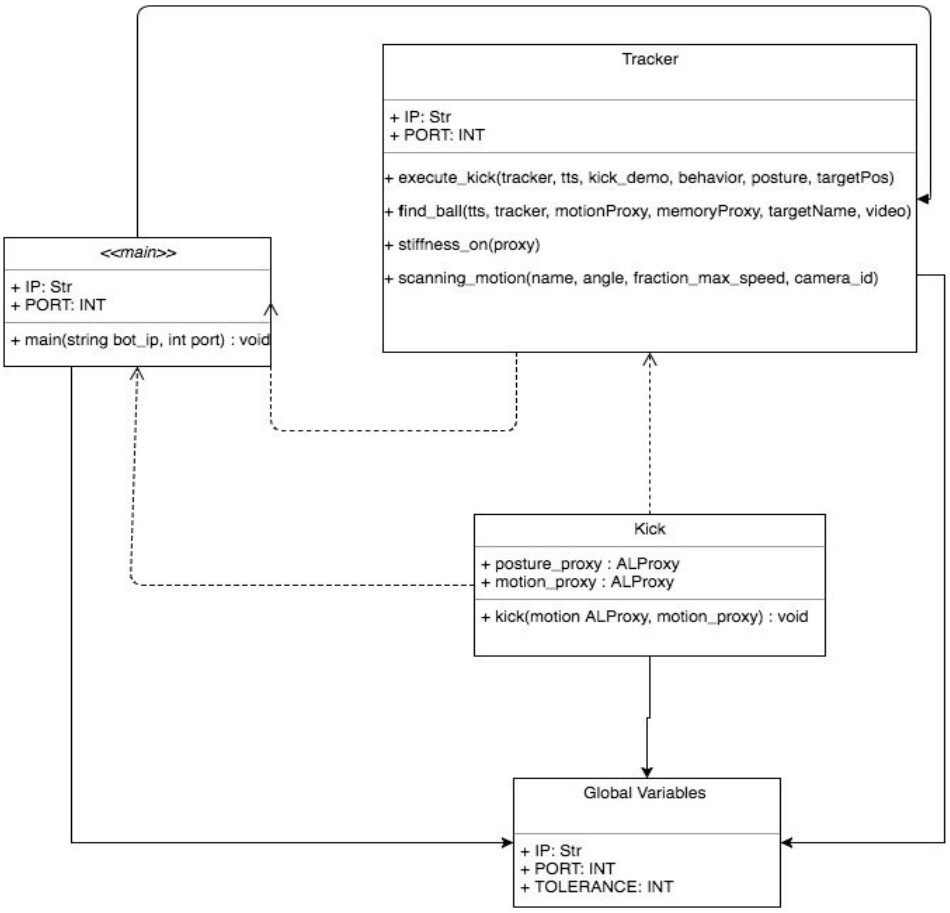


Figure 5: UML Diagram of the Tracker demo

4.3 Dance Demonstration

The dance demonstration was animated with the use of Choregraphe. Choregraphe is a powerful software tool that comes with NAO which can be used to manage movement, speech, and sensory input. Within the software the user creates 'flow diagrams' of boxes to control the behavior of the robot. There are three basic types of boxes that are used: Python, Dialog, and Timeline. Python boxes execute a script of Python code when their input signals are triggered which control NAO through the NAOqi API. Dialog boxes define a set of rules by which NAO can speak, recognize speech topics then send corresponding output signals. Timeline boxes are used for managing actions which require specific execution timing.

The dance demonstration is orchestrated using a Timeline box to the beat of *Harder, Better, Faster, Stronger* by *Daft Punk* at a tempo of 128 beats per minute. Within the box a motion timeline of numbered frames is displayed at the top with a line of behaviors below as seen in Figure 6.

Figure 6: Timeline

Along the motion line, joints of the robot can be stored which hold the position angle values of each actuator (i.e. HeadPitch, LArmPitch, RAnkleRoll). The timeline can be segmented into keyframes which can be used like individual behavior boxes. When the timeline is executed a marker moves along the motion and behavior lines at frames per second speed defined by the user. As the marker approaches a stored pose NAO will automatically move its motors to reach the pose when the marker hits it. When the marker reaches a new

keyframe a start signal is sent to the behavior box. The frames per second for this demonstration was set to 32 which calculated out to 15 frames per beat which allowed the motions to be saved synchronized with the song.

There are two methods to record and animate NAO's joints. First, the value of each actuator can be edited through Choregraphe by clicking the actuator in the Robot View window which will bring up a model of the actuator with a display of their values. The values may be modified using their corresponding slider or a numeric value may be entered as seen in Figure 7.

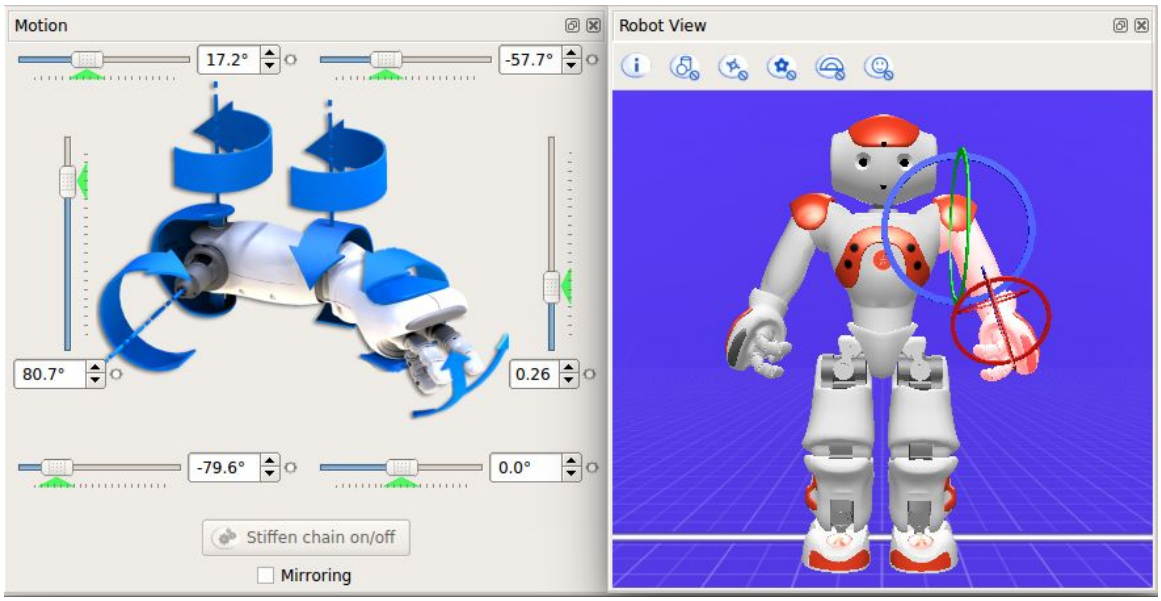


Figure 7: Motion Control

Second, Choregraphe has an Animation Mode which allows the user to release the motors of the joints by touching NAO's tactile sensors and move the joints. Once the joints have been moved to their desired positions, they may be saved in the timeline. The animations for the dance movements were all done individually using a combination of these two methods.

5.0 Design Decisions

5.1 Language Comparison

The NAO robot can be programmed utilizing many different languages. Specific languages prove to be better suited for the project (Table 1). Python was chosen because of its ease of use and time efficiency (in comparison to C++'s required overhead). Furthermore, the majority of NAO's tutorials and documentation are more prevalent for Python. Java does not meet the requirements due to the unnecessary step of cross-compiling down to C++ or Python.

Language Comparison					
	Ease of Use	Online Documentation	Time Efficiency	Requires Cross-Compilation	Total Score
Python	5	5	5	5	20
C++	3	5	2	5	15
Java	5	1	4	0	10

1. Values range from 0 (worst) to 5 (best)

Table 1: Weighted comparison of potential languages

5.2 Framework Comparison

Programs written for the NAO robot can be executed using two different frameworks: Choregraphe and the Robot Operating System (ROS) (Table 2). ROS is a command-line based framework which provides no GUI and has a much steeper learning curve than Choregraphe. Choregraphe is a GUI based framework that allows new users to quickly jump into writing functionality for the NAO robot. Although ROS is more versatile for programming a wider array of robots, Choregraphe was used due to time constraints of the project. Another reason Choregraphe was chosen was that it easily integrates Python scripts. Furthermore, the majority of tutorials and online documentation for the NAO robot use Choregraphe or Python scripts which can be added to Choregraphe with ease.

Framework Comparison					
	Ease of Use	Online Documentation	Time Efficency	Versatilty	Total Score
Choregraphe	5	3	5	1	14
ROS	1	1	0	5	7

1. Values range from 0 (worst) to 5 (best)

Table 2: Weighted comparison of potential frameworks

6.0 Results

6.1 Accomplishments

Over the course of the project three STEM demonstrations were completed, the first being a choreographed dance to the music of *Daft Punk's Harder, Better, Faster, Stronger*.

The next demonstrations that was implemented was tracking and kicking a red ball. The robot searches the environment for a red ball. Once the ball is found it will move to a relative position away from the ball then the robot will kick the ball. After the kick the robot then will search the environment again for the ball.

The last demonstration is a trivia game that highlights the dialogue abilities of the NAO robot. Questions and multiple choice answers are read by the robot that then waits for the players to answer. NAO keeps track of answers supplied by the players and determines which players were correct. The robot keeps track of the score, and at the end of the game winner(s) are announced.

6.2 Shortcomings

The optional set of deliverables was not completed due to time constraints and the learning curve. This tier of deliverables included the implementation of Simon Says using the NAO and a kinect, as well as adding to the tracking and

kicking demonstration. With regards to the kicking demo, kicking the ball at a target and searching for the ball if it is not in NAO's field of vision were explored but also not implemented.

6.3 Potential For Expansion

Each demonstration is extendable and can be used as a guide or baseline to create a new program for the NAO robot. For example, the trivia game can be extended into a more complex game by adding players, rules, new questions, or a number of other attributes depending on the desired game structure. Implementation of better sound localization to boost hearing of the answers, like looking at who's answering the question.

The kicking a red ball demonstration has the potential to be expanded by adding in the searching for a target to kicking the ball at the target. Another potential is to optimize the kick the robot performs. Using gyroscopic data to dynamically execute a kick by changing the speed of the kick or repositioning the foot in the better position. Adding the ability to track a different color or size ball would add versatility to the demonstration. The hamstring of this demonstration that ball has to be red and roughly 0.07m in diameter.

The dance demonstration could be expanded by adding more sequences to the routine. Using center of mass data add more striking moments mainly in the legs. This would be found using the built in sensors of the robot. This information gets logged when the program subscribes to a specific memory location. The data could help correct the balancing and inertia problem.

6.4 Lessons Learned

- Vigorous testing is essential to a successful system. Unfortunately, not all NAO demonstrations are able to be unit tested as some require hands-on user interaction (e.g. voice). However, every situation should still be tested because each can have its own unexpected results on the overall project.

Even if the tests pass the first time, they should be retested again and again. Changes made in one segment may not cause it to break, but it may affect the functionality in another area.

- The results that are coded for will not always come out the way they are expected. Environmental factors heavily impact the limitations of robotic sensors. During the early stages of testing facial and voice recognition, the lighting and background noise would cause significant problems with the expected results. Even if the code worked as intended in a controlled environment, changes had to be made to compensate in the future implementations.

- The learning curve for the robot operating system (ROS) was steep and a challenge to even get the enviroment setup. The documentation was out of date, tutorials worked half the time, and forums were hard to come by. The lesson here was that change happens often and more often than not things will not go smoothly. Additionally, the majority of NAO tutorials / documentation rely on utilizing the Choregraphe software or interfacing with Python.

- Getting familiar with the documentation of the API should have been the first step of the process. It would have been worth spending a week just learning the API and would have saved time during the coding process.

- NAO was recently (last two years) made available to consumers. Prior to this the sole use of NAO was for educational and research purposes. Therefore, the majority of NAO's tutorials and documentation were in the form of university research publications.

6.5 Conclusion

Three demonstrations were developed to showcase different aspects of the NAO robot in order to create interest in STEM education. The demonstrations were developed with the Choregraphe environment and to code in Python due to the simplicity of integrating the two. The code is written for the NAOqi operating system and utilizes the NAOqi API. Despite not completing the Simon Says deliverable due to time constraints, the other demonstrations combine to show off what NAO can do and what students may be able to do if they choose a STEM education. The demonstrations are extendable and will allow future developers to explore the capabilities of NAO more in depth. Working with NAO is rewarding and provides an excellent understanding of both the capabilities and limitations in humanoid robotics.