Android Control Application for Nao Humanoid Robot

Roslyn L. Brown, Heather L. Helton, Allison C. Williams, Michael T. Shrove, Mladen Milošević, Emil Jovanov
David Coe, and Jeff Kulick
University of Alabama in Huntsville,
301 Sparkman Drive, Huntsville, AL 35899

Abstract— *In this paper we present the Nao Controller, a first of its kind application to control the Aldebaran Robotics Nao Humanoid Robot. Nao Controller is the only Android application that has been successful in controlling the commands sent to a Nao Robot without using any sort of middleware for communication. Using packet capturing and executing replay attacks on the Linux machine inside the Nao robot allowed us to build unique movements into our Android application that allows a user to easily control the Nao robot by simply using Wi-Fi and an Android phone. The Nao Controller features many controlling aspects of the robot including Battery Status, Battery Safe Mode, Video Feedback from Nao's camera, Tilt Screen Mode, Manual Movement Mode, Saved Sequences, and a demonstration mode.*

Keywords: Nao Robot Controller, Android Smartphone, humanoid robot, Replay Attack, Nao Battery Monitor

1. Introduction

Smartphone applications are becoming increasingly popular means of use for general application purposes. The University of Alabama in Huntsville (UAH) purchased a popular humanoid robot known as Nao. To conduct research and demonstrations, Nao had great potential in making an impact on anyone who came to demonstrations. Throughout the research of Nao, one thing had been noted. The Nao robot could only be controlled with a computer. Any movement had to be programmed using Choregraphe or the NAOqi file [1]. This stemmed the idea of creating an Android application that would eliminate the use of a computer and make the robot more mobile and agile with creating and controlling movements performed by Nao. Once Nao became mobile, the possibilities of using him for demonstration and recruitment purposes seemed endless.

For this particular paper, we are going to outline the entire process that we used in the creation of our Android application, Nao Controller. First of all, we will explain features of the Nao robot and all of its capabilities. From there, we present other Android projects that are already in existence and being used to interface with the Nao robot.

Then, we describe the Nao development software, including Naosim and Choregraphe. We describe our approach to accomplish our Nao controller task. We had to discover how to use the NAOqi to coordinate with the robot and create our own movement packets sent to Nao. Next, we describe our application and capabilities that have been integrated into our final product. Finally, we discuss how the application has been used in accordance with our university, the University of Alabama in Huntsville.

2. Nao Robot

Aldebaran Robotics, a French-based company, came on the scene in 2004 with a new project known as Nao [1]. This humanoid robot developed into a product that rivaled upon the most sophisticated robots including Aibo, the robot dog developed by Sony [2]. In August of 2007, Nao won the Robot Soccer World Cup Standard Platform League beating Aibo [3]. Nao includes a Linux powered system that makes him very powerful. Along with his internal system, Nao comes with a suite of software used to control it on a user's PC including Choregraphe, which is a graphical programming tool [1]. Also, NAO came with a NAOqi file, which is used as the framework or API for the robot and can be used to program movements of the robot. Mainly used for research and educational purposes, universities have begun developing different capabilities of the Nao robot, such as to promote social engagement interaction with children who have Autism [4].

The twenty-three inch Nao robot is the first generation robot, Nao V3.2, featuring 25 degrees of freedom (DoF), and includes the following sensors: Ultra Sonic, Tactile\Touch, Microphones, Camera, FSR - Force Sensing Resistors, Gyro, and Accelerometer. For this version, the user can connect to Nao via Ethernet or Wi-Fi [5]. However, in the newest version, Nao V4 (NextGen), the user can even connect to NAO via Bluetooth, USB, or Infrared, in addition to the previously mentioned methods. All versions of the robot claim a 45 minutes battery life span when active, which we found was actually the case. [5]

3. Other Smartphone Control Applications

With Nao being so expensive and used mostly for research and academia, there are not very many Smartphone applications on the market that will control the robot. The two applications below were the only applications found that are used to control the Nao robot. These applications can be found on Robot App Store[5]. Even in these applications, it seems that each project only focuses on one or two highlighted features of Nao.

NAO Control [6] is an iPhone application that controls the NAO robot's movements using an on screen joystick. If the joystick is pushed up, Nao will move forward. The same act follows for backward. If the joy stick is pushed left or right,

the Nao robot will shuffle left or shuffle right accordingly. In addition to movements, the application controls Nao robot's speech. The user can select from a list of predefined speech phrases or type text to send to Nao to speak. This application's user interface is very user friendly and the connection of the robot is automated meaning the user does not have to input anything, and the robot tells the user once he is connected to the phone.

NAO Server- Remote Control From Android is an Android Application that controls the Nao robot's movements using the tilt of the phone (accelerometer and magnetic field sensors) [5]. The biggest disadvantage of this application is that it has to use a "middle man" server to communicate between the Nao robot and the Android phone. The server is used because this particular application needs access to the NAOqi file to operate. The NAOqi is not able to be loaded to the phone, which means that it has to be accessed through the server. The NAOqi file is an APK that allows a developer to easily interface with Nao's movements [1]. To connect to Nao with this application, Nao states his IP address by pressing his center chest button. The user enters the IP address into the server application. Then, the user connects the phone to the server. From there, the user can choose to stand or sit the robot. If the robot is standing and the dead switch is pressed, the operator can move the robot forward, backward, turn left, or turn right according to the tilt of the screen.

4. Development Environment with Nao

Nao robot supports many applications including voice recognition, balance, video feed, face recognition, bumper sensors, and text to Speech [1]. Because of the vast features that are included in Nao, we needed to tailor our project to address just a few specific needs for our purpose as oppose to addressing all of the features Nao has to offer. As mentioned above in Section 3, most if not all other Smartphone applications only focused on one to two features of the robot. We have expanded this in our project to focus on many more aspects including the following: battery status, battery safe mode, manual mode, demonstration mode, tilt control with video feed, and saved sequences. After discovering the two projects described above in section 3, we determined that we wanted an application that actually receives information from the robot and combines needed functionality for quickly assembled demonstrations of Nao's capabilities. Before we could begin implementation of the Android application, we had to first get familiar with Nao and the software packages that came with it that were meant to control it.

4.1. Choregraphe

Choregraphe is the programming software that will allow Nao users to edit and create movements in a simple user interface (UI) [1]. The user can create a series of behaviors by dragging and dropping the predefined behaviors from the library, NAOqi. These behavior boxes are easily configurable allowing a user to develop a new movement not currently held in the library. In the application, the user can view the robot's position as they are giving him each movement. They can choose to connect to a Nao robot or to connect to a Nao simulation robot (more to be explained in section 4.2 below). If the user is using a Nao robot, the video feed of the robot can also be seen in the application. This application was used in our development to understand the behavior and interfaces of the Nao robot as well as learning to see how the movements were sent to the robot. Figure 1 is an example screenshot from the Choregraphe application.

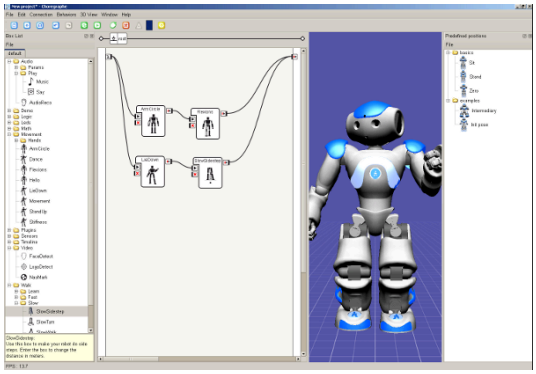

Figure 1: Choregraphe Nao Robot Software

4.2. NaoSim

NaoSim [1] is a simulation tool that was also part of the software package Aldebaran developed to support the Nao robot. NaoSim allows the user to launch a virtual world for the simulated Nao robot to navigate around as well as a virtual version of the Nao robot.

The robot in this environment mirrors the actual Nao robot, which means that if a new move is developed, the simulation Nao can perform that movement, and the user will see the reaction to that movement from the simulation. This protects the actual very expensive Nao robot from damage if the movement programmed does not go as planned. By watching the movement of the simulation, the user can determine if this was the intended movement and if the Nao robot will be able to execute that movement. Choregraphe and Naosim also connect with each other. This is how Choregraphe can use a simulation robot. Figure 2 shows an example of such a connection between Choregraphe and NAOsim.

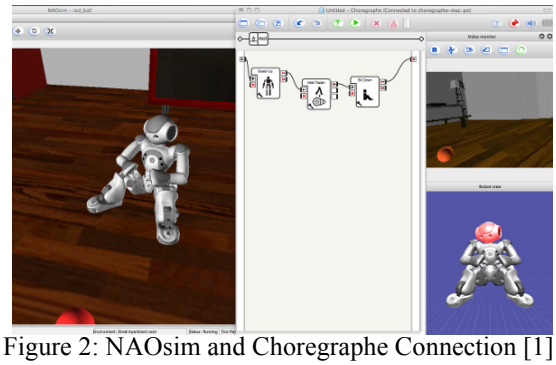

Figure 2: NAOsim and Choregraphe Connection [1]

4.3. NAOqi

NAOqi is the library file that is the API to the whole Nao robot. It runs on the robot's Linux machine and controls the commands of Nao. Every command the robot can execute will be in this library unless the developer creates an entirely new command sequence. For our project, we are currently

using the library version 1.10.10. Although an older version, our robot is the first generation robot, and we felt the risk of upgrading him to possibly receive more functionality from a newer version of the NAOqi was too high. The developer needs to know the "language" of the robot, and this is the file that will allow the developer to communicate. The NAOqi Framework is the programming framework for the developer. The framework is cross-platform and cross-language [1]. In order for the user to run a simulation robot, this file must be installed on the user's computer. This file, as described in Section 3, was a large part in determining our design of NAO controller.

4.4. Nao Web Client

In addition to all of the software packages that have been described, Nao also has a web client as shown in Figure 3. This web client shows the user virtually anything they might need to know about the Nao robot. This is where the robot's version number, Ethernet and Wi-Fi address, joint positions, networks, and many other features can be discovered [1]. This web client became a necessity when we began our implementation of the application as will be described later on in Section 6.

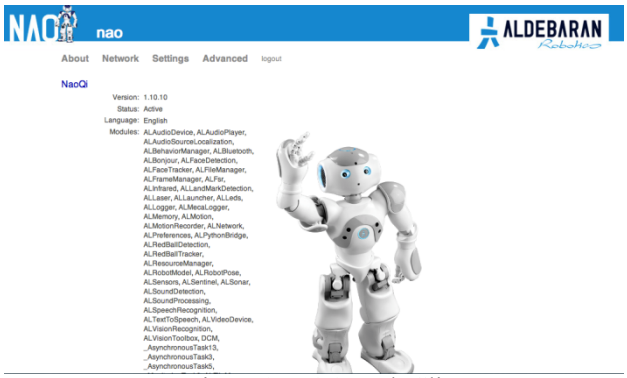

Figure 3: NAO Web Client

5. Our Approach

After dealing with the user interfaces provided by Aldebaran and discovering that the possibility of having the NAOqi file on the phone to access was infeasible, we were faced with having to figure out how to send commands to the robot with just the Android phone, which had never been done before. Finally, it was determined that the best way to tackle this issue was to see what was actually being sent across from the computer to the Nao robot. We set up our environment with Choregraphe and a packet analyzer known as Wireshark [7]. This environment is described below.

5.1. Using NAOqi

NAOqi had a library with it known as the jNAOqi.jar file that was used for the API. NAOqi is available in eight languages including java, and this was the Java library file [1]. Unfortunately, the NAOqi library for java was not compiled to run on a specialized virtual machine designed specifically for android devices, Dalvik [8], so we could not use the API to communicate to the robot through the phone. Since they do not provide the API specifically for the android platform, we created our own Java project and used the NAOqi library file to send command packets over TCP/IP to the robot with a PC. Then we sent the coded commands to the robot while monitoring the TCP/IP connection using Wireshark. Basically, we had to create a separate Java project that was specifically used for generating movements with the NAOqi file. The packets that are being sent are transferred over a TCP/IP connection using the Simple Object Access Protocol [9]. The Wireshark program was used to watch, capture, and strip the raw bytes we needed to resend in our Android application over in order to make the robot perform certain tasks. The following list contains the basic steps we used to create each unique movement sequence:

- Code what movement that needs to be created using NAOqi
- Run code while running Wireshark and observe the bytes sent
- Capture the bytes.
- If the packet size is too large to put directly into the android java files, serialize the bytes.
- Put the generated serialized file into the assets folder of the android program
- Reference the serialized file to pull in the bytes and send them over to the robot.

This process was repeated for each movement programmed into the Android phone.

5.2. Creating Unique Movements

It very quickly became apparent that we would need to create our own unique movements as well as the predefined movements already in NAOqi. Therefore, we used the NAO web client quite frequently to generate these movements. The web client has a Hardware section under the Advanced drop down menu. Here, the web client displays all 25 degrees of freedom and the position they are in at any given time. Consequently, we could hold Nao in the position we wanted to capture, take the values from the web client, and use them in our NAOqi Java application. Figure 4 is a code snippet of two joints, the HeadPitch and HeadYaw of Nao (as illustrated in Figure 5). As shown in the code of Figure 4, the keys represent the floating-point position of movement for that particular joint. Also, the times represent the time at which the Nao robot would be in that position. For this example, HeadYaw and HeadPitch were in key one at 1.5 seconds, key two at 3 seconds, and key 3 at 4 seconds. As shown, this could be quite labor intensive, especially when there are 25 degrees of freedom on the Nao robot. That means that we have to repeat these steps for each of the 25 degrees of freedom of the Nao robot. This contained a lot of trial and error for each unique movement, but our goal for the project was accomplished: create unique movements and send them from the Android phone.

```
names.push_back(new Variant("HeadYaw"));
            times.push_back(new Variant (new float[]{ 1.5f, 3f, 4f }));
            keys.push_back(new Variant( new float[]{ 0.06745f, 0.007628f,
0.06745f }));

            names.push_back(new Variant("HeadPitch"));
            times.push_back(new Variant (new float[]{ 1.5f, 3f, 4f }));
            keys.push_back(new Variant( new float[]{ -0.04146f, 0.42641f,
-0.04146f }));
```

Figure 4: Code Snippet of Joint Movements, Headpitch and HeadYaw

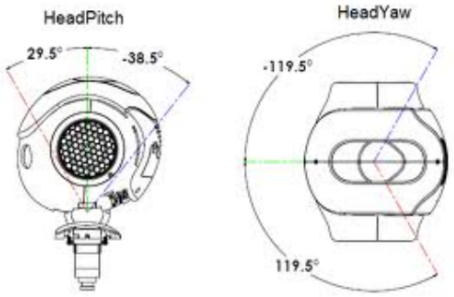

Figure 5: Example of Joint Movement [1]

5.3. Capturing and Sending Movement Packets

The most influential aspect of our application was the capturing and sending of movement packets. We could generate the movements with the NAOqi file all day, but if we did not know what was being sent or how to capture it, then our project would have failed. To capture the packets, we listened on Wireshark [7] for IP addresses communicating with each other (one from the computer and the other the Nao Robot). Each time we sent Nao a movement, we saw the packet go across Wireshark from our computer to Nao's IP address. Therefore, for any movement we programmed, we ran the movement and captured the packet. The captured packets would then be put into our Android application. We created a TCP socket to communicate with the Nao robot and send the captured packet over to Nao via TCP/IP protocol.

The captured packet was then saved. We used a replay attack [10] method where we resent the exact captured packets in the form of raw bytes through our PC using a TCP socket connection to the robot to make sure that he performed the same action.

Once we verified that the replay attack method would work, we then proceeded to add the packet bytes (either in raw form or serialized form if too large) used in our method to our Android application.

We created several different classes pertaining to the different parts of the body (NAOhead, NAOhand, etc). Then, we created several methods within each class that would be responsible for sending each movement. For example, the NAOhead class had a different function for moving the head up, down, left, right, and centering horizontally and vertically.

Within each function we stored the packet bytes into byte arrays. The arrays were either embedded into a function that was used to send the packet if small enough, or if the arrays were too big due to java method size limits of 65535 bytes [11], the arrays were serialized and read in.

Then, we proceeded to send the stored packets over the network. For each packet we send, we had to wait for an acknowledgement from the robot before we proceeded to make sure that the command was actually sent and received correctly. First, we write to the server. Then, we wait for an acknowledgement to come back and read the first sentence of that acknowledgement. For some movements we are reading the entire message, but in this case we are just reading the first line of the response to make sure the robot received our message.

6. Nao Controller Application

As described previously, our Nao controller application contains many features that have never been developed for the Nao robot before. One of the biggest perks to the Nao Controller application is the ability to communicate with Nao through an Android phone without a server/computer setup to be the "middle man". No middleman is needed for our application. Before this application, the only android application to control Nao had to use the server setup on user's computer to communicate the movements to the robot. This setup almost made the android application completely obsolete since one of the main advantages to having an application on your phone is the ease of use and the portability of the application. The reason why the other Android application was in need of the server is due to the NAOqi file. The NAOqi as described above contains all the possible commands that can be send to the robot. It is the API that is responsible for every command pertaining to the robot. Early on, it was discovered that it was not possible to load the NAOqi file onto the Android phone and have our program access it which lead us to the approach we took in Section 5. Using this approach revolutionized our application and made our app the very first of its kind: the first Android application to solely control and interface with the Nao robot thus keeping the application portable to help with demonstration ventures for which the application was originally intended. In addition to only using an Android phone to control the robot, the application went a step further and decided to tackle many different features of the Nao robot, something that the preceding applications did not do. The following features of the application were developed keeping in mind the limitations of applications previously developed for Nao.

6.1. Connection Screen

The connection screen is used to enter the IP address of the Nao robot. The IP address is obtained by pressing the center chest button of the Nao robot. Nao will then say the IP address to which it is connected. If for some reason Nao is not connected to the Wi-Fi, the user can go to Nao's web client while it is plugged into the router with an Ethernet cord. From there, the user can select the available network for Nao and assign Nao a wireless IP address. See Figure 6 as an example.

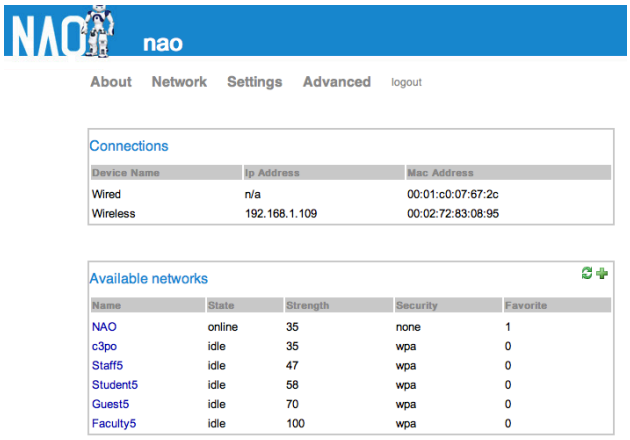

Figure 6: Nao Web Client Available Networks Tab

The EditText itself that is found on the Connection Screen has an IP Address Regex [12], which is a Pattern Detection algorithm, behind the scenes. This is to ensure that whatever the user types into the EditText is actually an IP address before the program tries to make a connection to the robot. Also, if it is in fact an IP address, it will then need to be determined if it is Nao's IP address.

To accomplish this task, we send a ping out to the robot using the IP address entered by the user. If we receive a battery status back, we know that the IP address is in fact that of Nao, and the user of the application will proceed to the Main Menu screen shown in Figure 7 which contains all the functionality of the application. If we do not receive anything back, we have a time out, and the user receives a Connection Unsuccessful pop-up box.

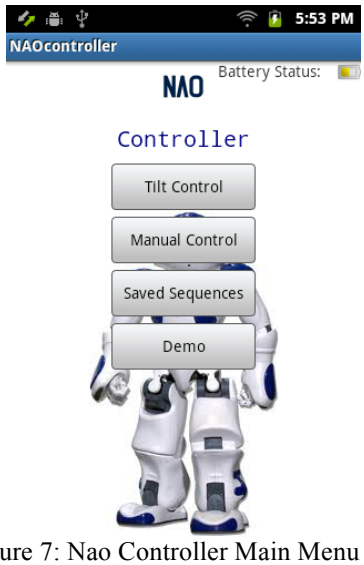

Figure 7: Nao Controller Main Menu

6.2. Tilt Screen

The Tilt Screen functionality was very similar to the Android project described above in section 3. The user can control the robot using the Gyroscope and Accelerometer sensors on the Android phone. The user must have one of the two dead switch button held down for the Nao robot to register the movement. As the user uses tilt to execute movements, the movements are displayed to the user via a Marquee bar. The battery status of the robot can also be seen in this screen, so the user of the application will always have a battery status of the robot while in Tilt operation. The battery status is updated every ten seconds. Therefore, the control aspect of the Tilt is much like that of the previous Android project with the addition of the Battery Status. However, our application went a step further by pulling the video feed from the NAO robot and displaying it to the user on the Tilt screen. Now, while the user is controlling the robot, they can also get up to date video feed from the Nao robot's camera.

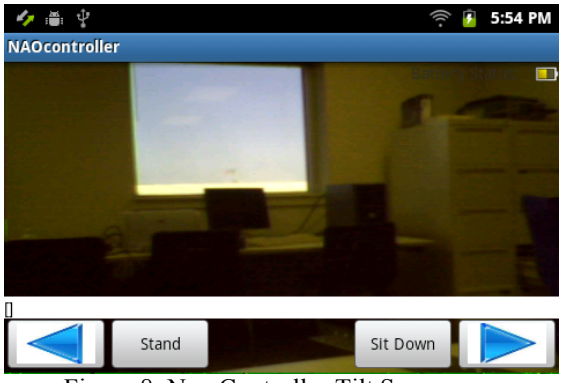

Figure 8: Nao Controller Tilt Screen

6.3. Manual Movement Screen

The manual movement screen was originally created just to test the functionality of each movement of the robot. However, it quickly became apparent that this Manual Movement screen would be useful if left in the application as well. In this screen, each of the Android's possible movements is displayed to the user in an expandable dropdown list. The top-level functions include Head, Arms, Legs, and Misc. The head function includes the following lower level functions: head tilt up, head tilt down, head turn left, head turn right, turn center vertical, and turn center horizontal. Arms has the following functions: close left hand, open left hand, close right hand, open right hand, right arm up, right arm down, right arm right (arm extended away from body), right arm left (into the body), right arm front, left arm up, left arm down, left arm right (into the body), left arm left (arm extended away from body), and left arm front. Legs have the following functions: walk one step forward, walk one step backward, walk backward eight steps, walk forward eight steps, crouch, turn left, turn right, sit, stand from sit, and stand from crouch. Finally, miscellaneous has the function to relax or stiffen the robot (cut on and off the servos of the NAO robot) and dance, which is the Gangnam style dance with the Nao robot playing the song through his speakers. Once the user selects a movement, the robot will perform the movement in near real time.

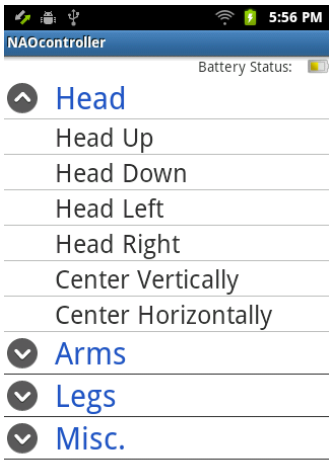

Figure 9: Nao Controller Manual Movement Screen

6.4. Saved Sequence Screen

In the Saved Sequence screen, the user has the option of creating their very own unique sequence using all of the possible manual movement controls described above. The user can choose to add a sequence, select all the movements they would like, and then save the sequence. From there, the sequence will show up in the list of Saved Sequences on the application. The user can choose to run the sequence, in which case the Nao Robot would perform each movement in series that the user has selected in that particular sequence. The user also has the option to delete or edit a sequence. This feature was added because of the feasibility of being able to quickly piece together sequences when needed. This would become great importance as a demonstration aspect. The user can queue up a list of sequences pertaining to that particular setting and execute those movements without ever having to use a laptop. There are some safety precautions put into place so that whatever sequence is entered, it will not allow a dangerous sequence to be created that could harm the robot. For example, the application will not allow the user to make the robot walk forward if the robot is sitting down.

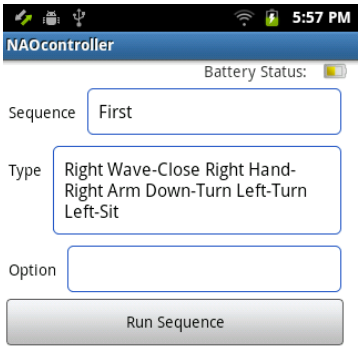

Figure 10: Nao Controller View Saved Sequence Screen

6.5. Battery Status and Battery Safe Mode

As described earlier, the battery status of the robot is available on every screen where the user can control the robot's movements. The battery status symbol changes from green to yellow to orange to red depending on the battery level of the robot. Once the battery reads 10% to 20%, the user is alerted via a pop-up dialog box that the battery is getting low for the Nao robot. Now, when the battery is at 10% or below, the robot will go into Battery Safe Mode. In Battery Safe Mode, the robot will automatically sit down and will no longer receive any commands from the Android application. This prevents damage to the Nao robot if a sudden decrease in battery life could cause the Nao robot to crash during a movement. From there, the user must go and plug Nao back in to begin the charging process.

6.6. Demo Mode

Demo Mode was created as a "wow" factor to show a range of motions that the robot could perform. We wanted to create a fun sequence of steps that would be entertaining for potential new UAH recruits and visitors of UAH. The most familiar dance at this point would be something known as Gangnam style, a YouTube video gone viral. Therefore, we took the steps we described in section 5 and began mapping out the movements of the dance. The end result is a forty-five second dance of Nao performing Gangnam style while the song plays from his speakers [13].

7. Lessons Learned

The Nao application was created to provide an easier way to set up and control the Nao robot for demonstrative purposes. Our purpose for the application has already been proven to be a success at the Board of Trustees meeting [13] with the President of UAH including members of the University of Alabama, the University of Alabama in Birmingham, and of course, UAH. Nao performed his dance and then took a humble bow to conclude the demonstration. Most of the members of the universities present claimed they would love to have something like this demonstration at their school. In addition to this meeting, Nao made another appearance with the President at the UAH Foundation Committee meeting. With all of the praise and hype of using the Android application with Nao, we even have recruitment opportunities to take Nao to local high schools to promote Engineering and recruitment possibility at the University of Alabama in Huntsville.

8. Future Work

One aspect that could be improved upon is automatic connection to the Nao robot. Instead of the user having to press the center chest button of the robot and manually entering into a connection screen, the phone could get this information from the Nao robot itself.

Also, the Nao robot has text to speech functionality. This would be an incredible new feature to add to our application that would help with demonstration impacts.

The robot could always have more movements and demonstration modes added. We limited the Nao robot's actual range of movements due to time constraints and limited his demonstration functionality to just one dance. For the future, adding more interesting demonstrations and movements would be an overall improvement to the project.

In addition to these recommendations, since our project focused a good amount of maintaining the safety of the robot, a new feature could be a fall sequence that is designed to minimize the impact of a potential fall if Nao is in the middle of an action. A fall initialization algorithm could be put into place to accommodate such an accident to prevent further injury to the Nao robot [14].

Finally, using an updated version of the NAOqi on the robot could help with building newer functionality that is already included in the newer version of the file as well as testing our application with newer versions of Nao.

9. Conclusion

We developed a unique Android project that can be used to control the Nao robot directly, without any special equipment. The user only needs our application on the Android phone, Wi-Fi, and the Nao robot. Our application has accomplished so much more than any of its predecessors. With a lot more functionality, the users can do almost anything to Nao with our application including receiving video feed, battery status, battery safe mode, demonstration mode, and manual movements as well as creating custom movement sequences on the fly. Also, our application is the first of its kind to be able to control Nao's movements on an Android phone without the use of any type of computer or hardware "middle man" equipment. The elimination of such PC-based middleware equipment created a portable demonstration system with our Nao robot and Nao controller. With the increased amount of portability for demonstrations, we have discovered that not only is our application and the Nao robot impressive in itself, but the application is being used as a recruitment mechanism and demonstration robot to bring attention to the University of Alabama in Huntsville and the Electrical and Computer Engineering department.

Safety features were set into place with each aspect of our Nao Controller to prevent damage to the robot. With the tilt based movement control, the user receives video streaming from Nao's camera to view what the robot is seeing. The dead switches on tilt control allow the user to control Nao's movements only when these are pressed, creating a safety barrier for the robot. Another safety net is the battery monitoring that commands the Nao robot to sit before the battery drops to an unsafe level.

Our breakthrough with Nao Controller is the new way we capture movements through Wireshark. Using this capability, we are able to perform replay attacks on the Nao robot to repeat the movements sent which allows the ability to easily add more movements and demonstration capabilities to our application.

10. Acknowledgments

We would like to thank Andrew Cecil for his work with setting up the Nao environment and insight into his experience on working with the Nao robot. He created the initial development environment that served as a starting point for our project.

We would also like to thank Mr. Ray Garner, Director of Government Relations and Public Affairs of UAH, and Dr. Robert Altenkirch, President of the University of Alabama in Huntsville. These two individuals made our concept of using Nao robot as recruitment tool a reality. By having Nao perform for the Board of Trustees and the UAH Foundation Meeting, we promoted Engineering at UAH and the capabilities students could develop while they are being taught here at the University of Alabama in Huntsville.

11. References

[1] "Nao" Aldebaran Robotics. Last accessed 22 April 13. http://www.aldebaran-robotics.com/en/

[2] Aibo. Last Accessed 28 April 13. http://www.sonyaibo.net/aibostory.htm

[3] RoboCup. Last Accessed 28 April 13. http://www.robocup.org

[4] Daniel O. David, et al. "Children With Autism Social Engagement In Interaction With Nao, An Imitative Robot." *Interaction Studies* 13.3 (2012): 315-347. *Communication & Mass Media Complete*. Web. 29 Apr. 2013.

[5] Robot App Store. Last accessed 22 April 13. http://www.robotappstore.com

[6] "Nao Store". NAO My Apps- Help- Contact. Last Accessed 22 April 13. https://store.aldebaran-robotics.com/category/applications/

[7] Wireshark. Last accessed 28 April 13. http://www.Wireshark.org

[8] "dalviik.system." Android Developers. Last Accessed 28 April 13. http://developer.android.com/reference/dalvik/system/package-summary.html

[9] Web3C Recommendation. Last Accessed 28 April 13. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[10] Replay Attacks. Last accessed 28 April 13. http://msdn.microsoft.com/en-us/library/aa738652.aspx

[11] Prashant Dava. "Method Size Limit in Java." DZone JavaLobby. 13 June 11. Last Accessed 28 April 13. http://java.dzone.com/articles/method-size-limit-java

[12] Regular-Expressions.Info. Last accessed 28 April 13. http://www.regular-expressions.info/examples.html

[13] Gattis, Paul. "'Gangnam Style': UAH students create app to teach robot the moves (video)". AL.com. 25 April 2013. Last accessed 28 April 2013. http://blog.al.com/breaking/2013/04/gangnam_style_uah_students_cre.html

[14] J. Ruiz-del-Solar, R. Palma-Amestoy, R. Marchant, I. Parra-Tsunekawa, P. Zegers, "Learning to fall: Designing low damage fall sequences for humanoid soccer robots", *Robotics and Autonomous Systems*, Volume 57, Issue 8, 31 July 2009, Pages 796-807, ISSN 0921-8890, 10.1016/j.robot.2009.03.011.