Remote Interaction with a NAO Robot using a Tablet Device

Ms. Jennifer Leaf, Eastern Washington University

Jennifer Leaf is a student in the Mechanical Engineering department at Eastern Washington University. She previously received a Bachelor of Science in Computer Science from Pacific Lutheran University and a Master of Science in Computing and Software Systems from the University of Washington, and worked as a software engineer and program manager in private industry. She intends to pursue graduate studies in robotics.

Dr. Robert E. Gerlick, Eastern Washington University

Dr. Gerlick is Assistant Professor of Mechanical Engineering and Mechanical Engineering Technology at Eastern Washington University. He teaches courses in the areas of Robotics, Mechanics, Thermodynamics, Fluids, CAD, and Capstone Design.

Dr. Donald C. Richter, Eastern Washington University

DONALD C. RICHTER obtained his B. Sc. in Aeronautical and Astronautical Engineering from The Ohio State University, M.S. and Ph.D. in Engineering from the University of Arkansas. He holds a Professional Engineer certification and worked as an Engineer and Engineering Manger in industry for 20 years before teaching. His interests include project management, robotics /automation, Student Learning and Air Pollution Dispersion Modeling.

Remote Interaction with a NAO Robot Using a Tablet Device

Abstract

The NAO humanoid robot includes several programming tools and development kits that are supported on commonly available operating systems for defining the behavior of the robot at runtime. However, there are situations where it is desirable to control the NAO, or to run specific programs, from tablets or other devices that are not supported by the vendor-provided tools. To support a research project for improving the usability of the NAO by therapists treating autism spectrum disorder (ASD) in children, we developed a method of using the WebSocket protocol to send commands from an app running on a tablet device to a NAO robot. A proof-of-concept architecture and implementation using an Android tablet app is presented. Finally, the alternative technologies and potential next steps for future enhancement are discussed.

Introduction

The NAO robot [1], created by SoftBank Robotics (formerly Aldebaran Robotics), is a humanoid robot with a rich set of features, including a vision system, text-to-speech system, speech and facial recognition, touch sensors on its head, hands, and feet, and 25 degrees of freedom to move its head, arms, and legs. It can be programmed using a drag-and-drop GUI software package called Choregraphe [2], or via the Python or C++ programming languages using software development kits (SDKs) provided by SoftBank. The SDKs permit a software developer to create programs to remotely control the NAO, in addition to writing programs that run directly on the NAO.

Motivation

To support a research project using the NAO in autistic therapy settings, a need to permit a non-programmer practitioner to control the NAO was identified. It is assumed that a knowledgeable programmer would create programs to run on the NAO to support various activities, such as interacting with a patient by reciting the text of a story and asking questions about the story. However the therapist should be able to command the NAO, in real time, to provide a desired response in order to support the unique needs of the patient currently interacting with the NAO. To minimize the technology footprint, it is desirable to provide the practitioner with an Apple iPad or similar touchscreen device containing an app that would permit them to run the pre-loaded programs on the NAO using a user-friendly interface.

While the SDKs provide a complete interface to the NAO's capabilities, the remote programs can only be run on supported operating systems, which include Windows, Mac OS X, and Ubuntu Linux. Noticeably absent from the officially supported list of platforms are the predominant mobile operating systems iOS (which is the operating system for the iPad) and Android. Programming the NAO using only tools and SDKs provided by the manufacturer minimizes the chances of significant rework when new versions of the NAO software are

released, thus reverse engineering the network communications as an opaque packets of bytes is ruled out as a solution. Defining the communications between the NAO and the tablet as part of the application design will permit the most flexibility when adding new features for end users.

Related Work

Brown et al. [3] reverse engineered the network communication between the NAO and a remote computer by writing a Java application to run on a PC, and then capturing the packets exchanged between the PC and NAO for various commands. The Android app was written to send the appropriate bytes to the NAO over a TCP/IP connection to mimic what a PC-based application would send. This approach can be particularly fragile from an implementation perspective, since the underlying binary format is not formally documented by the robot manufacturer and subject to change without notice.

At least three projects have documented a similar approach of creating a custom network protocol as the one described in this paper. Ahn et al. [4] created an Android app to remotely control the NAO. It used an intermediary bridge server (PC) to translate the commands sent by the app to the appropriate application programming interface (API) calls for the NAO. They streamed video from the NAO directly to the Android device in order to maintain a usable frame rate for the video stream.

Another iOS app[1] uses a datagram socket between the NAO and an iPhone/iPad device to transmit binary commands, which are interpreted by a custom behavior installed on the NAO and translated into the appropriate NAO API calls[2]. While the source code for the components to install on the NAO is published to GitHub, it did not include a license file, so it was unclear whether the code is open source, and therefore this source code was not examined nor used. A third iOS app[3] used a similar approach to the one described here, but using a unicast (TCP) socket and text based commands. The source code[4] is published as open source, but restricted from commercial use.

Several projects[5,6,7,8,9] have been published on YouTube or to a mobile device app store, but were not published as open source, nor did they reference any documentation or publication describing

---

[1] https://itunes.apple.com/us/app/nao-controller/id991306890?mt=8
[2] https://github.com/Bigshan/NAOControllerClient/blob/master/NAOqi2.1/NAOControllerClient/NAOControllerClient/behavior.xar
[3] https://play.google.com/store/apps/details?id=de.robotik.nao.communicator&hl=en
[4] http://northernstars-wiki.wikidot.com/projects:naocom
[5] https://www.youtube.com/watch?v=cDu2iLKLQdk
[6] https://play.google.com/store/apps/details?id=de.daboapps.androidnao&hl=en
[7] https://itunes.apple.com/us/app/nao-control-for-aldebarans/id534912647?mt=8
[8] https://play.google.com/store/apps/details?id=com.robinbonnes.naorobotcontroller&hl=en, also https://www.youtube.com/watch?v=NsO-a8u35cc and http://www.robotappstore.com/Apps/NAO-Robot-Controller-for-Android.html?x=3D508580-906C-40EF-8484-DB9E7C5AFEAA

their design. As their approach is not described, it cannot be used or adapted, but they are mentioned here for completeness.

Finally, the app shown in this YouTube video[10] was the model used for the approach described in this paper. This project used the Websocket protocol [5] to communicate between a NAO and an iPad. A Python Websocket server ran on the NAO, using the pywebsocket library[11] originally written by Google. This library was less suitable for an application targeted at non-technical users since it contains many scripts, which increases the complexity of deploying the software to the NAO. The software running on the iPad[12] is written in JavaScript.

System Design

In order to support development of a tablet-driven interface to the NAO, a number of existing technologies were assembled to bridge the gap between the tablet and the NAO software. Since access to an iPad device and a software developer with iOS programming experience were not readily available, the implementation of the tablet interface targeted an Android tablet. The NAO supports running programs directly on the NAO using C++ and Python. Python was selected for this project due to its shorter learning curve.

A set of Python scripts was developed that use NAO's text-to-speech API to read books and ask related questions. The books are pre-loaded onto the NAO as plain text files, with specific text markers to indicate the book title and questions as distinct from the text of the book. A user runs an app on an Android tablet to select the book to read, which section from the book to read, and which question to ask. The app includes buttons to instruct the robot to indicate a correct or incorrect answer. The app does not store the answers to the book questions. While the child is expected to answer the questions verbally, they may not be able to articulate the appropriate words clearly enough for the speech recognition software to decode them. Allowing the human user to determine if the answer is correct supports the customization of the therapy activity to the capabilities of the individual patient.

The system consists of components running on both a tablet device and directly on the NAO robot. Figure 1 shows the components created or used in the system. Boxes with a dashed border indicate components or libraries that were reused from other sources. Boxes with a solid border indicate components that were written specifically for this project.

---

[9] https://www.youtube.com/watch?v=r6zGxhUSPA0, also http://www.robotappstore.com/Apps/NAO-Server--Remote-Control-From-Android.html?x=41B586C9-FEC2-427C-B75F-1B7DAB589CD7

[10] https://www.youtube.com/watch?v=PYhVelznnFk

[11] https://github.com/google/pywebsocket

[12] https://github.com/jbyu/naoRemote.git

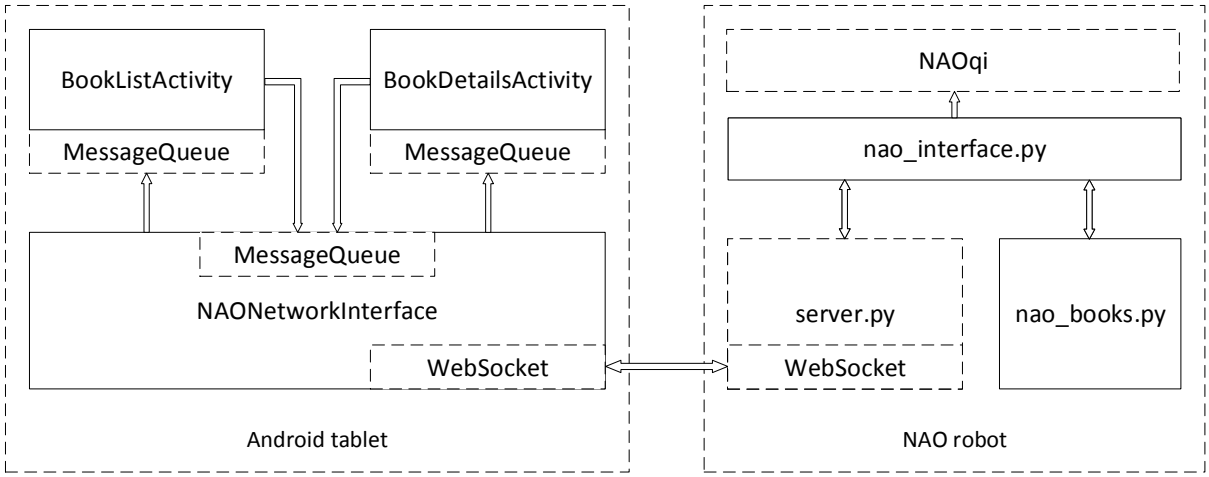

Figure 1. Software components for the NAO book reading system.

Since applications written for the tablet cannot use the NAO SDK directly, it is necessary to have a component running on the NAO that can interact directly with the NAO software. A Python script called nao_interface.py was written to accomplish this. Programs created in Choregraphe contain one or more Behaviors, which are a set of instructions that the robot should carry out. The ALBehaviorManager API allows a programmer to start or stop a Behavior by knowing the identifier of the Behavior. The nao_interface.py script is intended to be customized as new Behaviors are installed on the NAO. It can also be customized to use other NAO APIs to control NAO's operation directly, without writing a separate program in Choregraphe.

In order for a remote application to communicate with the nao_interface.py script, a technology called the WebSocket Protocol [5] is used. This protocol is an openly documented network communications format to create a low overhead, bidirectional communication channel that uses the widely implemented HyperText Transport Protocol (HTTP) [6] used by web servers worldwide. WebSockets was selected for this project since libraries that implement this protocol exist for many programming languages and operating systems, including iOS and Android. This allows development of NAO programs and the tablet interface to occur independently, and permits replacing the Android tablet with a different system in the future if desired.

Another Python script, websocket_server.py, creates and manages the server end of the communications interface between the NAO and the tablet app. This script is a third party open source program [7] that implements the basic functionality of WebSockets with no additional dependencies, keeping the software footprint running directly on the NAO as small as possible. This script is used as-is with no modifications for this project.

A third Python script, server.py, runs on the NAO when the NAO is started. It uses websocket_server.py to create the listening socket and wait for messages. When a message is received from the tablet, this script then calls a function in the nao_interface.py script to actually run the appropriate program on the NAO. The server.py script also originated from the

websocket_server project, and is used largely as-is, except for the interaction with nao_interface.py.

Table 1 describes the commands implemented in nao_interface.py for the book program. Arguments are separated by a single space. Response messages from NAO are formatted using JavaScript Object Notation (JSON) [8], which is a commonly used standard for exchanging data between two applications. The app uses the Autobahn open source library [9] to translate the JSON text into an in-memory tree representation so its contents can be inspected more readily. No error messages are currently returned if the client requests a book, section number, or question number that does not exist.

Table 1. NAO book reading commands.

Command name	Arguments	Return Value	Description
listbooks	None	JSON dictionary, key = book ID, value = book title	Gets a listing of all books loaded on the NAO. The book ID is the book filename, which is used in subsequent commands). The title is the text as extracted from the title line in the file.
getbookdetails	<id>	JSON dictionary, key = section number, value = number of questions	Gets the number of sections in the book, and the number of questions for each section.
readsection	<id> <sectionNumber>	None	Reads the n^{th} section number from the book, where n = the section number given in the command. Section numbers are 1-indexed.
readquestion	<id> <sectionNumber> <questionNumber>	None	Reads the m^{th} question from the n^{th} section in the book, where m = the question number given in the command and n = the section number given in the command. Section numbers and question numbers are 1-indexed.
correctanswer	None	None	Uses text-to-speech module to say affirmative response
incorrectanswer	None	None	Uses text-to-speech module to say negative response

A native Android application, written in Java, runs on the tablet. The application provides a simple user interface (UI), allowing the user to specify the IP address and port where the Python WebSocket server is running. Once the app connects to the Python server, it sends space-

delimited string messages to the NAO, which processes the command. In the case of the book program, the commands to get the list of available books, and to get the list of sections and questions for a particular book, return a response to the tablet app. This response is then used to create appropriate buttons on the UI for the user to control what NAO says.

The nao_books.py script encapsulates the details of where the book data is stored and how the book files are formatted. In this way, the format and location of the book data can be changed without impacting the rest of the system.

In order to customize the Android app screens for the specific book being read, the book text requires a small amount of custom formatting. Each book is in its own text file. All book text files are stored in a single folder on the NAO, which is configurable in the nao_books.py script. This is an excerpt from a formatted book file:

```
Title: Mary Had a Little Lamb

Mary had a little lamb.
Its fleece was white as snow.
And everywhere that Mary went
The lamb was sure to go.

? What type of animal did Mary have?
? What color was the lamb's fleece?
```

There are three distinct sections in this excerpt: the title, the main text of the book, and the questions. The title line must be the first non-blank line in the file, and it must start with “Title:”. The remainder of this line is considered to be the title of the book, and will be presented verbatim on the tablet screen to the user.

The lines with no special prefix are considered to be the text of the book. Books can contain one or more sections, where sections are divided by blank lines (i.e., two or more consecutive newlines). Sections can contain as few as one line, or as many as desired. Since lines are read by NAO one at a time, separating each sentence onto its own line may be desirable to more closely mimic human speech. Once a blank line is encountered, the section ends.

Question lines begin with a question mark, followed by the text of the question, which will be read verbatim by NAO. More than one question can follow a particular book section, provided each question is on its own line and the line begins with a question mark. Excess whitespace and blank lines between questions or consecutive sections are ignored when parsing the file.

If multiple buttons are pressed on the tablet app, the commands are processed sequentially by the Python script running on the NAO. The robot will finish saying the text for the current command before processing the next one.

There are two screens in the app. The main screen as shown in Figure 2 has a fixed button to request the list of books available. Once the list of books is received, a button is added for each book. Since the available books will vary based on the text files currently loaded on the NAO, the buttons on the UI are added at runtime. User interface controls have a field called Tag that allows storage of an arbitrary Java object to associate with the control. For the book list, the Tag is the book ID, which is the filename of the book. This ID is used in subsequent commands to specify the book that the user desires to read.

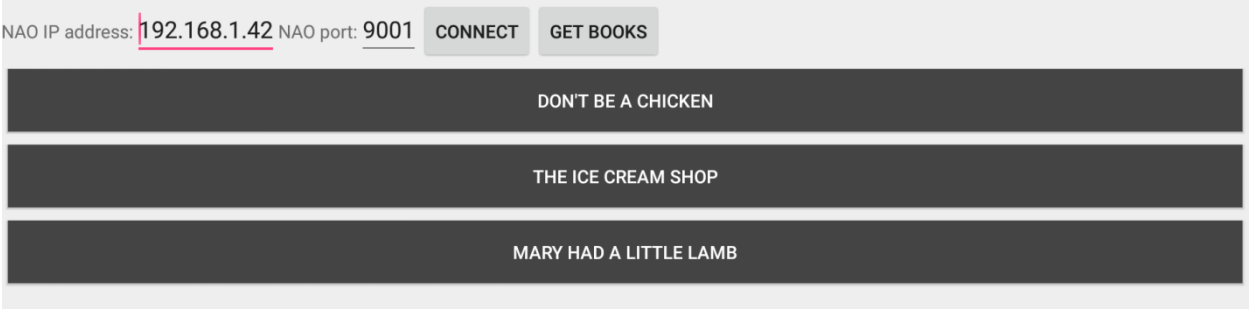

Figure 2. Book selection screen.

When the user presses the button for a specific book, a new screen is created for the buttons specific to the selected book. A representative example is shown in Figure 3. The left side of the screen contains commands that are available regardless of the book selected. The remainder of the screen is populated with buttons to allow the user to select which book section or question to read next.

Figure 3. Screen when a particular book is selected.

Conclusion

While the current programs are suitable for proof-of-concept use, there are a few modifications that could be made to improve the usability for non-technical end users.

Once the user has pressed a button for NAO to read or say something, NAO will read the section or question until completion. There is no means for the user to interrupt NAO, which may be problematic if NAO is in the midst of reading a lengthy passage. User studies should be done to determine if certain commands should interrupt NAO and immediately begin reading something else.

NAO will only say words that are pre-programmed into the book files or into the Python scripts. A user may want to have NAO say something ad hoc, so the ability to type some text on the tablet and send it to the NAO could be desirable.

In order to upload new books, the user would have to acquire and use an FTP program, and upload the files to the correct folder on the NAO. This may be intimidating to a non-technical user, so writing a standalone program to hide the details of connecting to the NAO and uploading files may prove helpful in reducing the complexity for end users.

In the next phase of this project, we plan to conduct a usability study with current practitioners to determine whether the tablet interface is useful to a practitioner during a therapy session. Based on this feedback, we will modify the tablet interface, and add more behaviors to the NAO and to the app to broaden its applicability to more patients.

References

[1] SoftBank Robotics, "Discover Nao, the little humanoid robot from Aldebaran | Aldebaran," [Online]. Available: https://www.ald.softbankrobotics.com/en/cool-robots/nao. [Accessed 5 July 2016].

[2] SoftBank Robotics, "Choregraphe User Guide," [Online]. Available: http://doc.aldebaran.com/2-1/software/choregraphe/index.html. [Accessed 5 July 2016].

[3] R. L. Brown, H. L. Helton, A. C. Williams, M. T. Shrove, M. Milošević and E. Jovanov, "Android Control Application for Nao Humanoid Robot," in *Proc. of the International Conference on Frontiers in Education: Computer Science and Computer Engineering*, 2013.

[4] H. Ahn, H. Kim, Y. Oh and S. Oh, "Smartphone-Controlled Telerobotic Systems," in *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2014 IEEE International Conference on*, Hong Kong, 2014.

[5] I. Fette and A. Melnikov, "The WebSocket Protocol," [Online]. Available: https://tools.ietf.org/html/rfc6455. [Accessed 25 July 2016].

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," [Online]. Available: https://tools.ietf.org/html/rfc2616. [Accessed 20 January 2017].

[7] Pithikos (alias), "Websocket Server," [Online]. Available: https://github.com/Pithikos/python-websocket-server. [Accessed 25 July 2016].

[8] Ecma International, "The JSON Data Interchange Format," 2013. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Accessed 12 January 2017].

[9] Autobahn Project, sponsored by Tavendo, "Autobahn|Android," [Online]. Available: http://autobahn.ws/android/index.html. [Accessed 9 August 2016].