Design and Implementation of Portable, Social Robot on Android with Speech Recognition and Text to Speech

Rhindress B.[1] Ge S.S.[2] Tu F.[3]

Swanson School of Engineering, University of Pittsburgh
Robotics Research Lab, National University of Singapore

ABSTRACT

In this paper, we present the implementation details and results for the design of a simplistic portable social robot. We aim to implement a social mobile robot powered with Artificial Intelligence. The platform of choice for this design is an Android-controlled, Arduino-powered robot. The robot sits on two tank-style treads each driven by a DC motor with wheel encoders. By integrating multiple off-the-shelf technologies such as the Android development platform, Text to Speech software, and Voice to Text recognition software, we are able to quickly and easily create a robot model that fulfills the criteria of being able to talk with people in a human fashion, detect human emotion, remember past interactions and allow for navigational control in an unfamiliar environment.

INTRODUCTION

Motivation

Many science fiction plots are predicated on the idea that robots will inevitably gain human characteristics. The modern equivalent of this idea is social robotics, the concept of creating machinery to interact with humans in the human world. After all, people communicate with each other via non-programmatic mediums: touch, sound, sight, etc. At the same time, modern technology is pushing ever towards mobile platforms. Mobile computing and robotic platforms are popular due to their potential versatile applications, scalability and ease of control. If technology is moving in both of these directions, there is some motivation to combine social and mobile robotics.

Previous Work: Hardware & Software

All work presented hereafter builds on the previous hardware and software development of NUS Electrical Engineering Masters student Chang Poo Hee [1]. The physical robot used is a two-motor, treaded wheeled tank robot powered with an Ultralife rechargeable 20V battery. An Arduino Atmega2560 micro controller board is used to communicate commands to the motors for movement. Commands to the Arduino come via a wired Android-Arduino USB serial connection. Note: in order for this connection to work, an Android phone must be used that allows USB hosting. This may not come standard, as most times a USB connection to an Android is to a computer, which is considered the host.

[1] Student
[2] Professor
[3] Mentor

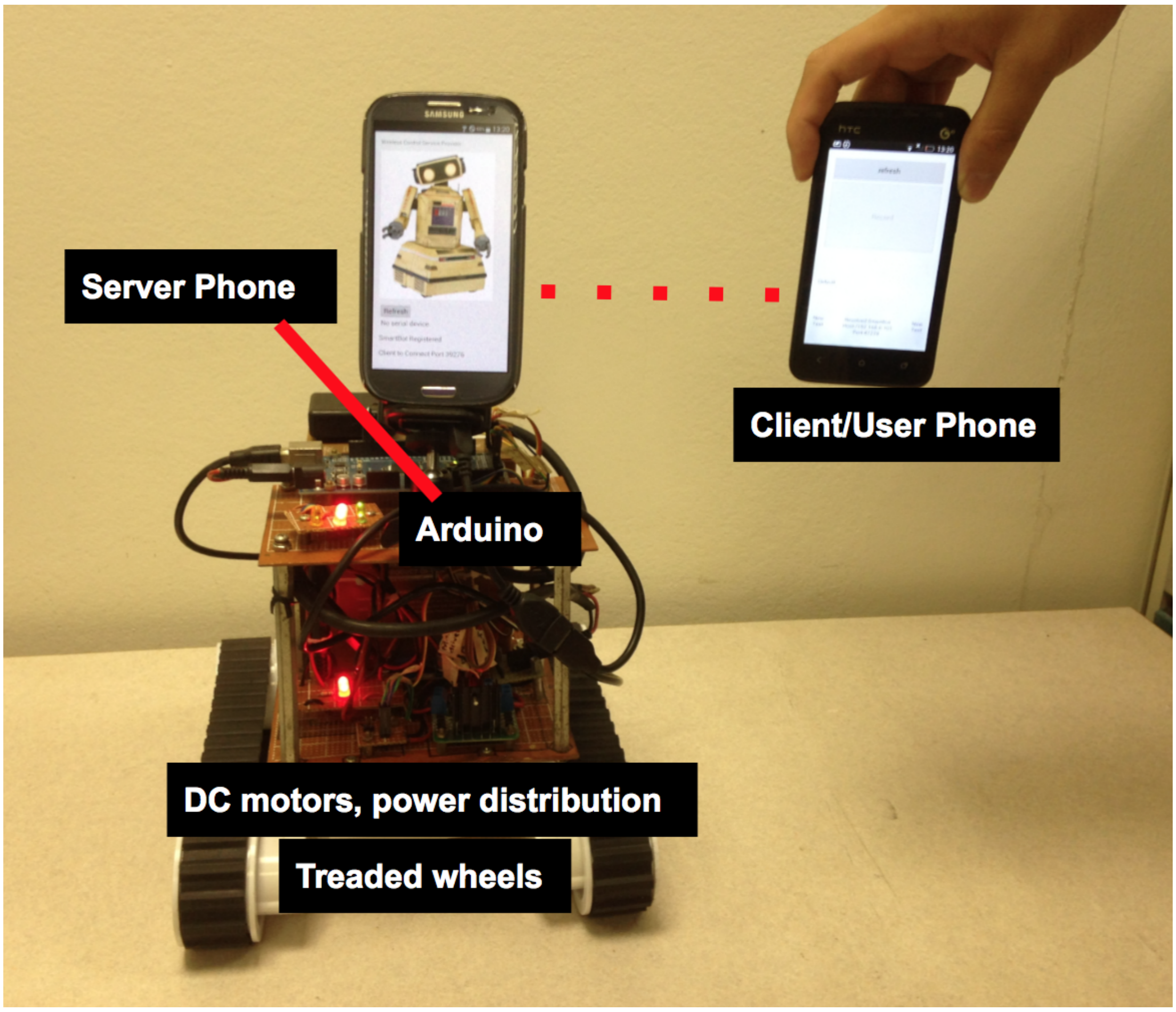


Figure 1: System previously created. Note three layers of abstraction: hardware (power system, DC motors, Arduino control), Arduino-Android interface (between robot and server phone), Android-Android interface (server and client phones).

Previous software design for the robot includes an Arduino program that opens a two-way line of communication. Since the Arduino receives external commands from a wired Android device, the first line of data passing comes from the Android device to the Arduino. A communication protocol was created that requires the Android phone to serially write a C to the Arduino at 4 Hz to ensure the signal flow is working properly. If this protocol is not obeyed, the robot system will not accept motor commands. If this protocol is obeyed, the Android phone may serially write right and left motor speeds to Arduino for commanding the motors. To provide users with real time feedback the Arduino also concurrently serially writes the current speed of each motor back to the Android device. These data are collected by way of wheel encoders.

On the user side, two Android devices are paired in a client-server architecture and connected on a local wireless network using Androids Network Service Discovery package, a zero-configuration network API. The phone wired via USB to the Arduino acts as a listening server for the C communication protocol message, and any motor commands. Once this server is initialized, the client attempts to bind. In the client-server architecture, communication and command messages must originate at the user-held client phone, which acts as the human machine interface.

Android applications were previously programmed for several user modes: a four-direction button mode, two-slider control and a facial recognition mode. Facial recognition mode uses the camera of the server phone to take a picture of a face and offloads its processing to a remote computer.

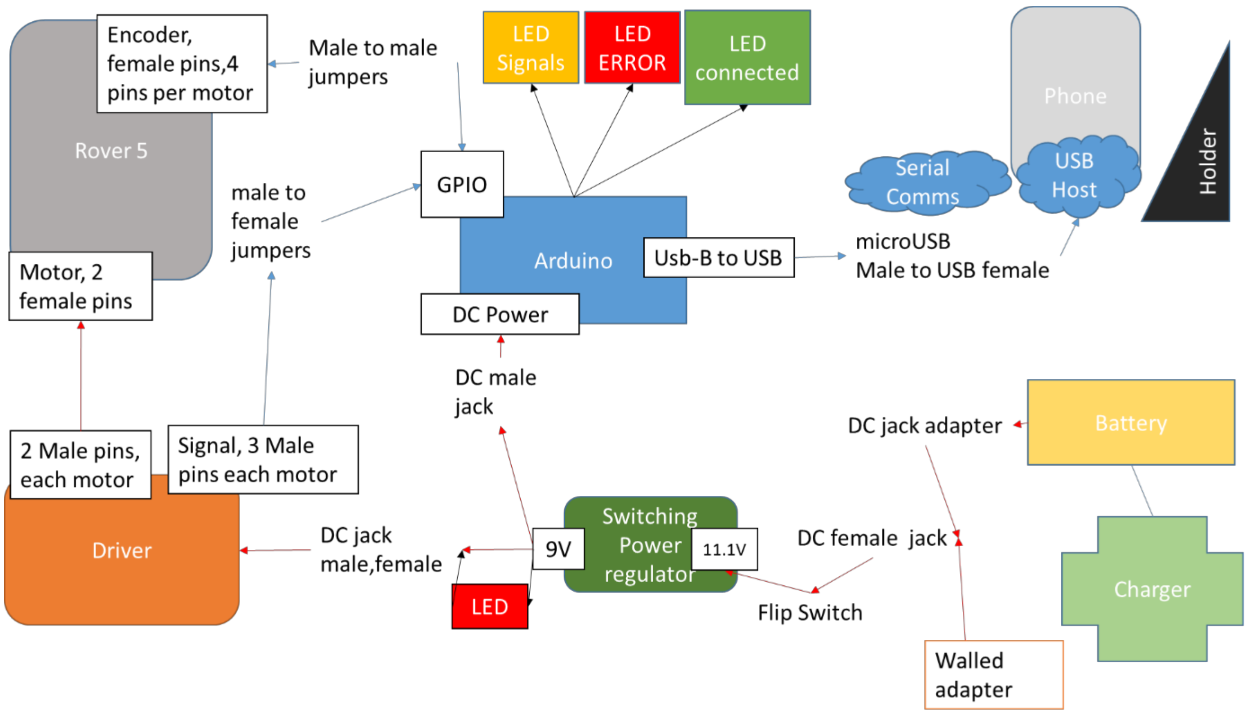


Figure 2: Overall system diagram schematic

Project Aims and Relevance

Using these hardware and software interfaces as a foundation, we seek to show a proof of concept for a social robot. To show some element of people-likeness, the design considerations for the robot are as follows:

- Simple Artificial Intelligence enabling conversational ability
- Emotional detection of human counterparts
- Memory of past interactions and/or ability to learn
- Navigational abilities
- Modular development for future feature augmentation

With these considerations in mind, a use case for the robot would be one in which you could:

1. Introduce yourself to the robot & tell something about yourself
2. Tell the robot how you are feeling & have it react accordingly
3. Say goodbye & part ways with the robot
4. Have a new conversation with the robot & it remembers you

IMPLEMENTATION

Many off-the-shelf software packages are available that can satisfy these needs. However, given time constraints and scope of this project, we kept the design simple and decided to use CMU Sphinx Voice-to-Text Speech Recognition and Android's own Text-to-Speech packages to implement a conversational robot.

Overall Architecture

The system design builds on the framework already provided in that the applications use a client-server architecture for delegating duties. In this design, the user client application implements a Speech Recognition package for interpreting user conversation commands. This client application also contains a decision tree module that chooses what the robot will say when it is spoken to. However, the system is purposefully designed to segregate the intelligence of decision making and the action of speaking. There are various reasons for this, chief being maintainability and ease of upkeep. Therefore, the server implements Androids Text to Speech (TTS) package, in addition to the previous features. The following describes the finer details of all of these modules and instructions for reproducible implementation.

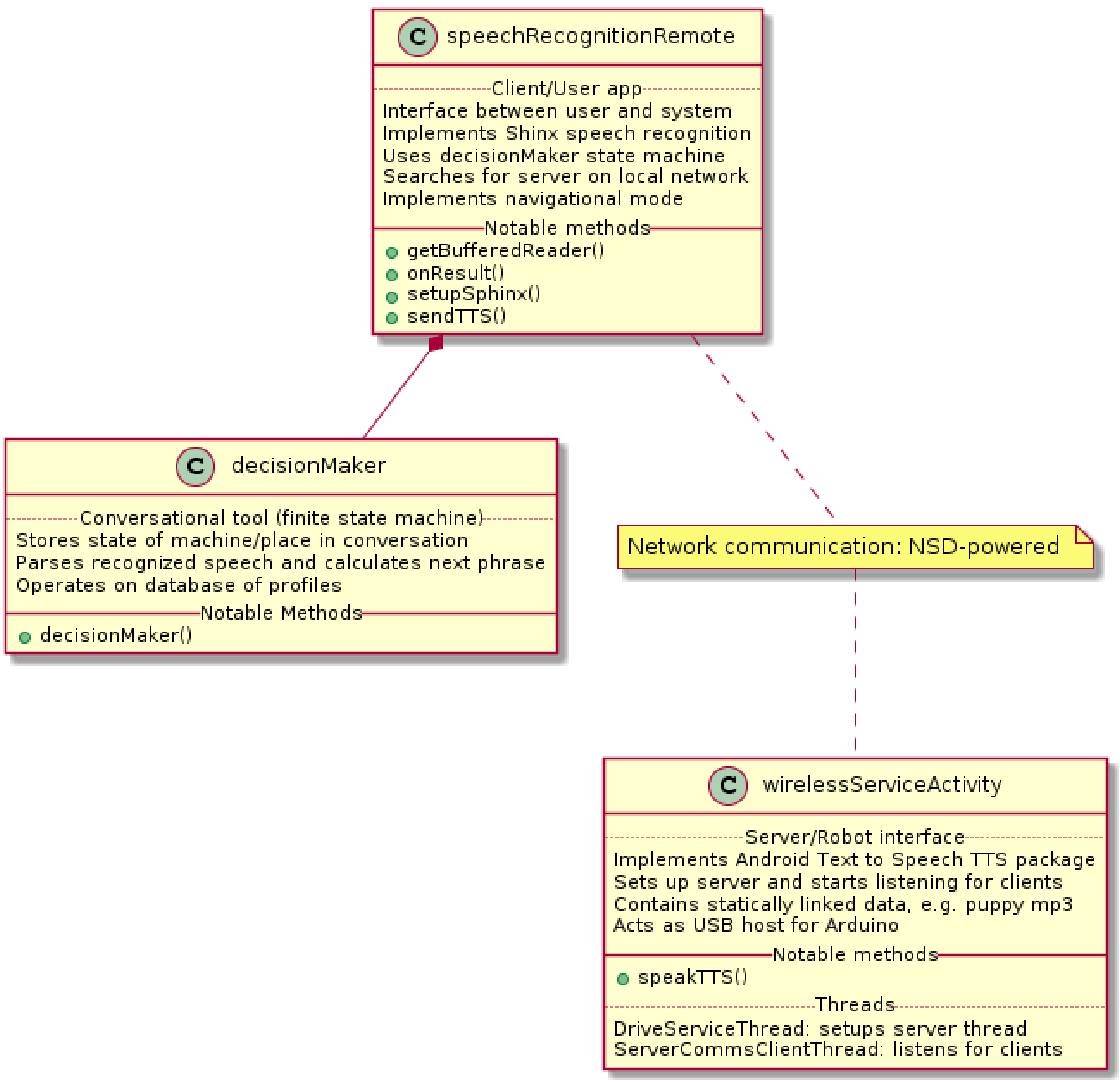
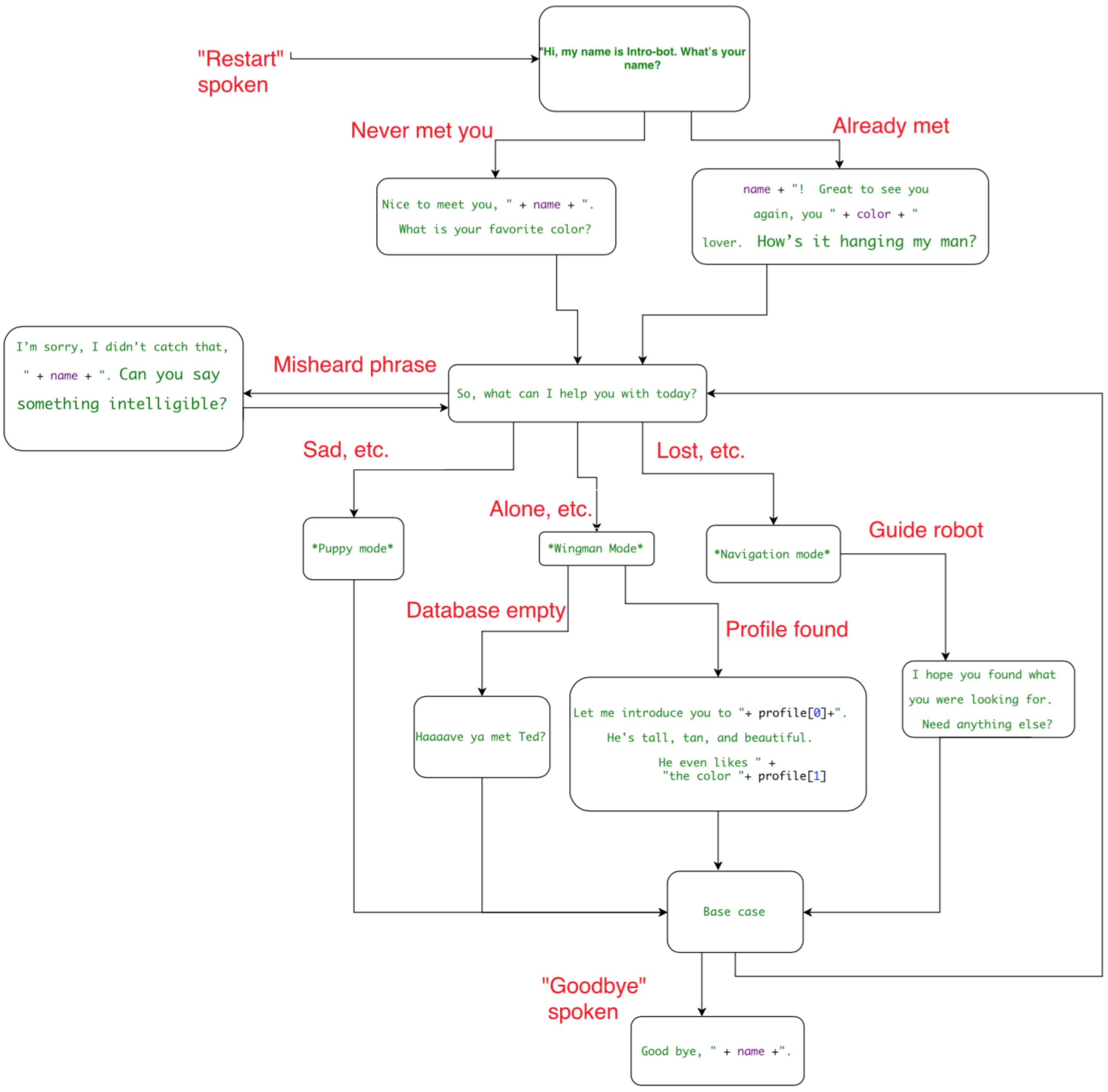

speechRecognitionRemote
Client/User app
Interface between user and system
Implements Shinx speech recognition
Uses decisionMaker state machine
Searches for server on local network
Implements navigational mode
Notable methods
getBufferedReader()
onResult()
setupSphinx()
sendTTS()
decisionMaker
Conversational tool (finite state machine)
Stores state of machine/place in conversation
Parses recognized speech and calculates next phrase
Operates on database of profiles
Notable Methods
decisionMaker()
Network communication: NSD-powered
wirelessServiceActivity
Server/Robot interface
Implements Android Text to Speech TTS package
Sets up server and starts listening for clients
Contains statically linked data, e.g. puppy mp3
Acts as USB host for Arduino
Notable methods
speakTTS()
Threads
DriveServiceThread: setups server thread
ServerCommsClientThread: listens for clients

Figure 3: Class diagram UML (abbreviated).

Decision Maker

After speaking to the robot, a decision tree/finite state machine chooses what the robot will reply. This central logic unit is realized as a finite state machine, where the robot knows its current state, what has been last said, and chooses what to say next based on the users recognized speech. Decisions are made in one of two ways, depending on the state. In a negative search pattern, the recognized phrase is reviewed in its entirety, looking for unwanted words. For instance, in the introduction state, we look for I, am, etc. This is convenient to find names, for instance, since we do not need to search a large bank of names. The other way of searching is positive logic, in which target activating words bring the robot into a state. For instance, the word sad activates puppy state.

The decision maker also encompasses the memory aspect of the robot, which is currently implemented by storing simple user profiles on an internal Android file with a reused name among each run. New profiles are written (appended) to the file and remembered profiles are used to make the conversation more personal.



"Restart" spoken
"Hi, my name is Intro-bot. What's your name?
Never met you
Already met
Nice to meet you, " + name + ". What is your favorite color?
name + "! Great to see you again, you " + color + " lover. How's it hanging my man?
I'm sorry, I didn't catch that, " + name + ". Can you say something intelligible?
Misheard phrase
So, what can I help you with today?
Sad, etc.
Alone, etc.
Lost, etc.
Puppy mode
Wingman Mode
Navigation mode
Guide robot
Database empty
Profile found
Haaaave ya met Ted?
Let me introduce you to "+ profile[0]+". He's tall, tan, and beautiful. He even likes " + "the color "+ profile[1]
I hope you found what you were looking for. Need anything else?
Base case
"Goodbye" spoken
Good bye, " + name +".

Figure 4: Conversational finite state machine diagram.

Voice to Text/Speech Recognition

The speech recognition engine used was CMUs, open source package, PocketSphinx for Android [3]. Sphinx is relatively well documented online in the tutorials and the creator Nikolay Shmyrev is always perusing forums to answer any questions. Using PocketSphinx for Android can be done in one of two ways: using the demo app and building an application within the preprogrammed app, or linking the library and building from scratch. Opening an existing Android app should be no trouble, but linking can present a few bugs if not done correctly, so here are some instructions to best link the library (see http://cmusphinx.sourceforge.net/wiki/tutorialAndroid for more help):

1. Download the PocketSphinx for Android demo app
2. You will need to copy some files from this demo into YOUR application (not the demo)
3. In the demo app file hierarchy, locate the app/libs folder and copy the jar file (presently named pocketsphinx-Android-5prealpha-nolib.jar) into the app/libs folder in YOUR application
4. In the demo app file hierarchy, locate the app/src/main/jniLibs folder and copy it into YOUR app in the exact same location (make it if it doesnt exist)
5. Look in the demo app at the assets directory inside the hierarchy. Copy this assets folder into YOUR app, or simply create an assets section with your own dictionaries, if you so choose.
6. Look in the demo and copy the app/asset.xml file into YOUR application.
7. Copy and paste the following code into the app build.gradle file at the very end of the file after everything else.
   - ant.importBuild 'assets.xml'
   - preBuild.dependsOn(list, checksum)
   - clean.dependsOn(clean_assets)
8. You may need to go into your apps file structure and formally link the library in Android studio by clicking File... Project Structure... Dependencies...

Yes, this appears to be a little roundabout, going into the demo and copying it so methodically, but as of now this appears to be the best approach. Now that the library is linked, there are two ways of making a language model. The first is to build a simple grammar file, which is useful for straightforward communication when there are few words needed and a limited variety of sentence structures. The second way is to train Sphinx with a block of written text, called a corpus, using the ngram statistical language model. Our system was implemented with this second model, and a hand-crafted corpus specific to the type of potential dialogue with the robot (see appendix). Creating the proper language building files can also be tricky, so here is a quick instruction set:

1. Obtain or construct a list of sentences to be used as a corpus and save them in a text editor as a .txt file
2. Using the following tool, obtain a zip of all the needed files for language model construction here: http://www.speech.cs.cmu.edu/tools/lmtool-new.html
3. Install the Sphinx command line tools. This may be tedious as you need to have python and swig installed on your computer. http://sourceforge.net/projects/cmusphinx/files/cmuclmtk/0.7/ (may need to see http://cmusphinx.sourceforge.net/wiki/download).
4. Take the files of the output of the lmtool and run them with the new command line toolkit to obtain the .dmp file: sphinx_lm _convert -i filename.lm -o filename.dmp
5. Place both the .dict file and the .dmp into the assets folder of your app and create filename.dict.md5 and filename.dmp.md5 files with an md5 hash of the filename title. You can obtain this hash with many online editors, e.g. http://onlinemd5.com/

Text to Speech

Using Androids built in TTS package is relatively straightforward and can be best done by following the tutorial by Android or elsewhere [2] [5].

With all of this done, all you need to do is implement the needed callbacks (shown in the demo app) in your application to make the Sphinx engine tick.

RESULTS & CURRENT ROBOT STATE

In its current state, the robot is capable of having a simple conversation and entering one of three action modes. Each mode is activated by a positive logical search as mentioned above e.g. searching for key target activation words. This conversational ability is diagrammed in figure 4.

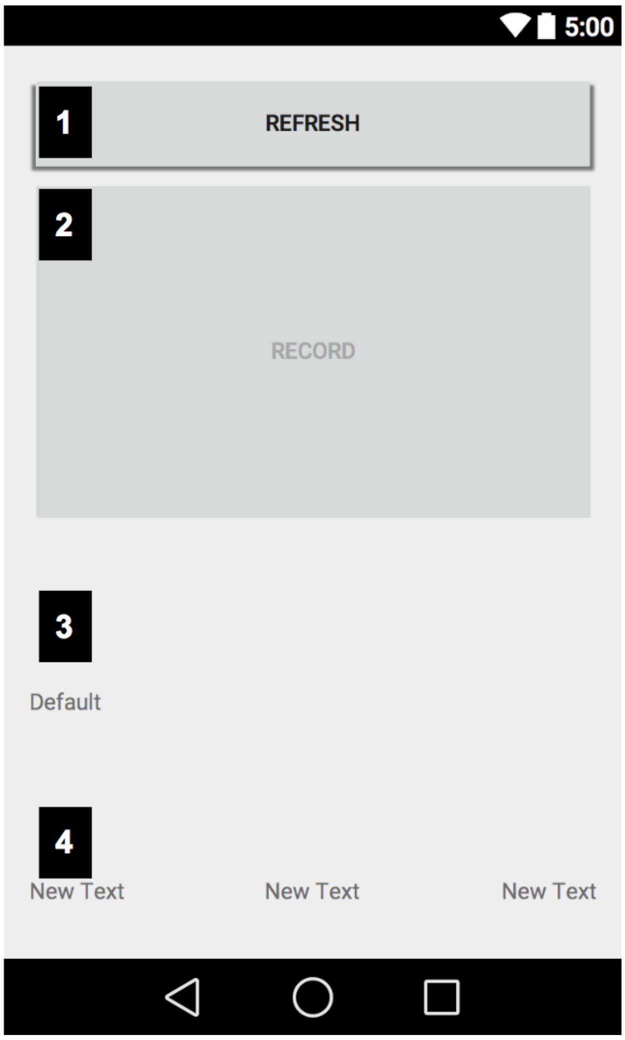


Figure 5: Client-side UI for talking to robot. (1)Refresh button clears and restarts client-server connection (2)When system ready, record button listens to a section of speech (3)TextView shows most recently recognized speech (4)TextView shows current attributes such as speed of robot and server connection information.

Puppy Mode

The first of these modes is puppy mode, in which the robot acts like a little puppy dog, changing its face to a dog, playing a puppy bark .mp3, and moving around sporadically. Puppy mode is triggered when the user alludes to feeling sad or in want of a companion in some way (trigger words: sad, puppy).

Wingman Mode

Wingman mode is a friend-making mode of the robot in which it introduces you to some of the other stored profiles in the database from previous interactions. This mode was motivated by an idea to make a social robot that connects people and augments human-human interaction instead of replacing it (trigger words: lonely, alone, wingman). In Wingman mode, the robot will first attempt to introduce you to one of its stored profiles. If it has never met anyone before, it will act as a standard wingman and speak some pickup line!

Navigation mode

Navigation mode tells you to direct the robot to the nearest landmark to help find out where you are (trigger words: lost, navigate). This mode is somewhat experimental in that it aims to use other future modular integration to resolve a location using image processing and object recognition. See future work section for ideas of next steps for this mode.

Referring to the previously mentioned design considerations, this robot satisfies each in some way. The conversational ability demonstrates a simple AI. The modes demonstrate emotional understanding. Use of the database to store profiles demonstrates memory of past interactions. The robot can navigate. Lastly, the modes are made in a modular fashion that should allow future integration without problem. See below for instructions.

Adding a new mode or conversation

To add new modes to the conversation, open the decisionMaker class and create a new case. Currently case 4 is the base case in which all conversation circles back to. Link a part of the current conversation to your new state by changing the state variable in the previous case to your desired case (e.g. if starting from case 4 and moving to a new case 8, set state = 8 inside of the place in case 4). In the new case, create the desired toTTS string that will be spoken by the robot. You may wish to do something other than just speak so this message may need to contain information beyond the message (e.g. start the string with a special character to be recognized like #). Lastly, open the speechRecognitionRemote class and write any needed code to handle your new message after the call to decisionMaker in getTTS().

FUTURE WORK

Overall, this project is merely a proof of concept and should be used in the future mainly for prototyping new social robotic ideas. There are several immediate shortcomings with the current implementation. We also present here some immediate suggestions for next steps.

Needed improvements

This implementation uses an ngram statistical model with the Sphinx package for both normal talk and navigational mode. I found the grammar file set up to be somewhat buggy, so I decided to use ngram instead for navigational mode. However, this is probably slower and less accurate. So, a future implementation should use a grammar language model for navigational mode.

Additionally, the puppy dance is currently in the server. There is no technical demand for this, and this code should be moved to the client to obey the given architecture principles. Lastly, a larger corpus and more versatile client commands would be useful for a more realistic speech recognition model.

Modular integration

Since this robot is designed to be portable and modular, it is advised to integrate other relevant mobile packages that might be functionally useful. For instance, using the graded speech algorithms developed by Dongwei this summer and some object recognition algorithm implemented by Taiming would give the robot totally new functionality in navigation mode. For instance, you could help the robot navigate to the nearest road sign, and then recognize your location based on a picture taken by the Android phone. This might be done either by offloading the image recognition processing to a remote server, or by using a MATLAB variant on Android like Octave.

Library/API construction

Constructing a general set of functions that package the dirtier details of networking communication, USB talk, TTS and Speech Recognition might be helpful for future development on this robot. That way function can be prioritized over form. A standard robot Android/Java library would do the job.

Autonomous driving

Adding some navigational sensors to the robot would allow implementation of an autonomous bot. This could be done with ultrasonic or optical sensors and/or a camera (perhaps even the Android device camera).

ACKNOWLEDGEMENTS

Thank you to all at NUS who have welcomed me this summer into the lab! I had an exciting time and learned a lot. A special thank you to Tu Fangwen for mentoring me and giving me direction in my project. Thanks to Prof. Ge for allowing us the space to work and for his experienced feedback. And many thanks to Chang Poo Hee for his previous work on the project. Without his previous work on the system, I would need to start from scratch to build any of the components described above. And finally, a thanks to all of the open-sourced library creators used in the project: CMU Sphinx and USB-to-Serial Android libraries.

REFERENCES

[1] Chang, Poo Hee. ”A Mobile Robot Platform Empowered with Android Smartphones.” (2015). Print.

[2] http://Android-developers.blogspot.sg/2009/09/introduction-to-text-to-speech-in.html

[3] http://cmusphinx.sourceforge.net/wiki/tutorial

[4] http://developer.Android.com/develop/index.html

[5] http://tutorialspoint.com/Android/Android_text_to_speech.htm

[6] https://github.com/mik3y/usb-serial-for-Android