GuideBot Project Proposal for ECE578, Fall 2011

Jim Larson, Jules Alfani, Robert Fiszer, Mike Lowe, Hamed Mirlohi

Project Vision

GuideBot will be a robot that can give guided tours of the Engineering Building with stops selected from a menu. Input to it will be via speech recognition and keyboard. Synthesized and recorded speech will be the main outputs; a face will be displayed on the monitor as well. A desirable goal is to have GuideBot show a sense of humor.

Fall,2011 Project Goal

Although GuideBot has been the subject of at least two prior class projects, there was very little usable documentation and many pieces were missing from it. So a major goal for this year is to create a stable, well-documented robotics platform which Future teams can build. This platform will also serve as prototype when requesting grants and funding. The software will be stable and documented. The capabilities we intend to provide are described in the following sections.

Speech Recognition and Synthesis (Jules)

I. SPEECH RECOGNIZER: POCKET-SPHINX

CMU SPHINX, also called Sphinx in short, is the general term to describe a group of speech recognition systems developed at Carnegie Mellon University.
These include a series of speech recognizers (Sphinx 2 - 4) and an acoustic model trainer (Sphinx-Train).

POCKET-SPHINX

A version of Sphinx that can be used in embedded systems (e.g., based on an ARM processor). Pocket-Sphinx is under active development and incorporates features such as fixed-point arithmetic and efficient algorithms for GMM (General mixture model) computation.
In statistics, a mixture model is a probabilistic model for representing the presence of sub-populations within an overall population, without requiring that an observed data-set should identify the sub-population to which an individual observation belongs.

ABOUT SPEECH RECOGNIZER:

Speech Recognizers aid in conversion of spoken words to text. It is another application of Natural Language Processing. Speech recognition engine requires two types of files to recognize speech. The first one is acoustic model, which is created by taking audio recordings of speech and compiling them into statistical representations of the sounds that make up each word (through a process called 'training'). The second one is a language model or a grammar file. A language model is a file containing the probabilities of sequences of words. A grammar is a much smaller file containing sets of predefined combinations of words. The modern speech recognizers are based on Hidden Markov Model. Speech recognizers are used in applications such as Health Care, Telephony, Military, assisting People with disabilities, etc.

POCKET-SPHINX – SPEECH RECOGNIZER USED IN PEOPLEBOT

The speech recognition software used in this project is open source POCKET-SPHINX. It is in the C programming language, making it available to a large variety of development platforms.

GRAMMARS IN RECOGNIZER

Speech recognition systems provide computers with the ability to listen to user speech and determine what is said. In order to achieve reasonable recognition accuracy and response time, most speech recognizers constrain what they listen for by using *grammars*. POCKET-SPHINX supports Word List, JSGF and N-Gram grammars. In this project **N-Gram** and **JSGF grammars** are used.

JSGF Grammar: JAVA Speech Grammar Format is a platform-independent, vendor-independent textual representation of grammars for use in speech recognition. This type of grammar file has '.gram' extension. JSGF gives 80% accuracy for smaller grammar. PeopleBot uses JSGF grammar for all transitions states, including AI Controller.

N-Gram Language Model: It is a subsequence of *n* items (phonemes, syllables, letters, words or base pairs) from a given sequence. An ***n*-gram model** is a type of probabilistic model for predicting the next item in such a sequence. N-grams used in sphinx-4 are files with '.lm' extension. The language model file is generated by a tool (web based) provided by sphinx-4. N-gram is good for large grammars as compared to JSGF, but it performs poorly for small grammars. In this project N-Gram is used for Interaction mode.

CHALLENGES FACED

A recognizer, once created, is attached to a specific grammar and cannot be changed in the lifetime of the program. Such a recognizer will not recognize sentences from other type of grammars. Also, it is not possible to have two recognizers running on same JVM. Except Interaction Mode, People-Bot uses very small grammar, hence only JSGF would suit other modes. In Interaction mode, if only JSGF is used, speech recognition is practically restricted. To overcome this, both N-gram and JSGF grammars are used. N-gram grammar is activated only in Interaction mode and runs in a separate JVM, but both grammars run on same machine. In interaction mode, two recognizers are running and will report the recognized text; to check duplicate processing; People-Bot is designed to respond to only one type of grammar, either JSGF or n-gram, for each recognized text.

Speech Recognition accuracy is approximately 60% in n-gram. We plan to improve n-gram as well as JSGF grammar's recognition accuracy. Machine learning will be incorporated to train the system for an individual's voice and accent.

II. SPEECH SYNTHESIZER: FREETTS

ABOUT SPEECH SYNTHESIZER:

Speech synthesis is an application of Natural Language Processing. It converts text or symbolic linguistic representation (phonetic transcription) into speech. Speech synthesis systems maintain a database of recorded speech either in phonetic form or in the form of entire words or sentences. These recorded pieces are concatenated to produce synthesized speech. The famous scientist Stephen Hawking uses speech synthesizer to communicate.

WHAT IS A TTS ENGINE

A text-to-speech engine is composed of front-end and back-end. Front-end converts raw text of numbers and abbreviations into written words. This process is known as text normalization or tokenization. It then assigns each token a phonetic transcription. The back end, known as synthesizer, converts the symbolic linguistic representation into sound waves. Incorporating vocal tract model and other human voice characteristics are done in this stage.
The current text normalization challenge is to design heuristic techniques to guess disambiguate homographs. For example, in sentences 'Have you read the book' and 'read the book', the word 'read' is pronounced differently. Thus synthesizer technologies are emphasizing their efforts on developing naturalness and intelligibility in synthesis engines.

FREETTS-SOFTWARE USED IN PEOPLEBOT

People-Bot uses text-to-speech synthesizer; hence Free-TTS (an open source text-to-speech synthesizer), written in JAVATM programming language. It is based upon Flite, a small run-time speech synthesis engine developed at Carnegie Mellon University and is built by the Speech Integration Group of Sun Microsystems Laboratories. It uses Hidden Markov Model (HMM) for synthesis. It provides partial support for the Java Speech API (JSAPI). People-Bot uses male US English voice.
Free-TTS is used as is in People-Bot. The goal of the project will be to try to employ machine learning to train a voice to produce sound with emotions.

III. INSTALLATION:

Pocket-sphinx is a library that depends on another library called Sphinx-Base which provides common functionality across all CMU-Sphinx projects. To install Pocket-sphinx, you need to install both Pocket-sphinx and Sphinx-base. It's possible to use Pocket-sphinx both in Linux and in Windows.

First of all, download the released packages pocket-sphinx and sphinx-base, checkout them from subversion or download a snapshot. For more details see download page. Unpack them into same directory. On Windows, you will need to rename 'sphinx-base-X.Y' (where X.Y is the Sphinx-Base version number) to simply 'sphinx-base' for this to work.

Windows

In MS Windows (TM), under MS Visual Studio 2008 (or newer - we test with Visual C++ 2008 Express):

- load sphinxbase.sln located in sphinx-base directory
- compile all the projects in Sphinx-Base (from sphinxbase.sln)
- load pocketsphinx.sln in pocket-sphinx directory
- compile all the projects in Pocket-Sphinx

MS Visual Studio will build the executable under .\bin\Release or .\bin\Debug (depending on the version you choose on MS Visual Studio), and the libraries under .\lib\Release or .\lib\Build. To run pocket-sphinx_continuous, don't forget to copy sphinxbase.dll to the bin folder. Otherwise the executable will fail to find this library.

IV. LINKS:

BUILDING APPLICATION WITH POCKETSPHINX
http://cmusphinx.sourceforge.net/wiki/tutorialpocketsphinx
http://www.mobilerobots.com/ResearchRobots/Resources/Videos.aspx

Route-Finding (Mike)

A Graph data structure will be used to represent the map of locations and routes in the FAB. An exhaustive Search will be used to find location based on user input. The robot will have instructions how to drive to any connected node and know which sequence of nodes to travel if it is not directly connected. These directions will be based on floor plans for the FAB building (http://www.fap.pdx.edu/floorplans/detail.php?buildingID=12). Sonars will be used for obstacle avoidance along the path of the robot, and bumpers will shut the robot off in a worst case scenario. Images from the Kinect will be used as a way to determine the robots current location. Based on the images it should be able to calibrate its position. In the event the robot looses track of its path it should be able to correct itself based on images it is currently receiving. If the robot is completely lost, it should conduct a search routine to find familiar images.

Natural Language Input (Robert)
The Guide-bot should interact with people as naturally as possible. Ideally, this would mean that the robot would interact with humans completely by speech. Although we are planning for this possibility, until the state of the art improves drastically, we will need typed text input as a backup means of communication. Regardless of the means of input, the robot must be able to comprehend what it is told. For this reason, I propose a drastic departure from the previous implementation of language comprehension. Instead of the look up table of sentences used by the previous group, the robot will use grammar and definitions to understand what it is told. In the first step, the program will convert the inputted text to a list of part of speech. It will then use grammars to build a sentence tree. Once it has this tree, the program will then be able to use it to understand what is meant. The two steps that the program will perform will be explained below.

Parts of Speech Example
Input:
Take me to the Intelligent Robotics Lab, please.
The program looks up the part of speech of each word:
transVerb, (pro)noun, preposition, article, properNoun, keyword.
The program checks if the keyword has a special priority. Not finding one it removes it, and finds a grammar rule that begins with “transVerb.” Finds verbPhrase = transVerb + pronoun. The next word is a (pro)noun, which fits the rule, so transVerb and (pro)noun are removed from the list, and replaced with a tree node “verbPhrase” which points to those two words.
verbPhrase, preposition, article, properNoun.
The function is then recursively called. It finds several rules that begin with “verbPhrase,” but none of them match, so the program tries to change the rest of the sentence. It goes to the next word in the list, “preposition.” Again it doesn’t find a match, so it continues to “article.” at article is finds a rule nounPhrase = article + properNoun. It replaces those entries, and backtracks.
verbPhrase, preposition, nounPhrase.
Now, the program finds a rule prepositionPhrase = preposition + nounPhrase, and it replaces those two entries.
verbPhrase, prepositionPhrase.
Now the program can find a rule command = verbPhrase+prepositionPhrase, and replaces it.
command
The program now has a sentence tree instead of a list of words. It can now understand the command.

Comprehension
The program now tries to understand the command, given the syntax tree that it solved earlier.
Take me to the Intelligent Robotics Lab, please.
The program looks up what take “means” for the robot, and finds that it should either carry an object or lead a person.
Lead person or carry object
The program looks up what “me” means and finds that it’s person. “Carry object” is removed as an option.
Lead person
The program explores the prepositionPhrase to find out where it should lead the person.
Lead person Intelligent Robotics Lab
The program now understands what the robot is supposed to do, and gives feedback to the user to signal its understanding.
Output("Please follow me")
The program then executes the command.
Movement(getLocationId (Intelligent Robotics Lab));
The program finds the location id of the location that it is supposed to go to. Then, it orders the motion controller to go to the specified location.

Vision (Hamed)
Use Kinect
Build on work by SreeRam (Include pointers - we may need to put the material in the archive on the Wiki.)

Integration (Jim)
The platform for GuideBot will be PeopleBot as used in the last two projects. The Mobile Robot Programming Toolkit MRPT http://www.mrpt.org/ will be used as the basis for our software. MRPT interfaces to PeopleBot and supports Kinect; speech and language can be integrated.
We will make compromises to build a functioning prototype. While we want to add some nice functionality, we will make sure that what we produce works and is well-documented. Development takes a back seat to documenting what is working.
We intend to ensure documentation is in place, all software is archived, and that all components are clearly identified and owned by the department or easily replicated.
GuideBot will be put on the internet using PAVE website. Currently, PAVE is only available inside PSU - maybe only inside the Engineering Building and FAB.

Status
PeopleBot (the robot platform that is the basis of GuideBot) sensors have been tested.

Ultrasonic sensors now work in front and rear of robot. At the heart of GuideBot will be a Zotac MiniATA motherboard. This board is functioning with Linux (Lucid) installed on borrowed SATA drive The. 12V power supply for using battery power has been tested and works. A USB track ball would be a good addition, since there really isn't a place to use a mouse then the robot is assembled. The wireless internet works is working.
The MRPT robotics software is installed and has been used to control the Kinect. A Kinect vision system has been purchased. Manuals for all items are (or will be) on the Wiki.

Team members are gathering software packages and doing initial testing of demo programs as a basis for future development.