Integration of the FONIX Speech Recognition library to the Prof. Perky Humanoid Robot Project.

Final Report

for

Intelligent Robotics I (ECE 578)
Portland State University, winter 2003/2004
Author: Stefan Gebauer

1. Abstract

In 1921, when the Czech author Karel Capek produced his best known work, the play R. U. R. (Rossum's Universal Robots) he created the word "robot" to give the intelligent machines a name. The actual meaning of the word robot was derived from the Czech word robota which means "forced worker". Since then, a lot of Science Fiction authors used the story and created a picture of the humanoid robot as it is in our heads nowadays. A fundamental feature of such a robot is an own voice as well as the capability to understand a human which speaks to the robot. Hence, Speech Recognition is one of the basic and necessary components of the auditory system of a robot. The auditory system is the interface between humans and robots. On the other hand it may be used for teaching purposes. This report deals with the integration of Speech Recognition to the PSU Prof. Perky Humanoid Robot Project.

2. Project Overview

The architecture of the whole project is depicted in Fig. 1. The box with the grey background indicates the modules, which I'm responsible for. At this time I describe the recognition module, which is the path between the microphone and the interpreter. Last term I programmed the Text-To-Speech (TTS) module. When I tested it, it turned out that it still has a bug. (it speaks a little bit more than it is supposed to speak. It doesn't really make sense, what it says additionally.) But it works though. I think I should add Fonix's TTS library to the module as a future project. Fonix TTS has a good quality and comes along with a bunch of male and female voices.

The actual reason for going to do this can be seen in Fig. 1. The synchronization of the robot's lips is not easy to solve with Microsoft's Speech API 5.1.Whereas Fonix TTS API is not that much encapsulated like MS SAPI is. The programmer

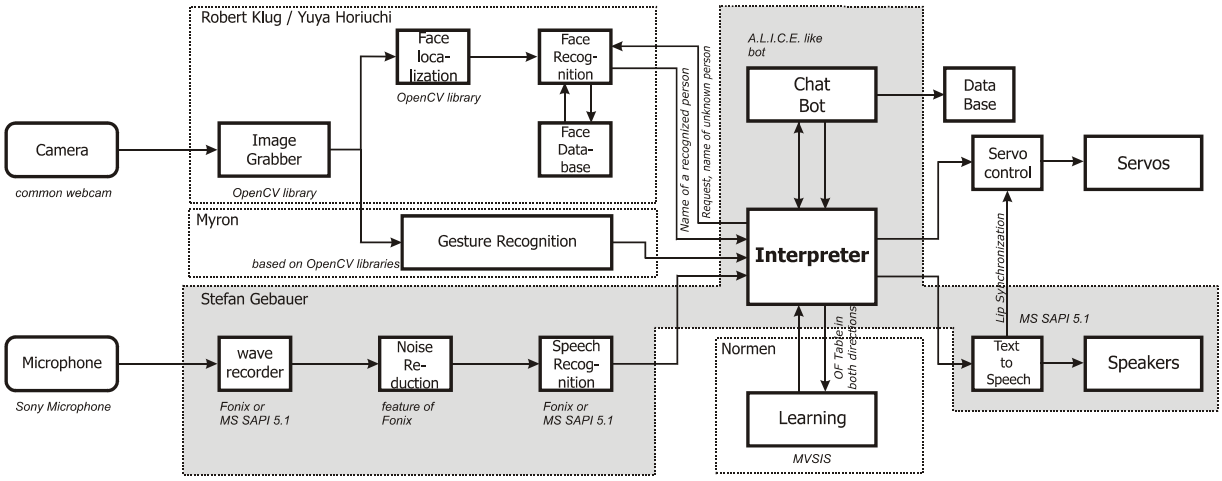

Fig. 1 Architecture of the Prof. Perky Robot

has access to main TTS functions. I hope to be able to solve the synchronization issue with Fonix TTS.

The interpreter plays a central role. There, all information get together. The interpreter evaluates it and sends corresponding actions to the single modules. The Interpreter is not designed so far. We are going to have a database which stores information about people the robot will interact with. At this point, I'm not sure where to put the database. Should rather the interpreter have access to the database than the chat bot, like it is shown in the picture?

In any case, the interpreter will have a link to a learning module. The communication is based on the OF-Table[4,5]. The communication between the face recognition module and the interpreter will be kept simple. The interpreter will request face recognition and the module will follow the command. After a successful detection the module sends the name of the person. If the person is unknown the interpreter requests the TTS module to ask for the name of the person. The speech recognition module will be activated by the interpreter. Eventually the name of the person will be obtained and sent to the face detection module which stores the name in its own database (compare with Fig. 1). For the pragmatically implementation of the interpreter, thread programming will be highly recommended.

3. Why did I decide for Fonix?

There are a lot of reasons. First, after a couple of tests, it turned out the recognition performance of Microsoft's SAPI is fairly poor. Second, the MS SAPI comes as a whole encapsulated package. It is hard to customize it or get control to core functions, if it is possible at all. Third, there is already some knowledge about Fonix Embedded Speech SDK 2.1 [2]. Hung et al. evaluated the recognizer [3]. When I'm comparing the results of MS

SAPI with those that they got, the superiority of Fonix's recognizer is obvious. Key elements of its capabilities are:

- Speaker-independent or dependent, as desired
- Performs with background speech and music
- Small memory footprint
- Computing capacity is adaptable (starts at 20 MIPS)
- Dynamic optimization--minimize memory and MIPS
- Word spotting
- Letter recognition
- Finite state grammars
- Far-field microphone

4. Description of the Speech Recognition Module (Technical)

The flow chart of the Recognition module can be seen in Fig. 3Fig. 2. The source code of the project has been moved to a separate document (appendix.doc) which makes the project report document handier. The first part of the code initializes and sets the directories. The shared path needs to be set. This done by the Fonix API `FnxCoreSetSharedPath`. The shared path contains the dictionary file (.dcc), the ASR (automatic speech recognition) rules file as well as the neural net files for digits (to recognize numbers) and the general neural net file to recognize words. In essence by setting the shared path means to select a certain language. Fonix supports UK English and German apart from US English.
The dictionary file (.dcc) can be a the large vocabulary dictionary, like the one that provides Fonix, or a smaller vocabulary dictionary created with the *DccCreate* tool and containing just the

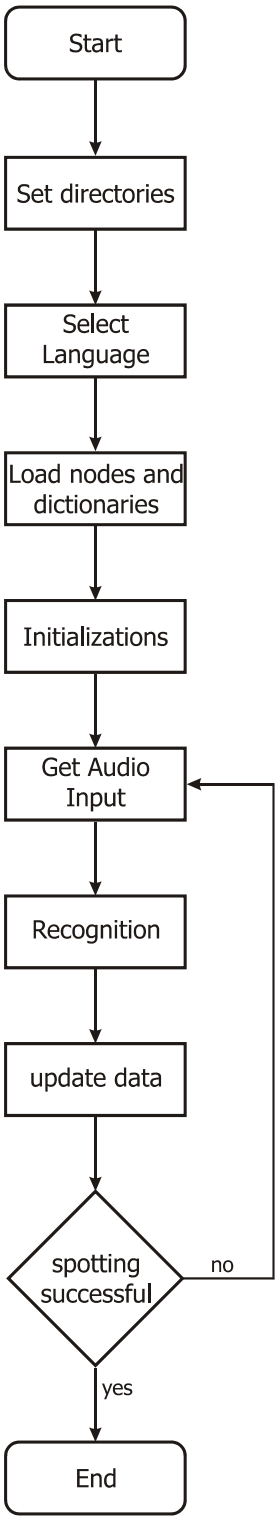

Fig. 2: flow chart of the ASR module

required words.Valid dictionaries have a

File Extension	Description
.dic	Limited-vocabulary TTS file. Used by the FonixCore API. Contains a fixed number of phrases and their pronunciation. Can be created by the Fonix Builder program or by Fonix by special request.
.dcc	ASR phoneme file. Used by the FonixCore API. Contains a fixed number of words and their phonemic representation. Large versions of the dcc files are provided; however customized ones with a smaller subset of words can be created using CreateDcc.exe.
.mtx	ASR confustion matrix file. Used by the FonixL2WAsr API. Contains data used to accomplish letter-to-word recognition.
.pni	ASR neural network file.Used by the FonixCore and FonixL2WAsr API's. Contains language-specific neural network data used in recognition.
.rules	ASR rules-based recognition file. Used by the FonixCore API.Contains rules which may be used to derive phonemic representations of words not found in the .dcc file.
.suffix	ASR suffix recognition file. Used by the FonixCore API. Contains suffixes which can assist the rules-based recognition.
.vocab	ASR vocabulary file. Used by the FonixL2WAsr API. Contains the vocabulary and pronunciations of words used in letter-to-word recognition.

Table 1: FileTypes: Description of the various file types and file usage by extension. [1]

.dcc extension and must be located in the shared path. How to create a custom dictionary is described in section 6.
The rules file is used to perform ASR for words not found in the ASR dictionary. Rules files must be located in the shared path, too.

Digit nodes recognize a combination of digits. Digit nodes use the digit neural net for recognition. The general purpose neural net is used in all non-digit nodes.
The speech recognizer is then initialized by making the files known to the instance through the Fonix ASR APIs `FnxCoreSetDigitNNetFile`, `FnxCoreSetGeneralNNetFile`, `FnxCoreSetAsrDictionaryFile`, `FnxCoreSetAsrRulesFile`, respectively.
The next step creates the nodes. By using the following API functions, nodes can be created `FnxCoreCreateDigitNode()`, `FnxCoreCreateWordNode()`, `FnxCoreCreateKeyWordNode()` and `FnxCoreCreateGrammarNode()`, respectively. I use the API functions `FnxCoreCreateDigitNode()` and

`FnxCoreCreateWordNode()` in the recognition module. The DigitNode function gets a grammar array (compare with source code, explanation of a grammar file can be found in section 7). The other function takes a word list. The word list represents a subspace of the dictionary. Here, I pass only some colors. The WordSpotter node then, recognizes a word from the list of words which has been passed.

FnxCoreRunNode runs the given node. Audio input is gathered, and recognition takes place depending on how the node has been configured. The whole process is hidden to the user and it is not influenceable by the user. After a recognition node is run, FnxCoreGetFirstAsrResult retrieves the highest scoring word from the subset of recognizable words for that node, or the very first spoken word for nodes which can return multiple valid words. The total word score is calculated using wordScore - garbageScore. The pBeginFrame and pEndFrame provide relative locations of where the recognized word occurred within the entire utterance [1]. `FnxCoreGetNetxtAsrResult` either retrieves the next highest scoring word from the subset of recognizable words for that node, or the next spoken word for nodes which can return multiple words like a special grammar node. I used the method for the number node (compare with source code). The word spotter node works only with `FnxCoreGetFirstAsrResult()`.

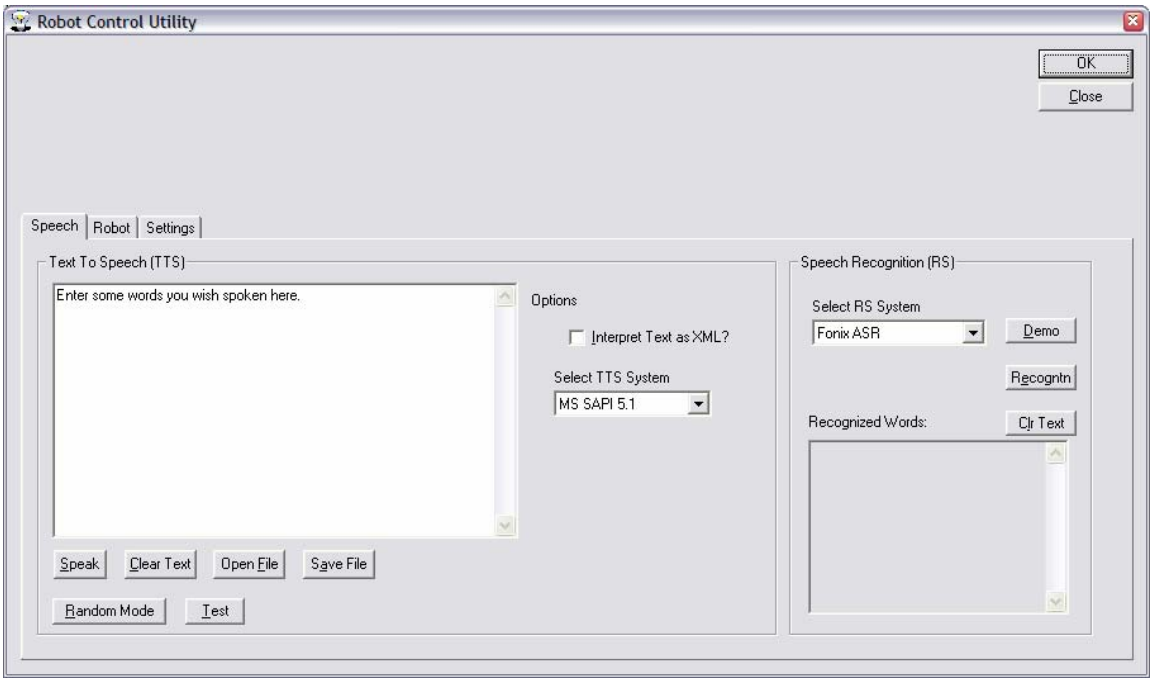

Fig. 3: The Robot Control Utility with the new ASR feature

5. Description of Speech Recognition Module

Because the module isn't connected to another module, I decided to display the output just on the screen. As in Fig. 3 shown, the dialog has been enhanced by an edit box, named "Recognized Words", three buttons and a drop down menu. In the drop down menu, the user can choose the speech recognition system. Right now, only Fonix works. The Demo button (I added the source code to the appendix file) executes a demo application that loads a project which has been created by the Fonix ASR Builder. This is a GUI based application which simplifies building simple ASR projects [3]. By clicking the "Recogntn" button the recognition program will be invoked. It calls the functions that I explained in the previous section. At first, it asks the user to say some numbers between 0 and 9. Results will be displayed in the edit box. If the user says three times in series 999, the program will terminate and the next program "color recognition" starts. It is only possible to recognize colors, that was making known to the program . Results will be displayed in the edit box, as before. The program will be terminated if the user says "exit". After each recognized word, a number in parenthesis is printed. A high number is good and means the word was recognized correctly with a high probability.

6. Create a User Defined Dictionary

The Fonix Automatic Speech Recognition (ASR) Dictionary Tool (Fig. 4) helps to create custom ASR dictionaries. It is useful to create sub dictionaries with smaller word lists. The ASR Dictionary Tool takes two inputs (Fig. 4). The word list text file is just a text file containing the desired words in it. Usually, there is one word at each line. The Large ASR dictionary is a file with *.dcc extension that comes along with Fonix. The file can be found in %fonix-installation-path%\fre\fd01\english\asr. There is a dcc file for each language version (German, UKenglish).

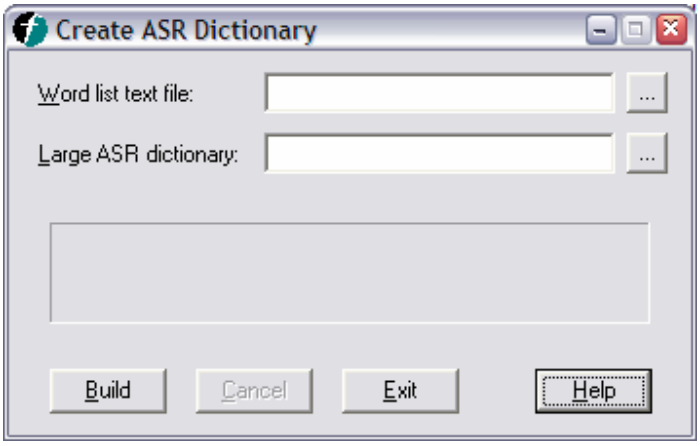

Fig. 4: Tool to create custom dictionaries

Eventually, the Tool looks up and searches in the Large dictionary for the word which were given in the word list file. If they were found it creates two files. First the customized dictionary (*.dcc) that contains the word list and additional ASR information. Second a phonetic content text file with an extension of *.phon.txt that contains the specified word list and their corresponding phonetic symbols. [1]
The method of creating smaller user defined dictionaries is very powerful. Thus, it will gain the recognition rate since the space for finding the correct word is smaller.

7. Grammar Nodes

Due to processing time, it only is possible to spot words. Thus, it is not possible to get complete sentences from the recognition module. However, a Grammar node can extract a series of words from a speech utterance, according to the set of rules that are defined by the grammar. The rules may dictate sequence, repetition, groupings, optional items, and Boolean expressions. Recognition words can be mapped to a value or word. Grammar syntax definitions are divided into six categories: variables, repetition, grouping/optional items, Boolean, output directives, and keywords. You will use these syntax definitions as you create a grammar [1]. The API prototype is defined as:

```
int FnxCoreCreateGrammarNode ( FnxCore, char * szNodeName, char * szGrammar )
```

Lets see an example for creating a grammar for a receptionist in the university. Creating a grammar is a kind of solving a story problem. For example, one could ask for directions or could inquire for faculty staff. For simplicity, say someone asks for directions to find room 155 and the office in the ECE building. Another person asks for Dr. Morris and Dr. Perkowski.

The word sets look then like:

inquire set: where, is, looking for
Person set: Dr. Morris, Dr. Perkowski
location set: Office, 155

Now, the word sets have to be assigned to a variable. It is a sort of pruning the space. For each variable, there are only a few possibilities. Note: I introduce more possibilities for saying 155:

$inquire = where is looking for;
$person = Morris Perkowski;
$location = office 155 one fifty five one hundred fifty five;

The relationship between the words in a group must be expressed in terms of Boolean relationships. There are two Boolean characters defined, OR (|), AND (a space):

$inquire = where | is | looking for;
$person = Morris | Perkowski;
$location = office | 155 | one fifty five | one hundred fifty five;

In some cases it may be useful to assign to the output another word as the recognized output. In the case above I do it with “one fifty five” and “one hundred fifty five” both are the same number 155:

$inquire = where | is | looking for;
$person = Morris | Perkowski;
$location = office | 155 | one fifty five%155 | one hundred fifty five%155;

Finally, the grammar sequence has to be created.A person who's looking for Prof. Morris or Prof. Perkowski could ask the following sentences:

Where is Prof. Perkowski?
Is Prof. Morris in his office?
I'm looking for Prof. Perkowski?
Do you know where Prof. Morris is?

Hence the grammar of those sentences may have to following structure:

$grammar = $inquire $person [$location].

The last variable is in brackets. This indicates location is optional which makes sense because not all of the sentence may have a location word in the question.

Same can be applied to the direction questions like.

Where can I find room 155?
Where is the ECE Office?

$grammar = $inquire $location.

8. Results

So far, I can demonstrate that the Fonix ASR module works. The results are much better than with Microsoft's SAPI 5.1. With a word list representing a subset of the main dictionary the recognition performance is quite good – even with a cheap microphone. I was able to show how to recognize numbers, words from a certain list and eventually how to put projects that have been developed by Fonix's ASR Builder to the robot utility. Hence, it is possible to develop speech applications easily with the Builder and make later on the whole functionality available to the robot utility.

8. 1 Limitations and Extensions

So far, the whole program depends on speaker independent recognition which seems to be practical. In most cases the person who talks to the robot will change. However, speaker dependent speech recognition might be practical, too. Caregivers (persons who teach the robot and evaluate new functions) will appreciate this function since the recognition rate will be higher. To implement the feature, the usage of `FnxCoreGetUnrecognizedRawData` or `FnxCoreGetRawData` is useful. The raw data can be immediately passed to `FnxSDAsrRecognizeFromRaw` to achive speaker-dependent recognition on the data. More details about that can be found in [1].

Performance is still an issue. I run the ASR program on my INTEL Pentium M 1.4 GHz. Even though we're only spotting a single word, the processing load is high. Up to now, I integrated the standard recognition routines only. There are optimization methods available but this means in essence a reduced complexity of the neural nets (`FnxCoreSetGeneralNNetFile( pCore, GEN_NNET, 1 )`, If optimizeSpeed is `1` the neural net will be pruned for faster recognition and a smaller footprint. A smaller footprint will have decreased ability to accurately reject out-of-vocabulary words. [1]). As a result, the recognition accuracy will be lower [1]. I'm integrating this feature and it will be available in the next version.

Every system that obtains analog data has to struggle with noise. Noise is always in the signal and affects the recognition rate. FONIX Embedded Speech SDK was designed for noisy environments such as hands free kits in cars. However, there is another kind of noise - more than one voice at a time. It really influences the system and it recognizes words that it is not supposed to recognize. As Hung Nguyen et. al [2], I made some tests in multi voice environments, with some microphones. In addition they tested the recognizer in a car and figured out that the microphone array is very useful to reduce noise. I used for the tests the Sony ECM R-100 Microphone and the built in microphone of my Laptop. The first evaluation environment was the room 155 in the PSU FAB where we used to have our Friday meeting. Other people were speaking when I was running the speech recognition module. The second place was my room. There were three people talking in the background when I was recognizing words. In both situations, it turned out that the recognition accuracy was not very much effected by the microphone. It was almost the same. But the background voices were grabbed and occasionally there were words recognized that I didn't say.

There might be two approaches to work around this. First, as introduced two paragraphs above, speaker dependant recognition could have the same affect as a filter. The recognition module is then more sensitive to a particular voice. Moreover the speaker dependant data could be loaded when the face recognition module knows the name of the person. This approach is very natural, since everyone knows the voice of a familiar person. Second possibility is to use a microphone that only obtains data from a smaller area (high impedance mic). A head set microphone seems to be the best solution for that.

The recognizer will have a hard time to get the right word for all those words which are not in the dictionary file. Our program could encounter this problem when it asks for names of persons. The recognizer wouldn't recognize the words properly. However, Fonix provides a special API, which may be useful to get around. The Letter to word ASR API supports recognition of words spelled letter by letter. The neural nets for this API are trained for alphabet letter recognition. A user who is communicating with the robot could tell the robot his name. Even though the name is very exotic like names of some foreigners, the name will be correctly recognized by the system. Once, the name is stored in the system, the TTS system will be able to pronounce correctly the name with its rules (English).

For the future, I'll wrap the most essential functions in a C++ class. I'm going to develop some applications for it like the scissor, rock and paper game. With a compact C++ code in an extra file, I'll be able to create a clear file with the mentioned game. Therefore, it will be easy for other users to add more games.

9. Literature

[1] FONIX *Embedded Speech SDK 2.1 ASR Help*, 2002

[2] Hung Nguyen, Phuong Than and Honghuong Nguyen, "*External Design Documentation of Direction Software Version 1.1*", Portland, OR, winter 2003

[3] Hung Nguyen, "*Speech Recognition Report – Fonix Product Evaluation*", Portland, winter 2003

[4] Stefan Gebauer, "*Project Report ECE 572 – Advanced Logic Synthesis*", Portland 2003

[5] Normen Giesecke, "*Project Report ECE 578 – Intelligent Robotics I*", Portland 2004