SAPIENZA
UNIVERSITÀ DI ROMA

CMU Sphinx: the recognizer library

Authors:

Massimo Basile
Mario Fabrizi

Supervisor:

Prof. Paola Velardi

01/02/2013

Contents

1 Introduction

In this report we aim to provide an easy guide for the basic use of Sphinx. After a brief contextualization of the work, Sphinx configuration and its relatives usage problems will be treated in detail.

Olympus is a complete framework for implementing spoken dialogue systems. It was created at Carnegie Mellon University (CMU) during the late 2000's and benefits from ongoing improvements in functionality. It is organized into Galaxy servers (or agents) communicating through a Galaxy hub as in Fig. 1; each server has a proper task and contribute for the global task, understanding spoken phrases and generating spoken answers. The following are the main Olympus agents:

- **Sphinx:** the recognition server, it recognises the source utterances and selects the best hypothesis for it.
- **Phoenix:** language understanding server that assembles grammar by concatenating a set of reusable grammar rules.
- **Helios:** confidence annotation server where the score reflects the probability of correct understanding.
- **Ravenclaw:** dialogue management server that interprets the input in the current dialogue context and deciding which action to engage.
- **Rosetta:** language generation server that uses the same grammar rules of Phoenix, it expresses in spoken language the action the system needs to perform.
- **Kalliope:** speech synthesis server, that transforms the output sentence of the system in real voice to communicate with the user.

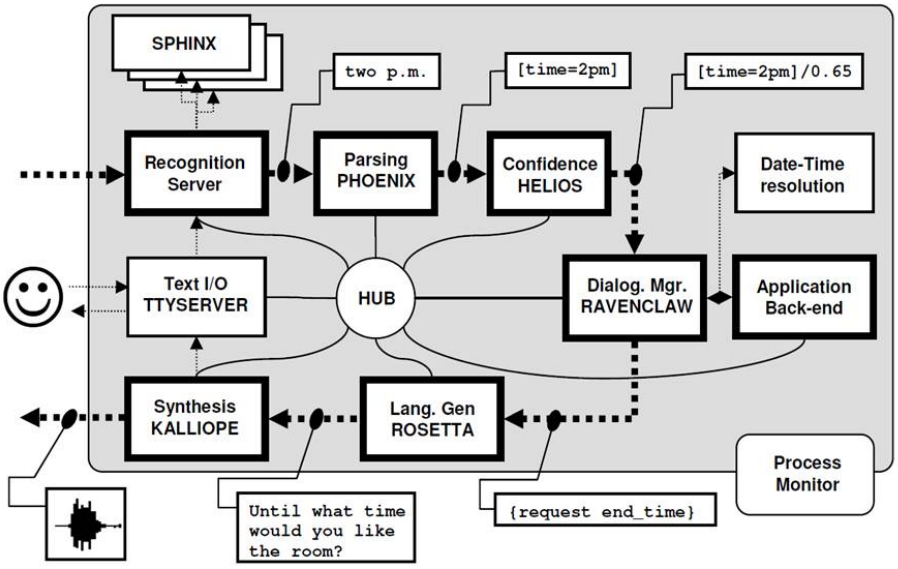


Figure 1: Olympus architecture.

From now on, we will completely focus the attention on the recognition of spoken language. Sphinx is the recognition server, it permits to decode an audio stream and collect a set of best hypotheses about the audio acquisition. To understand utterances Sphinx uses a specific language model adapted to the language and to the context of the conversation. The language model is based

on the n-gram model, a type of probabilistic language model for predicting the next item in such a sequence, obtained from a large collections of text or transcribed speech.

The CMUSphinx tool kit is a speech recognition tool kit with various tools used to build speech applications; it has a number of packages for different tasks and applications. We only use the first two packages of the list, but in order to make the discussion more complete as possible all the packages are going to be presented:

- **Pocketsphinx:** a recognizer library written in C.
- **Sphinxbase:** a support library required by Pocketsphinx.
- **Sphinx4:** an adjustable, modifiable recognizer written in Java.
- **CMUclmtk:** a tool to easily create a language model.
- **Sphinxtrain:** an acoustic model training tools.
- **Sphinx3:** a decoder for speech recognition research written in C.

The report is organized as follows:

- **Sphinx download and installation:** how to configure the libraries on a 64-bit Windows 7 machine with Microsoft Visual Studio 2010.
- **Easy application for speech recognition:** how to develop a code able to understand a sentence registered on an audio file.
- **Conclusions and future works:** Sphinx valuation and considerations about its utility.

2 Sphinx download and installation

Instructions are different according to the operative system one is going to use, in fact CMUSphinx package works on Windows, Linux-like and i-phone operative systems. As already said in this report only the Windows configuration will be treated even if the Linux-like systems are the recommended ones.

2.1 Download

The download of the released packages Pocketsphinx and Sphinxbase can be done from the web [1]; the latest available releases (sphinxbase-0.8 and pocketsphinx-0.8) are recommended. Then, after the download one needs to unpack them into the same directory and rename 'sphinxbase-X.Y' (where X.Y is the SphinxBase version number) to simply 'sphinxbase' for this to work.

2.2 Installation

Pocketsphinx is a library that depends on another library called Sphinxbase which provides common functionality across all CMUSphinx projects. To install Pocketsphinx, one needs to install both Pocketsphinx and Sphinxbase. The installation steps are the following:

- Open "sphinxbase.sln" located in the sphinxbase directory with Visual Studio.
- Compile all the projects.
- load "pocketsphinx.sln" in the pocketsphinx directory.
- Compile all the projects; to do this one need to add all the inclusion paths to Sphinxbase: at the **sphinxbase\include** folder in *project→pocketsphinx Properties→Configuration Properties→VC++ Directories→Include Directories* and at **sphinxbase\bin\Debug** and **sphinxbase\bin\Release** in *project→pocketsphinx Properties→Configuration Properties→VC++ Directories→Library Directories* for all the projects within pocketsphinx.sln.

After this, Sphinx libraries are correctly configured and ready to be used.

3 Easy application for speech recognition

3.1 How to write the code

Now we are ready to write a simple code to extract the best utterance hypothesis from an audio file, more precisely a .raw file. We will create a C source file called "audioRecognition.cpp" into a Visual Studio project. The first thing to tell is that every function of the pocketsphinx library takes a *ps_decoder_t** as the first argument that initializes the decoder structure from a configuration object whose creation will be explained in detail.

The beginning of the code is in Fig. 2. The *cmd_ln_init()* function is used to create the configuration object "config", this function takes a variable number of null-terminated string arguments, followed by NULL. The first argument is any previous *cmd_ln_t** which is to be updated. The second argument is an array of argument definitions, the standard set can be obtained by calling *ps_args()*. The third argument is a flag telling the argument parser to be strict, if this is TRUE, then duplicate arguments or unknown arguments will cause parsing to fail. MODELDIR is a define that contains the path to the language model file (.DMP) and the dictionary file (.dic) because the decoder structures need them to be configured. Finally the *psinit()* function initialize the decoder through the configuration object.

```
#include <pocketsphinx.h>

int
main(int argc, char *argv[])
{
        ps_decoder_t *ps;
        cmd_ln_t *config;

        config = cmd_ln_init(NULL, ps_args(), TRUE,
                             "-hmm", MODELDIR "/hmm/en_US/hub4wsj_sc_8k",
                             "-lm", MODELDIR "/lm/en/turtle.DMP",
                             "-dict", MODELDIR "/lm/en/turtle.dic",
                             NULL);
        if (config == NULL)
                return 1;


        ps = ps_init(config);
        if (ps == NULL)
                return 1;
}
```

Figure 2

Now, the next step is to decode an audio file, more precisely a single-channel (monaural), little-endian, unheadered 16-bit signed PCM audio file sampled at 16000 Hz, then a .raw file. To do it, after the file has been opened with *fopen()* and an handle f_h to this file is returned, pocketsphinx provides a lot of functions. The specific piece of code is in Fig. 3. The function *ps_decode_raw* is used to decode the file raw, while to get the best hypothesis, *ps_get_hyp()* can be used.

```
int rv;
rv = ps_decode_raw(ps, fh, "goforward", -1);
if (rv < 0)
        return 1;
char const *hyp, *uttid;
int32 score;

hyp = ps_get_hyp(ps, &score, &uttid);
if (hyp == NULL)
        return 1;
printf("Recognized: %s\n", hyp);
```

Figure 3

3.2 How to build the code

If someone tries to build the code, it probably doesn't build because some references to sphinxbase that is used to pocketsphinx are missing; so we need to add:

- The path to **pocketsphinx-0.8\include** and **sphinxbase\include** in *project→Properties→Configuration Properties→VC++ Directories→Include Directories*.
- The path to **sphinxbase\bin\Debug**, **sphinxbase\bin\Release**, **pocketsphinx-0.8\bin\Debug**, in *project→Properties→Configuration Properties→Linker→General→Additional Library Directories*.
- The additional libraries dependencies **sphinxbase.lib** and **pocketsphinx.lib** in *project→Properties→Configuration Properties→Linker→Input→Additional Dependencies*.

3.3 How to obtain the audio file and the language model

There are two types of models that describe language, grammars and statistical language models. Grammars describe very simple types of languages for command and control, and they are usually written by hand or generated automatically with plain code. There are many ways to build the statistical language models: if the data set is large, one should use CMU language modelling tool kit, if a model is small, an online quick web service is recommended and in the end if one need specific options one can use him favourite tool kit which builds ARPA models.

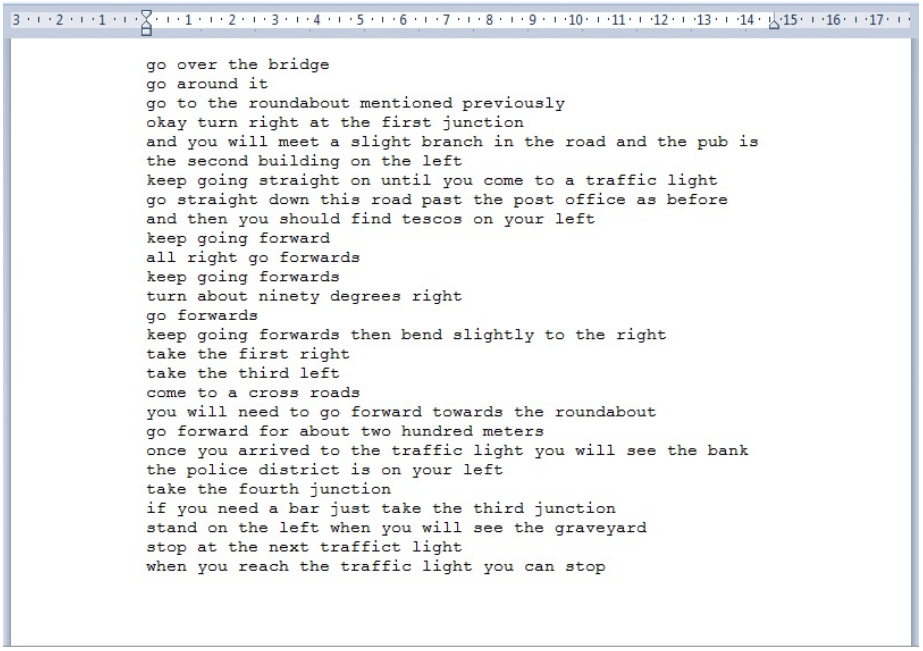
In our report the language model is quite short so we used an online web service, and it is what we did: we first create a corpus that is just a list of sentences that will be used to train the language model. Then when the corpus was completed, we found our web service on the page: http://www.speech.cs.cmu.edu/tools/lmtool.html; it is easy to use, we simply clicked on the Browse button, select the *corpus.txt* file created, then clicked COMPILE KNOWLEDGE BASE, and the language model is built. We downloaded these files, that have a name of 4-digit number followed by the extensions .dic and .lm. The grammar file is s ready to be inserted in the project, while the language model is not ready, because the project needs of a .DMP file. To convert an .lm file in a .DMP format we used the *sphinx_lm_convert* application, just typing on the prompt *sphinx_lm_convert -i <modelName>.lm -o <modelName>.dmp* going into the correct folder. Now, the last thing to do is to insert the grammar file and the language model file into the project just pasting these files in the folder *model\lm\en* of pocketsphinx. Obtaining the audio file is very simple; google provides a tool that allows to pronounce an English sentence and download the .mp3 file. As just said our application decodes only .raw files, so we converted the .mp3 file in the .raw format through Switch Sound File Converter freely downloaded from the web.

3.4 A real application

Now we can finally tell about our application and in section 4 we will tell about te results we have obtained. To implement the application we have closely followed what we explained in this section; the code is quite similar to Fig. 2 and Fig. 3. The dictionary and language model files have been obtained through the online server recommended in section 3.3, with the .txt corpus handwritten by us in Fig. 4.

We tested our application on a testing set made of 100 sentences which is shown below:

1. KEEP LEFT BEFORE THE POST OFFICE
2. TURN RIGHT AT THE ROUNDABOUT
3. GO STRAIGHT ON AT THE TRAFFIC LIGHT
4. TAKE THE SECOND JUNCTION ON YOUR RIGHT



go over the bridge
go around it
go to the roundabout mentioned previously
okay turn right at the first junction
and you will meet a slight branch in the road and the pub is
the second building on the left
keep going straight on until you come to a traffic light
go straight down this road past the post office as before
and then you should find tescos on your left
keep going forward
all right go forwards
keep going forwards
turn about ninety degrees right
go forwards
keep going forwards then bend slightly to the right
take the first right
take the third left
come to a cross roads
you will need to go forward towards the roundabout
go forward for about two hundred meters
once you arrived to the traffic light you will see the bank
the police district is on your left
take the fourth junction
if you need a bar just take the third junction
stand on the left when you will see the graveyard
stop at the next traffict light
when you reach the traffic light you can stop

Figure 4

5. YOU WILL PAST THE MENTIONED BUILDING
6. TURN LEFT AT THE THIRD JUNCTION AND A BRIDGE IS ON YOUR RIGHT
7. TAKE THE SECOND ROUNDABOUT YOU MEET
8. GO STRAIGHT ON UNTIL YOU WILL MEET A PUB
9. GO FORWARD AND TAKE THE SECOND JUNCTION
10. GO TOWARDS THE PREVIOUSLY MENTIONED BUILDING
11. CROSS THE ROAD AND YOU WILL FIND THE OFFICE
12. YOU WILL NEED TO TURN LEFT AT THE SECOND JUNCTION
13. TAKE THE ROUNDABOUT BEFORE THE TRAFFIC LIGHT
14. GO DOWN THE ROAD AND TURN LEFT
15. OKAY THE POST OFFICE IS ON YOUR RIGHT
16. CROSS THE BRIDGE AND TAKE THE ROUNDABOUT ON YOUR RIGHT
17. YOU SHOULD TAKE THE SECOND JUNCTION AFTER THE TRAFFIC LIGHT
18. TURN AT THE THIRD JUNCTION ON YOUR RIGHT
19. TAKE THE BRIDGE THEN YOU WILL MEET A TRAFFIC LIGHT
20. TAKE THE PREVIOUSLY MENTIONED JUNCTION
21. GO FORWARD UNTIL YOU MEET A ROUNDABOUT
22. GO STRAIGHT ON AND TURN AT THE THIRD JUNCTION

23. KEEP LEFT AT THE NEXT ROUNDABOUT
24. KEEP THE RIGHT WHEN YOU MEET A TRAFFIC LIGHT
25. TAKE THE FIRST JUNCTION ON YOUR LEFT
26. TURN AROUND THE BUILDING AND YOU WILL SEE A ROUNDABOUT
27. GO OVER THE BRIDGE AND THEN TURN LEFT
28. GO RIGHT WHEN YOU MEET A POST OFFICE
29. YOU WILL FIND YOUR OFFICE ON THE RIGHT
30. AFTER THE TRAFFIC LIGHT TURN LEFT AND GO FORWARD
31. AFTER THE ROUNDABOUT TURN RIGHT
32. WHEN YOU PAST THE POST OFFICE CROSS THE ROAD
33. CROSS THE ROUNDABOUT AND KEEP RIGHT
34. YOU WILL NEED TO KEEP RIGHT AFTER THE BRIDGE
35. YOU WILL FIND A PUB ON YOUR RIGHT
36. YOU WILL FIND A ROUNDABOUT ON YOUR RIGHT
37. YOU WILL FIND A TRAFFIC LIGHT AFTER THE ROUNDABOUT
38. TAKE THE PREVIOUSLY MENTIONED JUNCTION
39. KEEP SLIGHTLY ON YOUR RIGHT WHEN YOU MEET THE TRAFFIC LIGHT
40. THE PUB IS ON YOUR LEFT
41. ONCE ARRIVED TO THE FIRST TRAFFIC LIGHT TURN LEFT AND YOU CAN STOP
42. THE BANK IS ON YOUR RIGHT
43. TAKE THE FOURTH JUNCTION AND
44. GO FORWARD FOR TWO HUNDRED METERS
45. CROSS THE ROUNDABOUT THEN YOU ARRIVED
46. WHEN YOU ARRIVED TO THE TRAFFIC LIGHT STAND ON THE RIGHT
47. STOP WHEN YOU WILL REACH A TRAFFIC LIGHT
48. THE GRAVEYARD IS ON YOUR RIGHT
49. GO TOWARDS THE POST OFFICE YOU WILL SEE A BAR
50. YOU WILL SEE THE BANK ON YOUR LEFT
51. GO STRAIGHT ON FOR TWO HUNDRED METERS THEN STOP AT YOUR RIGHT
52. TURN RIGHT WHEN YOU WILL REACH THE BANK
53. GO TOWARDS THE GRAVEYARD THEN TURN LEFT
54. GO STRAIGHT ON UNTIL YOU REACH A BANK

55. IN TWO HUNDRED METERS TURN RIGHT AND YOU WILL SEE A BAR
56. YOU CAN SEE A BAR AFTER THE GRAVEYARD
57. TAKE THE FOURTH JUNCTION ON YOUR LEFT
58. AFTER TWO TRAFFIC LIGHTS THERE IS A BANK
59. KEEP RIGHT BEFORE THE POLICE DISTRICT
60. A POLICE DISTRICT WILL BE ON YOUR RIGHT AFTER THE BRIDGE
61. YOU WILL FIND A BAR ON YOUR LEFT AFTER THE GRAVEYARD
62. IF YOU NEED A BAR JUST TAKE THE SECOND JUNCTION AND THEN STOP
63. JUST TAKE THE SECOND JUNCTION IF YOU NEED A PUB
64. IF YOU TAKE THE FOURTH JUNCTION A BAR IS ON YOUR LEFT
65. REACH THE TRAFFIC LIGHT THEN TURN RIGHT AND GO STRAIGHT ON
66. GO TOWARDS THE BANK AND THEN TURN LEFT
67. CROSS THE ROUNDABOUT AND YOU WILL FIND A GRAVEYARD
68. YOU WILL REACH A PUB AFTER THE BRIDGE ON YOUR RIGHT
69. YOU CAN STOP WHEN YOU ARRIVED AT THE POLICE DISTRICT
70. ONCE YOU CROSS THE ROAD TURN RIGHT ON THE BRIDGE
71. CROSS THE TRAFFIC LIGHT AND GO STRAIGHT ON
72. YOU SHOULD FIND A BAR ON YOUR RIGHT
73. TESCOS IS ON YOUR LEFT PAST THE POST OFFICE
74. THE PUB IS THE FOURTH BUILDING ON YOUR RIGHT
75. OKAY YOU ARRIVED
76. TURN ABOUT NINETY DEGREES AND THEN REACH THE ROUNDABOUT
77. TAKE THE FOURTH JUNCTION AND STOP AFTER TWO HUNDRED METERS
78. KEEP RIGHT FOR TWO HUNDRED METERS THEN TURN RIGHT
79. IF YOU FIND A BAR STOP AFTER TWO HUNDRED METERS
80. CROSS THE BRIDGE AND KEEP RIGHT
81. KEEP GOING FORWARDS AND AT THE FIRST TRAFFIC LIGHT TURN LEFT
82. KEEP GOING FORWARDS AND YOU WILL SEE THE POLICE DISTRICT
83. YOU NEED TO REACH THE POST OFFICE IT IS ON YOUR RIGHT
84. TAKE THE BRIDGE AND STOP AFTER TWO HUNDRED METERS
85. PAST THE POST OFFICE AND GO FORWARD
86. CROSS THE ROAD AND TURN LEFT

87. AT THE POLICE DISTRICT TURN RIGHT OF NINETY DEGREES
88. ONCE YOU REACH THE TRAFFIC LIGHT TURN LEFT AT THE POST OFFICE
89. THE BAR IS TWO HUNDRED METERS TO THE LEFT
90. STOP AT THE PUB MENTIONED BEFORE
91. TURN ALL AROUND YOU SHOULD FIND A ROUNDABOUT
92. TURN OF NINETY DEGREE AND TAKE THE JUNCTION
93. THE THIRD TO THE LEFT IS YOUR ROAD
94. THIS POLICE STATION SHOULD BE OKAY
95. GO FORWARD UNTIL YOU REACH THE TRAFFIC LIGHT
96. KEEP THE SECOND ROAD AND MEET A BAR
97. CROSS THE ROAD THEN TURN TO THE ROUNDABOUT
98. TO REACH THE ROUNDABOUT GO STRAIGHT
99. OKAY IT IS OVER TWO HUNDRED METERS
100. CROSS THE BRIDGE AND TURN LEFT AT THE FIRST

To estimate the quality of sphinx we introduced 3 quality indices: the percentage of words correctly recognized (R), the percentage of deleted words (D), that are correct words doesn't recognized and the percentage of inserted words (I), that are words recognized but not actually present in the testing set. The results obtained are shown in Fig. 5 and they will be commented in section 4.

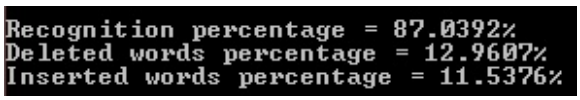


Figure 5

4 Conclusions and future works

Considering the simplicity of the code and the rapidity with which it can be written, the results obtained are quite good. They are not perfect but the reasons can be a lot, starting from the small size of the dictionary caused by the brief corpus we wrote; by producing a better corpus, then a better dictionary and a more complete language model, or by downloading an existing model for the language one intend to use, that should be the better choice, results will be surely better. The only bad thing is that the documentation is poor and in addition to the official site of CMUSphinx [1] there is not much to read on the web. Naturally, in terms of speech recognition, our work is not complete, because it deals only the recognition of words and sentences from an audio source, it could be integrated adding a text recognition for example, to simulate a chat or something like this. This project could be introduced in a bigger one, that implements a spoken dialogue agent, able to understand what the user tell thanks to Sphinx and to dialogue with the agent thanks to all the other Olympus modules.

References

[1] *http://cmusphinx.sourceforge.net/wiki/*