Frequently Asked Questions (FAQ)

- Q: Why my accuracy is poor
- Q: How to do the noise reduction
- Q: Can pocketsphinx reject out-of-grammar words and noises
- Q: Can sphinx4 reject out-of-grammar words and noises
- Q: pocketsphinx_continuous stuck in READY, nothing is recognized
- Q: Which languages are supported
- Q: How to add support for a new language
- Q: I have an issue with CMUSphinx and need help
- Q: What speech feature type does CMUSphinx use and what do they represent
- Q: How to implement "Hot word listening"
- Q: How to evaluate pronunciation
- Q: Pocketsphinx crashes on Windows in _lock_file
- Q: Failed to open audio device(/dev/dsp): No such file or directory
- Q: What is sample rate and how does it affect accuracy
- Q: How can I decode audio encoded with a codec (mp3, mu-law, mp4, g729)
- Q: How to record sound for pocketsphinx from microphone on my platform
- Q: Can I run large vocabulary speech recognition on mobile device / Raspberry PI
- Q: What do CMN values in pocketsphinx output represent
- Q: How can I parse NLU input to perform actions

Q: Why my accuracy is poor

Speech recognition accuracy is not always great. To test speech recognition you need to run recognition on prerecorded reference database to see what happens and optimize parameters.

You do not need to play with unknown values, the **first thing you should do is to collect a database of test samples** and measure the recognition accuracy. You need to dump speech utterances into wav files, write the reference text file and use decoder to decode it. Then calculate WER using the `word_align.pl` tool from Sphinxtrain. Test database size depends on the accuracy but usually it's enough to have 10 minutes of transcribed audio to test recognizer accuracy reliably. The process is described in tutorialtuning.

Q: How to do the noise reduction

There are multiple levels to fight with noise and corruption of the audio. Noise cancellation algorithm modify the audio itself, feature denoising can cleanup features. You can also reduce noise and mismatch in model level, just by adapting the model.

Recent CMUSphinx code has noise cancellation featur. In sphinxbase/pocketsphinx/sphinxtrain it's 'remove_noise' option. In sphinx4 it's Denoise frontend component. So if you are using latest version you should be robust to noise in some degree already. Most modern models are trained with noise cancellation already, if you have your own model you need to retrain it.

The algorithm impelmented is spectral subtraction in on mel filterbank. There are more advanced algorithms for sure, if needed you can extend the current implementation.

It's not recommended to perform an external noise suppression because many noise cancellation algorithms corrupts speech spectrum in unusual ways and reduce speech recognition accuracy even more than the noise itself. For that reason you need to be very careful when selecting the noise cancellation algorithm. Only some of them like Ephraim Malach or Kalman will work properly.

A reasonable way to fight with noise is to adapt the model or train it on a noisy audio. MLLR adaptation usually compensates quite significant part of the noise corruption. It's so-called multistyle training.

Q: Can pocketsphinx reject out-of-grammar words and noises

As for now pocketsphinx does not support confidence scores and out-of-grammar words detection in grammars. We are working on the implementation but it is not ready yet. We support the following modes that can help you to recognize in continuous stream:

Keyword spotting mode You can configure a list of keyphrases to search for and specify the detection threshold for each of them. This mode reliably works in continuous speech stream and can be used for keyword activation. The relevant options of pocketsphinx are -kws and -keyphrase. The methods are ps_set_keyphrase and ps_set_kws.

Large vocabulary decoding mode with a language model. In case of large vocabulary decoding you can retrieve result confidence scores, they should be more or less relable.

If you want to recognize several commands, you can use keyword spotting mode or keyword activation mode combined with the switch to grammar to perform actual operation.

Q: Can sphinx4 reject out-of-grammar words and noises

Sphinx4 has support for skipping out-of-grammar phrases. In default configuration it returns `<unk>` word if something is not matched with the grammar. This feature can not be tuned yet and there is no confidence in grammar mode.

Keyword spotting mode is not implemented in sphinx4.

In large vocabulary decoding mode sphinx4 should return proper confidence for recognition result.

Q: pocketsphinx_continuous stuck in READY, nothing is recognized

If continuous is showing READY and doesn't react to your speech it means that pocketsphinx recording silence. The reasons for that are:

- Pocketsphinx is decoding from a wrong device. Try to check log for the warning `Warning: Could not find Mic element` (try to change device with `-adcdev` option)
- Recording volume is too low (try to increase recording level in volume settings)
- Microphone is broken (test that other programs can record)

Q: Which languages are supported

CMUSphinx itself is language-independent, you can recognize any language. However, it requires an acoustic model and a language model. We provide prebuilt language models for many languages (Enlish, Chinese, French, Spanish, German, Russian, etc) in download section.

Q: How to add support for a new language

The process of building a new language model consists of the following steps:

- Data collection (you can collect audiobooks with text transcriptoin from project like librivox, transcribed podcasts, or setup web data collection. You can also try to contribute to Voxforge. You can start very quickly with just few hours of transcribed data.
- Data cleanup
- Model training
- Testing

Most steps are described in tutorial

Q: I have an issue with CMUSphinx and need help

When you report about problem always provide the following information:

- Version of the software you are using
- Information about your system
- Actions you've made
- Your expectations
- What went wrong

If you want to get fast answer, submit also the following information

- System logs
- Test sample. Try to make test sample as small and as self-contained as possible. It will help **you** to get fast and detailed answer

See How to ask questions howto for more details

Q: What speech feature type does CMUSphinx use and what do they represent

CMUSphinx uses mel-cepstrum MFCC features with noise tracking and spectral subtraction for noise reduction. There are various types of MFCC which differ by number of parameters, but not really different for accuracy (it might be a few percent worse or better).

The interpretation of MFCC (Roughtly introduced Alan V. Oppenheim and Ronald W. Schafer. From Frequency to Quefrency: A History of the Cepstrum. IEEE SIGNAL PROCESSING MAGAZINE) is not applicable as such, and the use of 12 or 13 coefficients seem to be due to historical reasons in many of the reported cases. The choice of the number of MFCCs to include in an ASR system is largely empirical. To understand why any specific number of cepstral coefficients is used, you could do worse than look at very early (pre-HMM) papers. When using DTW using Euclidean or even Mahalanobis distances, it quickly became apparent that the very high cepstral coefficients were not helpful for recognition, and to a lesser extent, neither were the very low ones. The most common solution was to "lifter" the MFCCs - i.e. apply a weighting function to them to emphasise the mid-range coefficients. These liftering functions were "optimised" by a number of researchers, but they almost always ended up being close to zero by the time you got to the 12th coefficient.

In practice, the optimal number of coefficients depends on the quantity of training data, the details of the training algorithm (in particular how well the PDFs can be modelled as the dimensionality of the feature space increases), the number of Gaussian mixtures in the HMMs, the speaker and background noise characteristics, and sometimes the available computing resources.

In semicontinuous models CMUSphinx uses specific packing of derivatives to optimize vector quantization and thus compress model better. Through years various features were used. Mostly they were selected by experiment.

Spectral subtraction of noise is one thing which differs CMUSphinx MFCC from other popular MFCC implementations, it is a simple extension that provides robustness to noise because it tracks and subtracts stable noise component in mel filter energy domain.

Q: How to implement "Hot word listening"

CMUSphinx implements keyphrase spotting mode in pocketsphinx decoder. To recognize a single keyphrase you can run decoder in "keyphrase search" mode.

From command line try:

```
pocketsphinx_continuous -infile file.wav -keyphrase "oh mighty computer" -kws_threshol
```

From the code:

```
ps_set_keyphrase(ps, "keyphrase_search", "oh mighty computer");
ps_set_search(ps, "keyphrase_search");
ps_start_utt();
/* process data */
```

You can also find examples for Python and Android/Java in our sources.

Threshold must be tuned for every keyphrase on a test data to get the right balance missed detections and false alarms. You can try values like 1e-5 to 1e-50.

For the best accuracy it is better to have keyphrase with 3-4 syllables. Too short phrases are easily confused.

You can also search for multiple keyphrase, create a file keyphrase.list like this:

```
oh mighty computer /1e-40/
hello world /1e-30/
other_phrase /other_phrase_threshold/
```

And use it in decoder with -kws configuration option.

```
pocketsphinx_continuous -inmic yes -kws keyphrase_list
```

This feature is not yet implemented in sphinx4 decoder.

Q: How to evaluate pronunciation

Please see pocketsphinx_pronunciation_evaluation.

Q: Pocketsphinx crashes on Windows in _lock_file

The stack trace is usually the following:

```
INFO: fe_interface.c(289): You are using internal mechanism to generate the seed.
INFO: feat.c(289): Initializing feature stream to type: '1s_c_d_dd', ceplen=13,CMN='cu
INFO: cmn.c(142): mean[0]= 12.00, mean[1..12]= 0.0
INFO: acmod.c(153): Reading linear feature trasformation from acoustic/feature.transfo
INFO: mdef.c(520): Reading model definition: acoustic/mdef
INFO: bin_mdef.c(173): Allocation 104810 * 8 bytes (818 KiB) for CD tree
INFO: tmat.c(205): Reading HMM transition probability matrices: acoustic/transition_ma

Stack trace:
ntdll.dll!774f8db9()
        [Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll
        ntdll.dll!774f8cc8()
>       msvcr100.dll!_lock_file(_iobuf * pf)  Line 236 + 0xa bytes      C
        msvcr100.dll!fgets(char * string, int count, _iobuf * str)  Line 71 + 0x6 byte
        sphinxbase.dll!002319ef()
        msvcr100.dll!_unlock(int locknum)  Line 375     C
        msvcr100.dll!_unlock_file(_iobuf * pf)  Line 313 + 0xe bytes    C
        msvcr100.dll!fread_s(void * buffer, unsigned int bufferSize, unsigned int elem
        msvcr100.dll!fread(void * buffer, unsigned int elementSize, unsigned int count
        sphinxbase.dll!00231743()
        sphinxbase.dll!00231cbf()
```

sphinxbase was compiled iwth MultiThreadedDLL runtime, see in vcxproj

```
<RuntimeLibrary>MultiThreadedDLL</RuntimeLibrary>
```

If you don't compile your project with similar setting it will crash. Use proper runtime library or recompile sphinxbase

Q: Failed to open audio device(/dev/dsp): No such file or directory

Device file /dev/dsp is missing because OSS support is not enabled in the kernel. You can either compile pocketsphinx with ALSA support by installing alsa development headers from a package libasound2 or alsa-devel and recompiling or you can install oss-compat package to enable OSS support.

The installation process is not an issue if you understand the complexity of audio subsystems in Linux. The audio subsystem is complex unfortunately, but once you get it things will be easier. Historically, audio subsystem is pretty fragmented. It includes the following major frameworks:

- Old Unix-like DSP framework – everything is handled by the kernel-space driver. Applications interact with /dev/dsp device to produce and record audio
- ALSA – newer audio subsystem, partially in kernel but also has userspace library libasound. ALSA also provides DSP compatibliity layer through snd_pcm_oss driver which creates /dev/dsp device and emulates audio
- Pulseaudio – even newer system which works on the top of libasound ALSA library but provides a sound server to centralize all the processing. To communicate with the library it also provides libpulse library which must be used by applications to record sound
- Jack – another sound server, also works on the top of ALSA, provides anoher library libjack. Similar to Pulseaudio there are others not very popular frameworks, but sphinxbase doesn't support them. Example are ESD (old GNOME sound server), ARTS (old KDE sound server), Portaudio (portable library usable across Windows, Linux and Mac).

The recommended audio framework on Ubuntu is pulseaudio.

Sphinxbase and pocketsphinx support all the frameworks and automatically selects the one you need in compile time. The highest priority is in pulseaudio framework. Before you install sphinxbase you need to decide which framework to use. You need to setup the development part of the corresponding framework after that.

For example, it's recommended to install libpulse-dev package to provide access to pulseaudio and after that sphinxbase will automatically work with Pulseaudio. Once you work with pulseaudio you do not need other frameworks. On embedded device try to configure alsa.

Q: What is sample rate and how does it affect accuracy

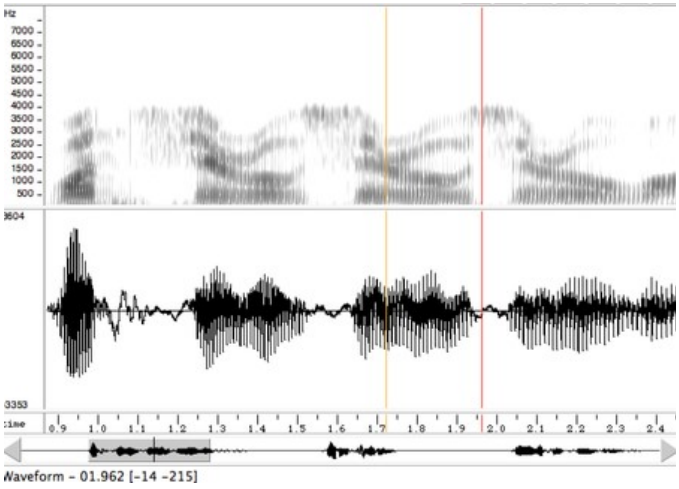
Unfortunately we don't provide universal models for different bandwidths (8khz models are 10% worse in accuracy) and we can not detect sample rate yet. So before using decoder you need to make sure that both sample rate of the decoder matches the sample rate of the input audio and the bandwidth of the audio matches the bandwidth that was used to train the model. A mismatch results in very bad accuracy.

First of all you need to understand the difference between sample rate and frequency bandwidth. Sample rate - the rate of samples in the recording. Sample rate 16000 means that there are 16000 samples collected every second. You can resample audio with sox or with ffmpeg:

```
sox file.wav -r 16000 file-16000.wav
ffmpeg file.mp3 -ar 16000 file-16000.wav
```

Then there is a bandwidth - the range of frequencies included in the audio, it doesn't change with sample rate. If audio had frequencies up to 8khz only no matter how you resample the bandwidth will be still the same. And due to mismatched bandwidth the audio will not be recognized properly.

To check the bandwidth of the audio you need to see it's spectrum in audio editor like Wavesurfer. You'll see dark spectrum only up to 4khz if audio is 8khz. If your audio was recorded from telephone source or was compressed by voip codec most likely it has only 8khz bandwidth.



So first you need to do the following: make sure that frontend sample rate matches the sample rate of the audio. And then make sure that bandwidth of the audio matches your model bandwidth. Use en-us generic model for 16khz bandwidth and en-us-8khz generic model for 8khz bandwidth.

Q: How can I decode audio encoded with a codec (mp3, mu-law, mp4, g729)

CMUSphinx decoders do not include format converters, they have no idea how to decode encoded audio. Before processing audio must be converted to PCM format. Recommended format is `16khz 16bit little-endian mono`. If you are decoding telephone quality audio you can also decode `8khz 16bit little-endian mono`, but you usually need to reconfigure the decoder to input 8khz audio. For example, pocketsphinx has `-samprate 8000` option in configuration.

You can find out file format using `soxi` command or `file` command.

You can convert your encoded audio into required format with ffmpeg or with other specialized decoder:

```
ffmpeg -i file.mp3 -ar 16000 -ac 1 file.wav
```

If you are writing software you might want to integrate the format converter like ffmpeg/avconv as a separate library.

Q: How to record sound for pocketsphinx from microphone on my platform

Sound recording is pretty unique in various software platforms. For example, Android uses Java Audio Recorder. Python bindings use PyAudio. On Windows we record with WinMM API, on Windows Phones there is another API. All those APIs are not compatible with each other.

For that reason there is no standard audio recording framework in pocketsphinx/sphinxbase. You have to record audio with the platform API and just pass the data into decoder. For example, if you want to record audio in Android NDK, you can use OpenSL ES library or any other framework.

There is also libad library which works on Linux/Windows/Mac, but it is very limited and we are not going to extend it into other platforms.

Q: Can I run large vocabulary speech recognition on mobile device / Raspberry PI

No, you can't. The CPU is too slow for large vocabulary speech recognition. Phone CPU is usually 9 times slower than desktop. Speech recognition with unlimited vocabulary requires very big computational and memory resources (terabytes of memory) and thus it's very hard to do that in iPhone on other embedded device. iPad is easier.

Google has put very big effort to make their engine work offline for dictation, and still it prefers to send data to the server because it is significantly more accurate.

Because of that most of the solutions running on small devices use limited vocabulary. Though this vocabulary can be large enough so you will not notice that. Usually 500-1000 words is enough to cover most practical situations.

To train a language model you need texts from your domain (words and expressions). Language model training is described in tutorial.

Q: What do CMN values in pocketsphinx output represent

```
INFO: cmn_live.c(120): Update from < 40.00 3.00 -1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.
INFO: cmn_live.c(138): Update to < 18.15 -10.38 4.88 -7.93 3.09 -0.89 0.93 -1.01 6.29
```

To model the speech more accurately CMUSphinx applies volume normalization. Basically audio level is reduced to a certain standard value with the certain scale. This process is called 'cepstral mean normalization'. Note that not just the whole level is adjusted, we also adjust the level in the individual frequency bands. It helps to deal with so-called channel distortion when level is raised for high frequencies by means of some filter.

The normalization is performed during decoding in live mode and starts from initial value. If the initial value is not well fit, the accuracy for first utterance will be lower than expected. For the second utterance the value should reflect proper signal level.

During training the normalization is performed in batch mode, means the whole utterance is normalized at once. This is better approach for longer utterances, but for short utterances it might fail to properly estimate the signal level.

If signal level changes quickly it is hard for decoder to adapt, you have to wait few seconds. For example if you move away from the microphone it might affect the decoder accuracy. In the future we also might want to refactor the input pipeline to estimate CMN more reliably for example, to wait few first seconds until CMN is properly estimated instead of relying on the initial value.

Unusual CMN values help to identify the problems with the speech like input data byte order problems. In this particular example value of C0 18.5 is pretty low it means that recorded sound is very quiet. Proper signal level should be values are around 40.0

Q: How can I parse NLU input to perform actions

There are number of solutions here depending on the complexity of the input and the variety of cases.

Simple commands could be parsed by comparing the result as a string to a fixed template. Something simple like this:

```
if hyp == 'up':
   do_go_up()
elif hyp == 'down':
   do_go_down
```

For more advanced command parsing you can use regular expressions to extract entities and custom code to parse more complex things like numbers. See an example on Stackoverflow.

There are more advanced NLU libraries for intent parsing which can be trained from sample, for example check RASA NLU. Many such libraries are based on Spacy toolkit.

In the end you can train an deep learning neural network to map input directly to action, see for example DialogStateTracking. Deep neural network can also provide you more natural behavior and create you natural language outputs.

Issue Tracker

Telegram

Feeds

Posts