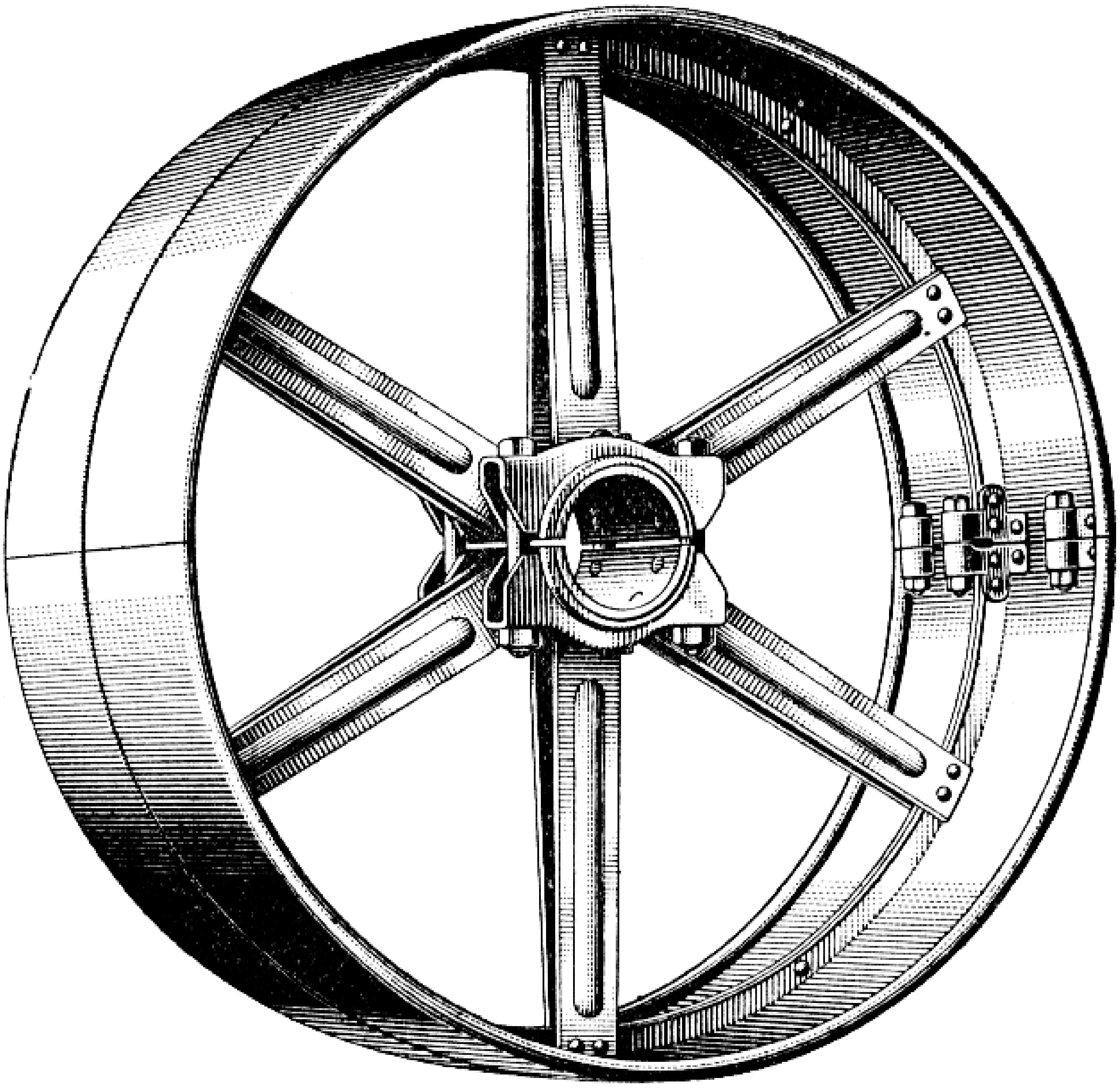

FIG. 16

RapidEars 2.03 Manual

Written by Halle Winkler, published by Politepix

Friday, January 9, 2015

Introduction, Installation and Support

Introduction

RapidEars is a plugin for OpenEars which adds the new ability to do real time speech recognition on in-progress speech for live voice recognition on the iPhone, iPod and iPad, in English or in Spanish. If your application has a need for speed and you are shipping for devices that can support more CPU overhead, all you have to do is install the RapidEars plugin in your working OpenEars project and PocketsphinxController and OEEventsObserver will get some new methods that will let you stop waiting for pauses in speech and start evaluating immediately.

It's great for games or any application that needs real time speech recognition and fast feedback. Because of the optimizations that are needed to do instant recognition on the iPhone, its live recognition does not quite have the accuracy rates of OEPocketsphinxController in the stock version of OpenEars (its final recognition is the same, however) and should be used with smaller vocabularies.

RapidEars can be purchased at the Politepix shop here and it's very important to thoroughly evaluate the demo version before purchasing, which can be downloaded from this shop page. The installation and usage process is the same for both the demo and licensed version, but the demo times out after 3 minutes of use and can't be submitted to the App Store.

The best way to get started using RapidEars is to get a tutorial from thePolitepix interactive tutorial tool. Steps for getting started and more in-depth documentation are also provided on this page.

Installation

How to install and use RapidEars:

RapidEars is a plugin for OpenEars, so it is added to an already-working OpenEars project in order to enable new OpenEars features. In these instructions we are using the OpenEars sample app as an example for adding the plugin and new features, but the steps are basically the same for any app that already has a working OpenEars installation. Please note that RapidEars requires OpenEars 2.0 or greater.

1. Download and try out the OpenEars distribution and try the OpenEars sample app out. RapidEars is a plug-in for OpenEars that is added to an OpenEars project so you first need a known-working OpenEars app to work with. The OpenEars sample app is fine for this to get started. You can also get a complete tutorial on both creating an OpenEars app and adding RapidEars to it using the automatic customized tutorial.
2. Open up the OpenEars Sample App in Xcode. Drag your downloaded RapidEarsDemo.framework into the OpenEars sample app project file navigator.
3. Open up the Build Settings tab of your app or OpenEarsSampleApp and find the entry "Other Linker Flags" and add the linker flag "-ObjC". Do this for debug and release builds. More explanation of this step can be seen in the tutorial by selecting the live recognition tutorial, which will also show exactly how to use the new methods added by RapidEars.

Support

You can have one free email support incident with the demo version of RapidEars, and as many questions on the OpenEars plugins forums as you like. To use your free email support incident, you must register your app and a verifiable name (company name or personal name) here.

You can also send as many sales inquiries as you like through the contact form, and you don't need to register in order to do so, although a sales inquiry with a tech support question will be considered a support incident and we'll ask you to register in order to have it engaged.

Once you have completed licensing of the framework for your app, you get more email support incidents if you have purchased an email support bundle, and continued forum support. Extra email support incidents for demo and licensed versions can always be purchased at the Politepix shop. Support contracts for multiple email support incidents with Politepix can also be purchased. Licensing the framework requires giving the exact application name that the framework will be linked to, so don't purchase the license until you know the app name. Please read on for the RapidEars documentation.

OEEventsObserver+RapidEars Category Reference

Detailed Description

This plugin returns the results of your speech recognition by adding some new callbacks to the OEEventsObserver.

Usage examples

> *What to add to your implementation:*

At the top of your header after the line

```
#import <OpenEars/OEEventsObserver.h>
```

Add the line

```
#import <RapidEarsDemo/OEEventsObserver+RapidEars.h>
```

And after this OEEventsObserver delegate method you added to your implementation when setting up your OpenEars app:

```
- (void) testRecognitionCompleted {
 NSLog(@"A test file that was submitted for recognition is now complete.");
}
```

Just add the following extended delegate methods:

```
- (void) rapidEarsDidReceiveLiveSpeechHypothesis:(NSString *)hypothesis
recognitionScore:(NSString *)recognitionScore {
    NSLog(@"rapidEarsDidReceiveLiveSpeechHypothesis: %@",hypothesis);
}

- (void) rapidEarsDidReceiveFinishedSpeechHypothesis:(NSString *)hypothesis
recognitionScore:(NSString *)recognitionScore {
    NSLog(@"rapidEarsDidReceiveFinishedSpeechHypothesis: %@",hypothesis);
}
```

Warning

It is a requirement that any OEEventsObserver you use in a view controller or other object is a property of that object, or it won't work.

Method Documentation

```
- (void) rapidEarsDidReceiveLiveSpeechHypothesis: (NSString *) hypothesis
                               recognitionScore: (NSString *) recognitionScore
```

The engine has detected in-progress speech. This is the simple delegate method that should be used in most cases, which just returns the hypothesis string and its score.

```
- (void) rapidEarsDidReceiveFinishedSpeechHypothesis: (NSString *) hypothesis
                                   recognitionScore: (NSString *) recognitionScore
```

A final speech hypothesis was detected after the user paused. This is the simple delegate method that should be used in most cases, which just returns the hypothesis string and its score.

```
- (void) rapidEarsDidDetectLiveSpeechAsWordArray: (NSArray *) words
                                  andScoreArray: (NSArray *) scores
```

The engine has detected in-progress speech. Words and respective scores are delivered in separate arrays with corresponding indexes.

```
- (void) rapidEarsDidDetectFinishedSpeechAsWordArray: (NSArray *) words
                                      andScoreArray: (NSArray *) scores
```

A final speech hypothesis was detected after the user paused. Words and respective scores are delivered in separate arrays with corresponding indexes.

```
- (void) rapidEarsDidDetectLiveSpeechAsWordArray: (NSArray *) words
                                     scoreArray: (NSArray *) scores
                                 startTimeArray: (NSArray *) startTimes
                                   endTimeArray: (NSArray *) endTimes
```

The engine has detected in-progress speech. Words and respective scores and timing are delivered in separate arrays with corresponding indexes.

```
- (void) rapidEarsDidDetectFinishedSpeechAsWordArray: (NSArray *) words
                                         scoreArray: (NSArray *) scores
                                     startTimeArray: (NSArray *) startTimes
                                       endTimeArray: (NSArray *) endTimes
```

A final speech hypothesis was detected after the user paused. Words and respective scores and timing are delivered in separate arrays with corresponding indexes.

```
- (void) rapidEarsDidDetectLiveSpeechAsNBestArray: (NSArray *) words
                                   andScoreArray: (NSArray *) scores
```

The engine has detected in-progress speech. N-Best words and respective scores are delivered in separate arrays with corresponding indexes.

```
- (void) rapidEarsDidDetectFinishedSpeechAsNBestArray: (NSArray *) words
                                        andScoreArray: (NSArray *) scores
```

A final speech hypothesis was detected after the user paused. N-Best words and respective scores are delivered in separate arrays with corresponding indexes.

OEPocketsphinxController+RapidEars Category Reference

Detailed Description

A plugin which adds the ability to do live speech recognition to OEPocketsphinxController.

Usage examples

> *Preparing to use the class:*

Like OEPocketsphinxController which it extends, we need a language model created with OELanguageModelGenerator before using OEPocketsphinxController+RapidEars. We have already completed that step above.

> *What to add to your implementation:*

Add the following to your implementation (the .m file): Under the @implementation keyword at the top, after the line #import <OpenEars/OEPocketsphinxController.h>:

```
#import <RapidEarsDemo/OEPocketsphinxController+RapidEars.h>
```

Next, comment out all calls in your app to the method

```
startListeningWithLanguageModelAtPath:dictionaryAtPath:languageModelIsJSGF:
```

and in the same part of your app where you were formerly using this method, place the following:

```
[[OEPocketsphinxController sharedInstance]
startRealtimeListeningWithLanguageModelAtPath:lmPath dictionaryAtPath:dicPath
acousticModelAtPath:[OEAcousticModel pathToModel:@"AcousticModelEnglish"]];
// Starts the rapid recognition loop. Change "AcousticModelEnglish" to
"AcousticModelSpanish" in order to perform Spanish language recognition.
```

If you find that sometimes you are getting live recognition and other times not, make sure that you have definitely replaced all instances of startListeningWithLanguageModelAtPath: with startRealtimeListeningWithLanguageModelAtPath:.

Warning

Please read OpenEars' OEPocketsphinxController for information about instantiating this object.

Method Documentation

```
- (void) startRealtimeListeningWithLanguageModelAtPath: (NSString *) languageModelPath
                                        dictionaryAtPath: (NSString *) dictionaryPath
                                     acousticModelAtPath: (NSString *) acousticModelPath
```

Start the listening loop. You will call this instead of the old OEPocketsphinxController method

```
- (void) setRapidEarsToVerbose: (BOOL) verbose
```

Turn logging on or off.

```
- (void) setFinalizeHypothesis: (BOOL) finalizeHypothesis
```

You can decide not to have the final hypothesis delivered if you are only interested in live hypotheses. This will save some CPU work.

```
- (void) setReturnDuplicatePartials: (BOOL) duplicatePartials
```

Setting this to true will cause you to receive partial hypotheses even when they match the last one you received. This defaults to FALSE, so if you only want to receive new hypotheses you don't need to use this.

```
- (void) setRapidEarsReturnNBest: (BOOL) nBest
```

EXPERIMENTAL. This will give you N-Best results, most likely at the expense of performance. This can't be used with setReturnSegments: or setReturnSegmentTimes: . This is a wholly experimental feature and can't be used with overly-large language models or on very slow devices. It is the sole responsibility of the developer to test whether performance is acceptable with this feature and to reduce language model size and latencyTuning in order to get a good UX â€“ there is absolutely no guarantee given that this feature will not result in searches which are too slow to return if it has too much to do. If this experimental feature results in an excess of support requests due to too many not reading this warning and consequently reporting realtime n-best slowness of arbitrary models as a bug, it could end up being removed in a future version.

```
- (void) setRapidEarsNBestNumber: (NSUInteger) rapidEarsNBestNumber
```

This is the maximum number of nbest results you want to receive. You may receive fewer than this number but will not receive more. This defaults to 3 for RapidEars; larger numbers are likely to use more CPU and smaller numbers less. Settings below 2 are invalid and will be set to 2. It is not recommended to ever set this above 3 for realtime processing.

```
- (void) setReturnSegments: (BOOL) returnSegments
```

Setting this to true will cause you to receive your hypotheses as separate words rather than a single NSString. This is a requirement for using OEEventsObserver delegate methods that contain timing or per-word scoring. This can't be used with N-best.

```
- (void) setReturnSegmentTimes: (BOOL) returnSegmentTimes
```

Setting this to true will cause you to receive segment hypotheses with timing attached. This is a requirement for using OEEventsObserver delegate methods that contain word timing information. It only works if you have setReturnSegments set to TRUE. This can't be used with N-best.

```
- (void) setLatencyTuning: (NSInteger) latencyTuning
```

This can take a value between 1 and 4. 4 means the lowest-latency for partial hypotheses and 1 means the highest. The lower the latency, the higher the CPU overhead, and vice versa. This defaults to 4 (the lowest latency and highest CPU overhead) so you will reduce it if you have a need for less CPU overhead until you find the ideal balance between CPU overhead and speed of hypothesis.