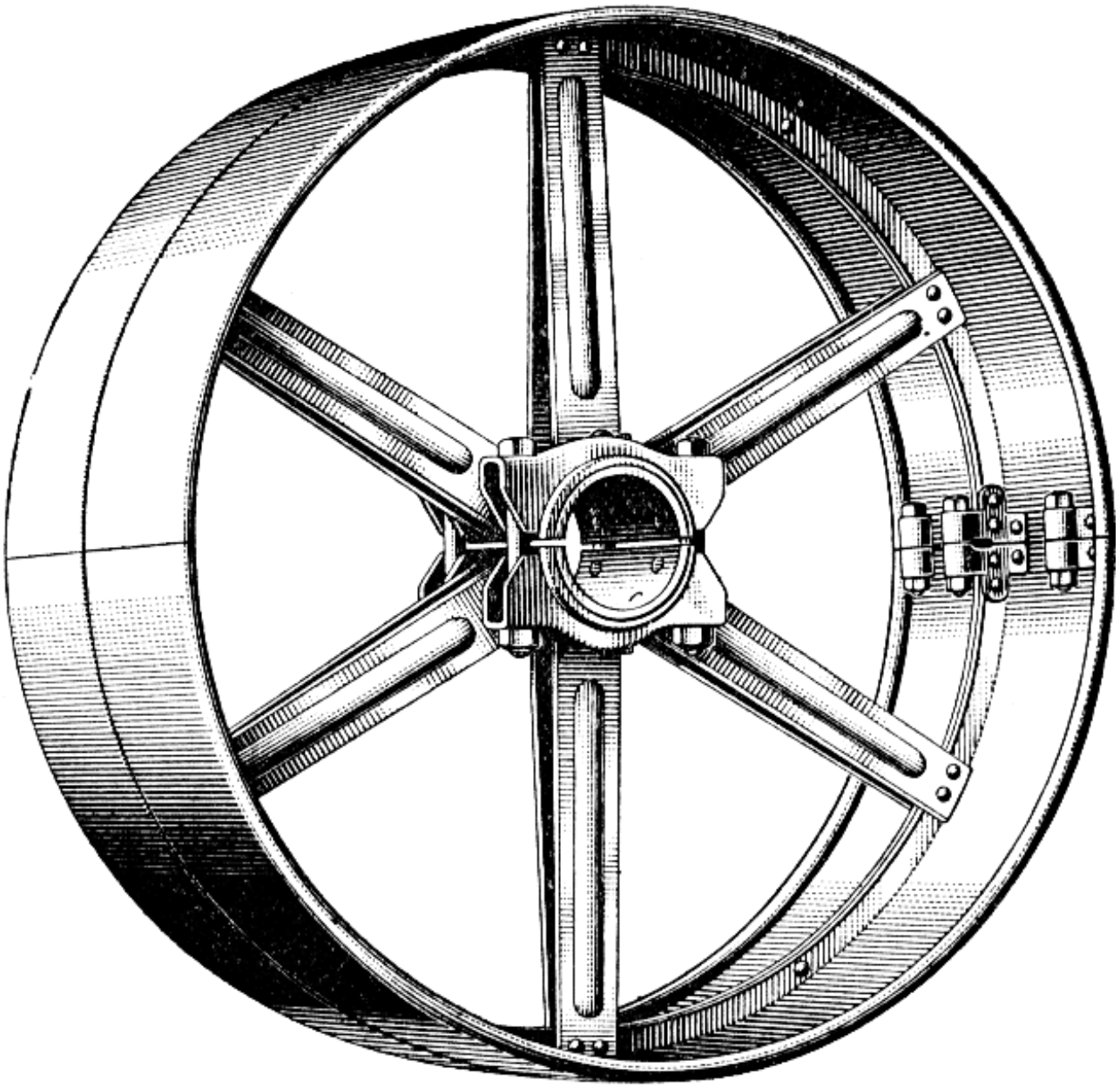

FIG. 16

RapidEars 1.1 Manual

Written by Halle Winkler, published by Politepix

Thursday, September 20, 2012

Table of Contents

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. If your application has a need for speed and you are shipping for devices that can support more CPU overhead (we recommend that this feature is restricted to the iPhone 4 and higher), all you have to do is install the RapidEars plugin in your working OpenEars project and PocketsphinxController and OpenEarsEventsObserver 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, it does not have the accuracy rates of PocketsphinxController in the stock version of OpenEars and should be used with smaller vocabularies.

RapidEars can be purchased at the Politepix shop here and we recommend thoroughly evaluating 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 the Politepix 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 1.2.1 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 two more email support incidents 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.

OpenEarsEventsObserver+RapidEars Category Reference

Detailed Description

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

Usage examples

> *What to add to your implementation:*

At the top of your implementation after the line

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

Add the line

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

And after this OpenEarsEventsObserver delegate method you added when setting up your OpenEars app:

```
- (void) pocketSphinxContinuousSetupDidFail {

}
```

Just add the following extended delegate methods:

```
- (void) rapidEarsDidDetectLiveSpeechAsWordArray:(NSArray *)words
andScoreArray:(NSArray *)scores {
    NSLog(@"detected words: %@",[words componentsJoinedByString:@" "]);
    // NSLog(@"detected scores: %@",scores);
}

- (void) rapidEarsDidDetectFinishedSpeechAsWordArray:(NSArray *)words
andScoreArray:(NSArray *)scores {
    NSString *hypothesis = [words componentsJoinedByString:@" "];
```

```
    NSLog(@"detected complete statement: %@",hypothesis);
    // NSLog(@"detected scores: %@",scores);
}

- (void) rapidEarsDidDetectBeginningOfSpeech {
    NSLog(@"rapidEarsDidDetectBeginningOfSpeech");
}

- (void) rapidEarsDidDetectEndOfSpeech {
    NSLog(@"rapidEarsDidDetectEndOfSpeech");
}
```

Method Documentation

```
- (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) rapidEarsDidDetectBeginningOfSpeech
```

Speech has started. This is primarily intended as a UI state signal.

```
- (void) rapidEarsDidDetectEndOfSpeech
```

Speech has ended. This is primarily intended as a UI state signal.

PocketsphinxController+RapidEars Category Reference

Detailed Description

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

Usage examples

> *Preparing to use the class:*

Like PocketsphinxController which it extends, we need a language model created with LanguageModelGenerator before using PocketsphinxController+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/PocketsphinxController.h>:

```
#import <RapidEarsDemo/PocketsphinxController+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:

```
[self.pocketsphinxController setRapidEarsToVerbose:FALSE]; // This defaults to FALSE but will give a lot of debug readout if set TRUE
[self.pocketsphinxController setRapidEarsAccuracy:1]; // This defaults to 20, maximum accuracy, but can be set as low as 1 to save CPU
[self.pocketsphinxController setFinalizeHypothesis:TRUE]; // This defaults to TRUE and will return a final hypothesis, but can be turned off to save a little CPU and will then return no final hypothesis; only
```

```
partial "live" hypotheses.
[self.pocketsphinxController setFasterPartials:TRUE]; // This will give
faster rapid recognition with less accuracy. This is what you want in
most cases since more accuracy for partial hypotheses will have a
delay.
[self.pocketsphinxController setFasterFinals:FALSE]; // This will give
an accurate final recognition. You can have earlier final recognitions
with less accuracy as well by setting this to TRUE.
[self.pocketsphinxController
startRealtimeListeningWithLanguageModelAtPath:lmPath
andDictionaryAtPath:dicPath]; // Starts the rapid recognition loop.
```

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

There can only be one PocketsphinxController+RapidEars instance in your app.

Method Documentation

```
- (void) startRealtimeListeningWithLanguageModelAtPath: (NSString *) languageModelPath
                                  andDictionaryAtPath: (NSString *) dictionaryPath
```

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

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

Turn logging on or off.

```
- (void) setRapidEarsAccuracy: (int) accuracy
```

Scale from 1-20 where 1 is the least accurate and 20 is the most. This has an linear relationship with the CPU overhead. The best accuracy will still be less than that of Pocketsphinx in the stock OpenEars package and this only has a notable effect in cases where setFasterPartials is set to FALSE. Defaults to 20.

```
- (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) setFasterPartials: (BOOL) fasterPartials
```

This will give you faster partial hypotheses at the expense of accuracy

```
- (void) setFasterFinals: (BOOL) fasterPartials
```

This will give you faster final hypotheses at the expense of accuracy. Setting this causes setFasterPartials to also be set.