Pitch Detector User Guide

document version 1.0

Introduction

This pitch detector algorithm is based on C implementation of RAPT (i.e. Robust Algorithm for Pitch Tracking) distributed as part of the ESPS toolkit downloaded from http://www.speech.kth.se/speech/esps/esps.zip (there are other copies available in GitHub). The algorithm is ported to C#.

RAPT Pitch Tracking algorithm is described in paper *A Robust Algorithm for Pitch Tracking by David Talkin 1995.* A *s*canned copy of the paper is available in https://www.ee.columbia.edu/~dpwe/papers/Talkin95-rapt.pdf.

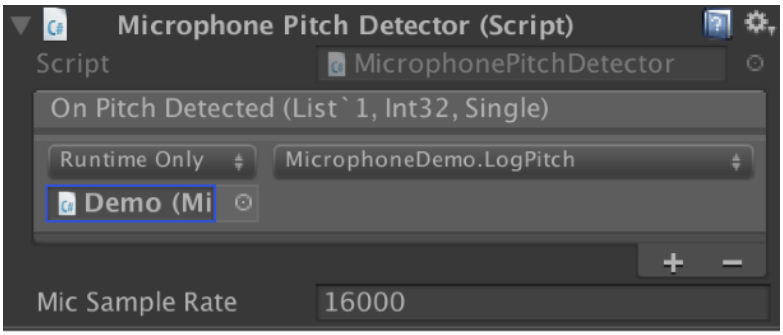
Getting Started

1. Drag `MicrophonePitchDetector` prefab into your scene.
2. Write a callback method that uses the pitch values. The method should have thee arguments:
   - a list of pitch values in Hz (`List<float>`),
   - the number of audio samples from which the pitch values were analyzed (`int`), and
   - the volume in full scale decibels (`float`).

   For example:

```
// Print pitch values to console
public void LogPitch (List<float> pitchList, int samples, float db) {
	var midis = RAPTPitchDetectorExtensions.HerzToMidi (pitchList);
	Debug.Log ("detected " + pitchList.Count + " values from " + samples
		+ " samples, db:" + db);
	Debug.Log (midis.NoteString ());
}
```

3. Configure `MicrophonePitchDetector.onPitchDetected` in editor:



OR

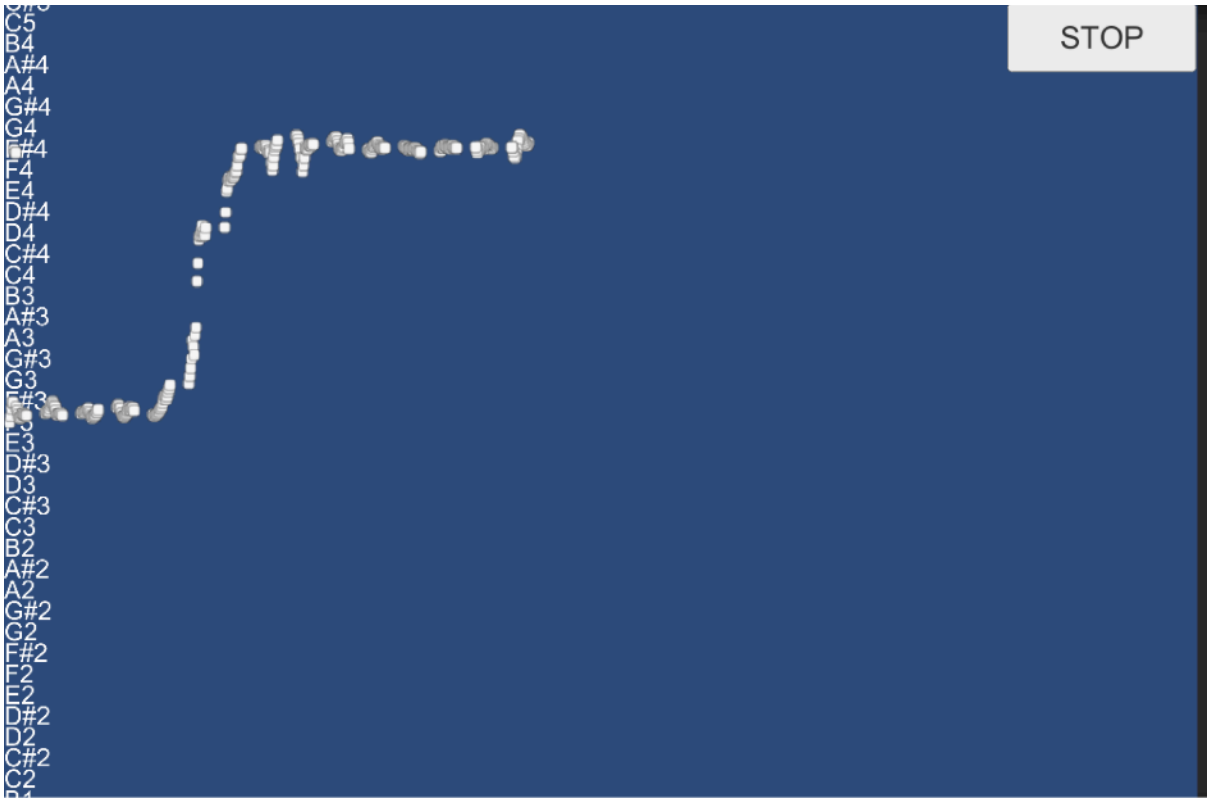
Add callback method via code. For example:

```
public MicrophonePitchDetector detector;
…
detector.onPitchDetected.AddListener (LogPitch);
```

4. Set `MicrophonePitchDetector.Record` to `true` for starting microphone input and receiving `onPitchDetected` events.

Example Scene

Pitch Detector asset contains `ExampleScene`. The scene has a record button at the top-right corner. The button is used for starting and stopping the default microphone of the device. When the microphone is on, the callback `MicrophoneDemo.DrawPitch` gets the pitch values and draws them on the screen.



Parameters

`MicrophonePitchDetector.micSampleRate:` Microphone sample rate. default: 16000
`MicrophonePitchDetector.interval:` Interval (in sec) at which the microphone input is sent to the pitch detector algorithm. The interval should not be too short since the algorithm calculates pitch in Params.frame_step frames (see below). default: 0.07

`Params.frame_step:` Size (in sec) of one analysis frame. default: 0.015
`Params.minF0:` Minimum pitch (in Hz) to search for. default: 50
`Params.maxF0:` Maximum pitch (in Hz) to search for. default: 800

The algorithm has many other parameters that are defined in `Params` class. Asset author has not conducted extensive research on parameters and most parameter values are derived straight from the C implementation.

Results

Analyzed pitch is received via event that has three arguments:

```
public class PitchEvent : UnityEvent<List<float>, int, float> {}
```

The first argument is a list of pitch values in Hz. If the list is empty, the algorithm failed to find suitable pitch value. The pitch values in the list are ordered by time from the oldest to the newest.
The second argument is the number of samples that detection was based on. (Note that this is the number of audio samples that were given as input to the algorithm, not the number of pitch values that were given as output of the algorithm.)
The third argument is full scale decibels where value 0 denotes maximum voice level. The voice level is calculated from the same samples as the pitch values.

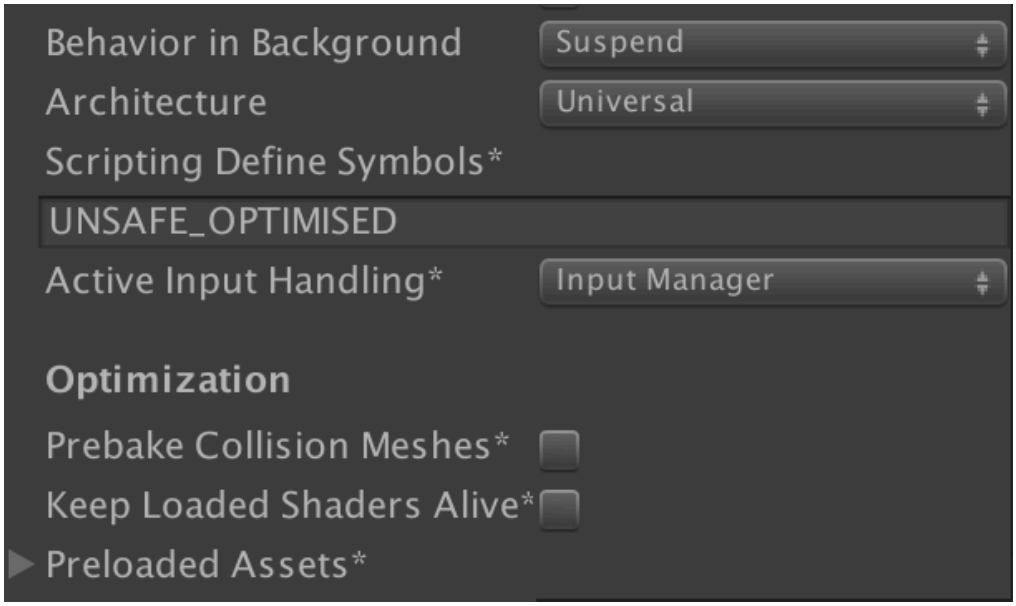
Depending on the use case, the asset user might apply additional smoothing to the results. The third argument (i.e. the voice level) can be used to filter out low voices.

Performance Considerations

The algorithm works in time domain and the cost of computation grows with sample rate. Thus, reducing the sample rate of the input signal is the best way to reduce the CPU load of the pitch detector.

Additional speedup can be achieved by using pointers in array operations. In order to enable pointer operations:

1. define `UNSAFE_OPTMISED`. You can do this though Scripting Define Symbols text box in Player settings:



2. Add '`-unsafe`' switch into mcs.rsp file. The mcs.rsp file is located in project Assets folder. If the folder does not contain mcs.rsp file, you can create an empty text file named mcs.rsp and add '`-unsafe`' text there.

If the Pitch Detector algorithm blocks the Unity main thread for too long time in your use case, you can use the algorithm in background thread (e.g. `System.Threading.Thread` or `BackgroundWorker`). In that case, every thread should have its own instance of `RAPTPitchDetector` object.

Programming

This pitch detector algorithm is based on code that is ported from C to C#. The C implementation was downloaded from http://www.speech.kth.se/speech/esps/esps.zip. The important C source files are `get_f0.c`, `dp_f0.c`, `get_cands.c` and `sigproc.c`.

C# implementation is mainly in file `RAPTPitchDetector.cs`. Best effort has been made in refactoring the C implementation to more readable form but the code of `RAPTPitchDetector.cs` is still pretty complex and C heritage is visible.

If source code modifications are needed, unit tests can be utilised for checking if something has gone wrong in modifications. The pitch detector asset contains unit tests in `EditorScripts/PitchTrackerTest.cs`. Before testing, the file `PitchTrackerTest.cs` must be copied to `Editor` folder of the Unity project. After that, the unit tests can be run via menu **Window/Test Runner**. The most important test cases are

- **PitchTrackerTestFrequnecySweep** that tests algorithm with 50-700Hz sine wave,
- **PitchTrackerTestClips** that analyses three audio clips in TestData folder,
- **PitchTrackerTestStreaming** that does same as PitchTrackerTestClips but in smaller chunks.

Audio clips for the unit tests are loaded from path:

```
Path.Combine (Application.dataPath, "HumanVoicePitchDetector/TestData/")
```