# Idi's user manual

Wagner Frédéric

November 28, 2010

## 1 Introduction

Welcome to Idi (you should pronounce "eedee"). This manual is intended as a guide for the end user. Idi is a speech to text system designed to help people with physical disabilities to type and keep using their computer. As a bonus feature it can also be used as a remote control for your computer and therefore will allow you to keep typing while in your bath.

Idi works in the following way : the user is recording a set of samples for each word he wants to have recognized by Idi. For each sample, the set needs eventually to be refined into smaller subsets which share common characteristics. Idi then builds a model file which will allow him to compare incoming sounds to models of all recognized words. Eventually the incoming sample will be matched with the corresponding model and the corresponding key sequence typed.

Be warned on the following :

- The typing speed with Idi is not very high. You might get around 50 keys per minute.
- Idi will only recognize words you teach him.
- Changing your microphone settings might require you to painfully record all your samples again.
- Idi is still in development.

## 2 Installation

Installing Idi is pretty straightforward.

1. Ensure all required libraries are installed with the corresponding header files :
    - aosd
    - sndfile

- portaudio

2. Type "make"
3. That's it. No "make install" is provided. Update your path.
4. If you intend to use helper scripts you also need perl and gnuplot.

## 3 Configuration

Configuring idi is done in several steps.

### 3.1 Adjust mixer settings

Try recording a sound sample with your mixer and display it in a sample editor (audacity for example). Check that your signal is neither too weak nor too strong as shown in Figure **??**. In case of troubles you need to adjust your mixer settings or the position of your microphone.

Once ok you need to ensure that your mixer settings will always be the same when using Idi or when recording new samples. *Be warned!* Should these settings change, you might need to start the training step from scratch. In order to avoid catastrophies Idi will try to reload mixer settings when launched. This feature only works when using alsa. To achieve this type the following commands :

```
mkdir ~/.idi
alsactl -f ~/.idi/asound.state store
```

### 3.2 Adjusting cut parameters

The next step is to adjust the cut parameters. Idi will try to split incoming signal into separate words by detecting activity. Activity means that the signal is detected above a threshold for a time long enough while silence means the signal being below that threshold for another duration. A default set of parameters are supplied but it might be best to personalize them for your own sound card / noisy environment. Especially, if you intend to use Idi in a noisy place, you need to raise thresholds.

You can create a configuration file by launching the following command :

```
silence_level
```

The next step is then to test these parameters and eventually to modify them.

For that we should first record a test sample. We create a *seed* directory which will contain the initial set of samples. Try the following command and repeat several times an input word ("b" in the example).

```
record 12 seed/b.wav
```

It is *best* to record using the *record* command provided by idi and not an external tool since some programs (audacity for example) might give you other results (they adjust mixer settings ?).

We can now test the "cut" algorithm.

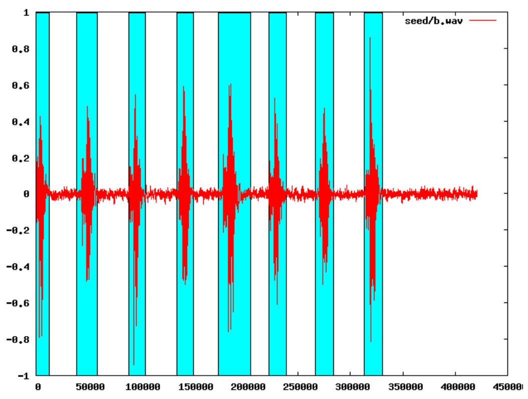
```
splitter seed/b.wav
```

The splitter program will take an input wav file, split it using idi algorithm and generate the corresponding wav files.

First, you need to check that you obtain the exact number of samples you pronounced. You should also check that the cut is taking place at the right place. There is a perl script to help you check that everything is ok.

```
plot_cut.pl b.wav
```

plot_cut.pl will generate an image displaying how the sample is cut. Note that this image is different from the REAL samples generated as not all processing is applied during split.



You can now eventually adjust the different parameters of the cut algorithm. There is a trade-off between speed and accuracy.

Parameters of the split algorithm are located in the configuration file located in ~/.idi/config. Interesting parameters are:

- noise_threshold: above this value the signal will be considered as voice.
- noise_size: we need a signal above noise_threshold for this size before switching to voice.
- silence_size: we need a signal below noise_threshold for this size before switching to silence.

- sample_size_limit: below this size we do not keep the sample.
- kept_size: we keep this more at end of sample.
- start_size: we keep this more at beginning of sample.

### 3.3 Creating initial samples set

You can now record some more samples in the seed directory (10 different words is a good start). After creating all samples, run the *generate_ samples.pl* script to initialize idi's sample directory.

```
generate_samples.pl seed
```

You can check one more time that the cutting algorithm is well parameterized by checking the content of ~/.idi/samples is matching your recordings.

## 4 Training

We are now ready to start training and using idi. The basic idea is the following : when you use idi some words will not be recognized. These words need to be recorded and added to the database. An other possibility is that the addition of new words will generate conflicts with existing ones. For examplie let your database contains utterances for 'a' and 'b' without any problem for identifying 'b'. No problems here because 'a' and 'b' sound rather different. However when adding new samples for 'c' to the base you are likely to see some of your original 'b' samples now confused with 'c'. You will then need to refine your classification of 'b' or add additional samples.

The main ideas of the classifications are the following :

- all samples for word 'a' are located in ~/.idi/samples/a
- ~/.idi/samples/a might contains subdirectories containing more samples of 'a' grouped by similarity
- all of these directories should contain at very least 6 samples (because the models rely on average deviations)
- incoming sounds (usually because samples number $< 6$) are stored in ~/.idi/recorded
- ~/.idi/recorded stores all unidentified samples
- ~/.idi/recorded/a stores all 'a' samples which are in too little numbers
- the game is to classify samples in ~/.idi/recorded in move them in the right ~/.idi/samples folders

To achieve this, you are provided with a lot of tools. The most important script is used to check that samples in the database are matched as they should.

```
check_model.pl
```

This script will rebuild the models and check if each sample in the base is correctly matched. By default samples in 'a/xxx' will be declared matched if they match 'a' or any subdirectory of 'a'. It is possible to enforce a stricter test by applying the '-f' flag. With this flag, the script will accept a sample in 'a/xxx' if and only if it matches 'a/xxx'. It is also possible to specify only one directory (one word) to test in order to avoid long computing times.

Figure 1 displays an example output of *check_ model.pl.* As you can see here, in this snippet one file is mis-identified : sample 'a-804646.wav' which is an 'h' sample is matched as 'e'. Moreover *check_ model* is displaying that of 1411 files in the base, 16 files are mis-identified which gives an error rate of around 1%.

What is the solution to these problems ? *Training* !

First we quickly check that *a-804646.wav* is not a meaningless sample due to a bad cut or a bad recording.

If the sample is ok, we need to record additional 'h' samples. Of all these new samples, most will be correctly identified and present no interest to us. Some however will also be matched as 'e' and they are valuable. When we reach a set of 6 or more 'h' samples mis-matched as 'e', we create a new subdirectory *~/.idi/samples/h/e* and move all corresponding samples there. We then relaunch the *check_ model* script to check for improvements.

The main difficulty here is to find enough 'h' samples mis-matched as 'e'. Such samples might occur very rarely. *~/.idi/recorded/h* serves as buffer directory to store the files until there are enough of them.

This example details the most basic classification process. This is enough to get you started but we will now detail some tools to ease this process.

## 4.1 Recording new files

There are many ways to record additional samples. The first possibility is to use idi itself with the *-r* flag specifying to record. The *-r* flag *requires* another argument which is the name of the word you are trying to record.

For example:

```
idi -r a
```

This will launch idi recording additional 'a'. You can now say as many utterances of 'a' as you want. Idi will try to match incoming sounds against

Figure 1: Extract of check_model log

```
...
testing : l
testing : l/v-b-p-t
testing : l/m
testing : l/n
testing : l/n-m-error2
testing : l/n-m-error
testing : h
error matching /home/wagnerf/.idi/samples/h/a-804646.wav (matched as e)
testing : q
testing : q/u
testing : q/u-q-u
testing : g
testing : g/b
testing : g/p-b-d
testing : g/v-b-v-b-p-t
testing : error
testing : error/e-o
testing : error/dollar
testing : error/m-n-m-error
testing : Shift_L
testing : 30
testing : 1
16 errors out of 1411 files (1.13394755492558 %)
```

'a'. If a sound matches, it is discarded. If not, the corresponding samples will be saved in *~/.idi/recorded/a*.

This method is usually enough for most of training needs. However, in some cases, some unrecognized samples only appear in natural sentences. Thus it might be difficult to record such samples while merely repeating an out of context word.

Another way to record additional samples is by

## 4.2 Sorting files