

Using Mouth for Offline Analysis with Unlock

James Percent
james@syndeticlogic.net

May 23, 2013

1 Overview

This document provides details about using the Unlock framework, on the Mouth workstation, to collect data for offline analysis. Section 2 covers Mouth account setup, Unlock software setup and MOBILab hardware setup, and Section 3 covers collecting data.

2 Setup

A Mouth account is required to get started. If you do not have a Mouth account, then send me an email and I'll create 1 for you.

Next, login to Mouth and launch the Git Bash Shell. To launch the Git Bash Shell, click Start Menu→Git Bash Shell. See Figure 1.

Note that the images in this document have high enough resolution to clearly read the text on them, so if you're having trouble reading the text, then use the Zoom In feature of your PDF viewer increase the size of the image.

The next step is Github configuration. Github needs to be configured with your public ssh key from Mouth. You do not have a public key on Mouth by default, so we need to generate 1 using ssh-keygen. After we generate a

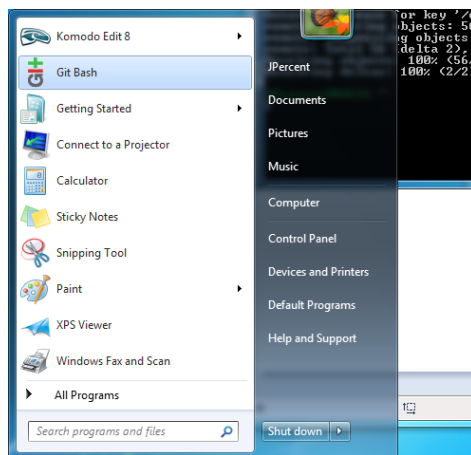


Figure 1: Git bash shell

public key, we need to upload it to Github. Finally, after the public key is uploaded, we can clone the repo.

Figure 2 shows the commands that need to be run from Mouth; in particular note that the generated public key is located in the `.ssh` directory of your home directory; also note, public keys end in `.pub`. To upload the key to Github, click on the Account settings icon, and, after that page loads, click on SSH Keys link.

The MOBILab device must be connected to collect data. Connecting the MOBILab to Mouth consists of 2 steps: powering on the MOBILab and connecting its USB fob. The MOBILab USB is the 1 with the green key-ring identifier.

3 Collecting Data

After completing the steps in Section 2, a directory called `unlock-npl` should exist in your home directory; enter this directory. `Collector.py` is the pro-

```

JPercent@MOUTH ~
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/JPercent/.ssh/id_rsa):
/c/Users/JPercent/.ssh/id_rsa already exists.
Overwrite (y/n)? n
JPercent@MOUTH ~
$ ls -l .ssh/
total 2
-rw-r--r-- 1 JPercent Administ 1743 May 21 16:50 id_rsa
-rw-r--r-- 1 JPercent Administ 396 May 21 16:50 id_rsa.pub
-rw-r--r-- 1 JPercent Administ 407 May 21 16:44 known_hosts
JPercent@MOUTH ~
$ git clone git@github.com:NeuralProsthesisLab/unlock-npl
Cloning into 'unlock-npl'...
Enter passphrase for key '/c/Users/JPercent/.ssh/id_rsa':
remote: Counting objects: 56, done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 56 (delta 2), reused 53 (delta 2)
Receiving objects: 100% (56/56), 60.31 KiB, done.
Resolving deltas: 100% (2/2), done.
JPercent@MOUTH ~
$

```

Figure 2: Bash commands

gram that we use to collect data. Running the collector is simple. To see the options run the following command.

```
$ python collector.py --help
```

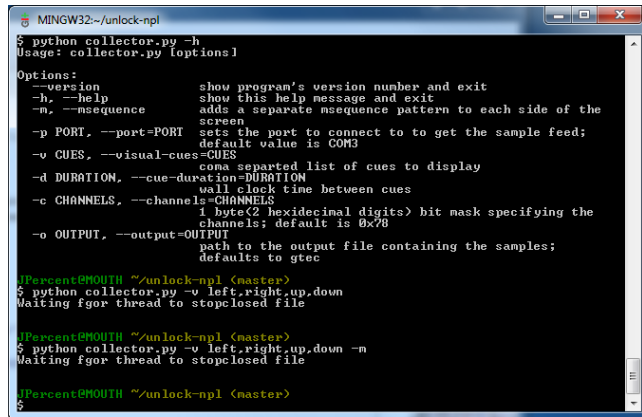
Running an m-sequence visualization, with 4 cues, separated by 5 seconds, can be accomplished by executing the following command.

```
$ python collector.py -v left,right,up,down -m
```

The output file consists of a sequence of samples separated line breaks. Each sample consists of $n = channels + cue$ integers, separated by tabs. For example, a 4 channel sample would look like the following.

```
11123 -2123 23456 1234 0
```

The cue will be the last integer in the sequence. A cue value can only be 0 or 1. Figure 3 shows the help output and some possible executions. The



```
MINGW32:~/unlock-npl
$ python collector.py -h
Usage: collector.py [options]

Options:
  --version            show program's version number and exit
  -h, --help           show this help message and exit
  -m, --msequence      adds a separate msequence pattern to each side of the
                        screen
  -p PORT, --port=PORT sets the port to connect to to get the sample feed;
                        default value is COM3
  -v CUES, --visual-cues=CUES
                        comma separated list of cues to display
  -d DURATION, --cue-duration=DURATION
                        wall clock time between cues
  -c CHANNELS, --channels=CHANNELS
                        1 byte(2 hexadecimal digits) bit mask specifying the
                        channels; default is 0x78
  -o OUTPUT, --output=OUTPUT
                        path to the output file containing the samples;
                        defaults to gtec

IPercent@MOUTH ~/unlock-npl (master)
$ python collector.py -v left,right,up,down
Waiting for thread to stop/closed file

IPercent@MOUTH ~/unlock-npl (master)
$ python collector.py -v left,right,up,down -m
Waiting for thread to stop/closed file

IPercent@MOUTH ~/unlock-npl (master)
$
```

Figure 3: Example collector usage

default number of channels is 4, the default duration between cue events is 5 seconds, and the default output file will be named gtec concatenated with a timestamp. Figure 4 shows what an m-sequence visualization looks like.

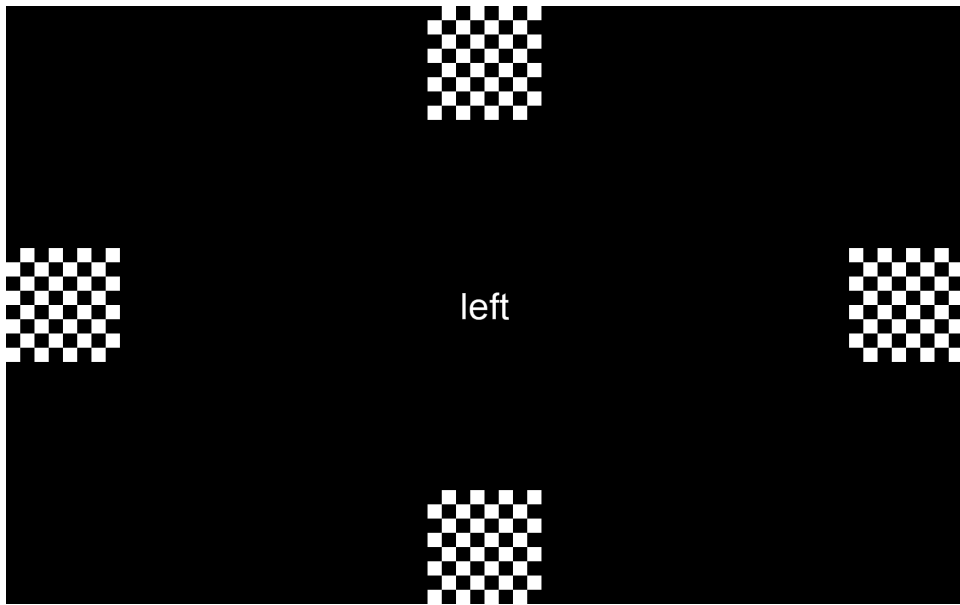


Figure 4: Example of running the m-sequence collector