Help Manual

System requirements:

Hardware:

Balltrack setup with two optical mouse sensors for reading data and a working network connection between the computer with the visual stimuli software and the one with FlyTracker. Network connection is not mandatory barring for using network trigger.

Software:

- Ubuntu Linux (for sure works on: 10.04 LTS and 12.04 LTS, no other tested)
- Matlab (for sure works on: R2012a no other tested)

Setup:

Put the FlyTracker-folder anywhere on the desktop but make sure that a path (with subfolders) is added to it in Matlab.

There is a lot of reading and writing to files going on in FlyTracker, relative paths are set within in the FlyTracker folder and therefore it is very important that files are not moved manually within in the main folder. Only exception is the data folder which contains data from experiments as these are never read from, you can also set the save path to anything you like in the GUI.

The path also has to be manually set in utilities.py (located in the python subfolder), set to the location of FlyTracker folder.

Calibration

Calibrating the setup is the most complex task, it is done in six setup steps:

1. Password – The user needs to enter their user login password as this is needed to get root access which in turn is needed to read data from USB mice. This is a potential error source as an incorrect password here will render the program useless
2. Number of mice – Just make sure that there are exactly three mice (including the two used as sensors) connected to the computer.
3. ID-mouse – In this step the user is prompted to press the button “Identify mouse” and then move the mouse using the only actual one. Here an error message will occur if for example the user entered the incorrect password in step one. The user will be shown either a success message or an error-message.
4. Ball settings – Enter diameter of the ball used in the setup and the angle denoted as “a” in the image. The arrow is what will be defined as the fly's forward direction. Its also adjusted for the fact that the ball rotating in the direction of the forward vector is actually the fly going backwards.
5. Calibrate translation – The first calibration step where the user calibrates translation (forward and sideslip). If the angle “a” in the previous step is set so that a mod 45 = 0

both sideslip and forward movement are recommended to be done as each will only use one of the sensors readings. In any other angle its sufficient to calibrate for either forward or sideslip only. The distance that is used needs to be the same for each run, though it can vary between sideslip and forward. The ball needs to be rotated away from the sensors, if not the positive and negative direction will be flipped.

6. Calibrate yaw – The second calibration step. Here the user just need to enter the rotational distance in degrees. The ball should then be rotated counter-clockwise to make clockwise the positive direction in the fly's rotational plane. If you rotate the ball the other direction it will still work but the negative and positive directions will be flipped.

To be able to run the program for real experiments the software needs to be correctly calibrated. Before that it is not possible to perform experiments, this is implemented as a safety measure.

After calibration is done, it is still possible to edit certain parts of the calibration values by redoing that specific part without having to go through all steps again. You can for example change the angle of the fly in relation to the sensors and change the size of the ball if the setup allows that. What you also can do is to redo calibration for translation and yaw respectively.

Trigger

No Trigger

System can be run without trigger, either by having the user manually start and stop the recording or setting a timer for how long it should record.

Setting up the network trigger client

The simple trigger client is implemented in python and is provided as the file TriggerClient.py. This file needs to be called from the visual stimuli software so that commands can be sent over the network.

Currently the Python client supports communication via a named pipe, this is a sufficiently fast solution for interprocess communication that is also allowing many different platforms for the visual stimuli implementation.

The server accept three types of commands, “start”, “pause” and “quit”. Needless to say you first need to start the recording by sending “start” and finally its important that the server is stopped by providing the “quit” command as otherwise it will run until time out.

To be able to run it three things needs to be configured.

Host: Set this to IP-address or hostname of FlyTracker, the information can be found in the menu of the main window of FlyTracker. Choose Settings>Network Settings.

Use either hostname or IP-address.

Port: Use the same port number in the client as in FlyTracker. This can be changed but

beware that the new port is open and valid. It normally needs to be larger than 1024 and lower than 65534 but can vary between platforms.

Pipe: the pipe in TriggerClient.py needs to have the same filename and path as the pipe created in the visual stimuli software.

Matlab sample code for the stimuli system:

Before drawing commences:

```
pipe = '/home/Username/pipe';
system('mkfifo /home/Username/pipe');
system('python /home/..../TriggerClient.py');

fid = fopen(pipe,'w');
fwrite(fid,'start');
fclose(fid);
```

Pause:

After drawing is done:

```
fid = fopen(pipe,'w');
fwrite(fid,'pause');
fclose(fid);
```

Using the quit command is not necessary because you can use stop in FlyTracker to stop the recording and close down the network connection.

Saving data

Data is saved in different ways depending on what kind of trigger system is used. The recording has three states, “started”, “paused” and “stopped”. One datafile is created from the moment when it first is started until its stopped. If the recording is paused during a recording that will not create a new data file but rather divide the file into data blocks.

Currently only the network trigger supports pauses.

Data is saved into .mat-files with the filename of the current time with second precision. Each data file consists of one 5 x n cell where n is the number of blocks. The elements of the first row in the cell consists of arrays of forward velocity, second row is for sideway velocity, third is for rotational velocity and the forth one is a time stamp. Each column represents data for that particular time stamp. The final row of the cell holds the starting time of that block so it does not hold an array but rather a date and a time.

Plotting

There are four plots, the three first ones display velocity, position over time or change in

position for each of the three degrees of movement (forward, sideslip and yaw). The fourth one displays a 2D map of the fly's path. When using network trigger data is potentially divided into blocks, in this case not all the data will be displayed but rather the latest block. After stopping the recording the user can go back and look at older blocks using the drop-down menu.

Changing what to plot is done in the settings-menu.

Plotting is done in pseudo-real time by redrawing the plot every 100 data points. This can be changed but only by changing the hardcoded value in the readdata.m file.

The Python-Matlab interface

The main application uses both Matlab and python code. The communication between them is limited as much as possible and is exclusively done via system calls reading from and writing to named pipes.

Matlab package

Given Matlab's weak support for object oriented programming there is a high level of coupling in the matlab code but a division into interface files and data processing files has been done.

Model
Holds all files dealing with reading and processing data.

View
The view holds all files regarding presenting the data and handling user interactions.

Python package

DAQ file

The Coordinates object is a dictionary object and must be such, its however trivial to add new parameters to this and the json-parser in the Matlab part of the application can still read and retrieve the new parameter.

MouseHandler
This class deals with instantiating the mice, inherits from Thread and runs a loop that evaluate the mice coordinates and saves them to file anytime they read data and also timestamps this. MouseHandler thereby runs three threads, its own loop that saves data and the two sensor-mice each one in a separate thread.

Mouse class tree
Currently two types of mice, SensorMouse and IdMouse. Inherits from AbstractMouse which in turn is of type Thread. The SensorMouse runs an infinite loop reading from its mouse and writing the results into the coordinates dictionary. IdMouse is just used to id the mice used as sensors and the mouse used as an actual mouse.

MouseID
This file holds the code for identifying the connected mice. It is dependent on that there are three mice in total and one of them is a regular one and the other two are the sensors.

Utilities

A utility class providing static methods for different kinds of data manipulation, file I/O and exception logging for debugging.

Calculation of fly movement

The algorithm to calculate the mouse inputs to fly walking movement is pretty straightforward. The main algorithm is copied from “FicTrac: A visual method for tracking spherical motion and generating fictive animal paths”. Ball movement measured by the horizontal component of each mouse sensor is transformed into the yaw component of the fly's movement while the vertical components of the sensors are used to calculate translation.

Some issues have been detected regarding yaw rotation and it's important that the ball is at such height that the sensor is aimed at the center of the ball. Otherwise a non-significant yaw component will be measured even in pure forward and sideslip movement.

To further reduced this issue an algorithm which always picks the smallest absolute value between the two horizontal readings (unless one is zero in which case the average is used) and the average of the two has been implemented. No scientific testing has been applied to test whether it actually reduce the issue or not but from what we can see it does.

Known issues:

- Sometimes the sensors seem to stop reading, not sure why this is the case but might be to the surface of the ball not having enough varying texture or the distance between the ball and the sensor. It does seem to be solved by disconnecting the mice for example.
- If FlyTracker is closed during a recording or a recording is stopped incorrectly for example by using CTRL-C the python process spawned by FlyTracker will not be closed. If then FlyTracker is restarted and recordings are made the data will be corrupt. To solve this close Matlab or if you are familiar with how to close processes from the terminal window kill python from there. Symptoms of this is clear as plot will be nonsensical.
- As the main application is in Matlab users won't be explicitly warned about any exceptions in the python process. Therefore there is a log-file in the data-folder where all thrown exceptions are saved so if FlyTracker is acting strangely, having a look in this txt file could provide answers.