**“User-Centered Process for Designing and Implementing Interactive Motion Tracking Software”**

1. Introduction

When designing a completely new software system there are many factors to consider. One of the most important ones are the future end users. This is especially true for systems aimed a small group of end users as it will provide the possibility to tailor make the software to a much greater extent. Depending on the background of the end user, restrictions can be put on the software system and demands might vary greatly. As a result of this, involving the user in the actual software development process can produce a superior software system. The option to involve the user in the entire process might not always be available, but if it is the user can prove to be a invaluable asset in the software development process.

Motion vision is very important for any seeing animal species. Motion vision is for example used to track moving targets or to estimate self motion from optic flow (Borst and Euler 2011). There are several aspects of motion vision to study, an important one is to study at behavior as a function of visual stimuli.

Using insects for motion vision research is easier than using vertebrates. Surprisingly enough certain insects code motion vision in a fairly similar manner as vertebrates do.

Getting a better understanding of motion vision is important for advanced technology such as self driving cars, air planes and robots. Learning how motion vision is processed in for example insects is particularly interesting as they apparently process this rapidly despite having very limited brain capacity.

2. Background

* 1. Motion vision at Uppsala University

At the motion vision lab in the neuroscience department of Uppsala University flies, and in particular hoverflies, are used for their research. For doing behavioral research different methods are used, for example you can record the fly's movement in a contained area while presenting visual stimuli for it and then do image analysis of the recorded material to get the bhavioral data. Another technique that the lab at Uppsala University is using is a trackball setup on which a tethered fly can walk, thus generating a virtual movement path.

The setup is built by using two optical sensors extracted from two high quality gaming mice. These are held in place in such a way that they are perpendicular to each other and aimed at the center of the cup where the ball is placed. Using light airflow streaming from the bottom of the cup the ball is hovering slightly and very easy to spin which is necessary for the fly to be able to rotate the ball. The fly itself is tethered so that it hangs low enough to be able to walk on the ball and high enough not to be pushed into it. Furthermore, the fly is aimed at a CRT monitor, which in the experiment is going to be used for showing visual stimuli.

Insert figures of setup here  
  
The two mouse sensors on the trackball setup are going to be used as simple motion detectors. This means that there is a need for software that can read the raw mouse data and decode these to the fly’s virtual delta coordinates. As there are little to none ready-made and available software alternatives for this purpose a new software system is needed.

* 1. Agile software development

Agile software development methods are the group of methods that follow the principles of the agile manifesto. They are also considered to be the opposite of the other large group of software development methods, i.e. plan based processes (Beyer). In plan-based methods all activities need to be planned and scheduled extensively before they are started (Sommerville). In contrast to this, trademarks of the agile methodology are rapid design, development and releases of working prototypes, welcoming changes in customer requirements rather than the opposite and a lot of informal face-to-face communication, as opposed to communicating through large amounts of unnecessary documentation. (Paulk 2002) Agile methods were developed since plan based methods had poor performance. Large project were planned ahead of time and during the project requirements changed and time schedules were too tight, which in the end lead to abandoned projects. (Beyer)

In many of the agile methods, for example Scrum, work is divided into shorter sprints. Sprints are initialized by choosing requirements to implement during that sprint. These requirements are then frozen for that time and when the sprint is over these should be fully implemented and testable. At the end of each sprint an evaluation session is held where the customer can test the working prototype and give feedback on the project. During this session there is potential for adding, removing or updating requirements, along with changing customer demands. (Beyer)

* 1. Adding user-centered practices

User-centered software design can have different focus and be using different tools depending on the work environment and the type of software system to be implemented. Some of the tools described in my project are described below (Beyer).

**2.3.1 Contextual inquiry**

Contextual inquiry means that the design team performs an analysis of the end-user by observing them doing their actual work and seeing in which manner they perform these tasks. This is called contextual interview. In this phase, design of the system is not in focus, but rather what the working environment looks like and how work generally is done in this particular context. The objective is not to look for problems but rather to understand the users and their need.*[[1]](#footnote-1)*

After the contextual inquiry all notes and data gathered about the user need to be organized in a logical fashion. Since the data might be overwhelming and can be difficult to structure, *interpretation session* is held. For this a*ffinity diagrams* may be used for organizing the data. These diagrams are built bottom up by building small groups of items that have a connection and then iteratively build larger groups of newly created groups.[[2]](#footnote-2)

**2.3.2 Sequence models**

*A sequence model* is a model describing user strategies for performing tasks, the model defines the intent and the steps required to perform a task. The model describes in detail exactly what the user currently does when solving certain problems. Modeling this gives an overview of what is actually happening and what potential issues there are. A sequence model is therefore a good tool to possibly find better strategies than the ones currently employed by the users.[[3]](#footnote-3)

**2.3.3 Scenarios**

There are different types of usages for scenarios, such as conceptual scenarios. A conceptual scenario is an abstract description of how a user performs a specific task There is no detailed description of tools and technologies used. nor suggestions for how the tasks should be solved. Using conceptual scenarios is a good way to generate requirements as it leads to concrete scenarios which are descriptions with high level of detail. These will provide suggestions for the future interface design (Benyon 2010).

**2.3.4 Prototypes**

Using prototypes is a good way to provide the user with a clear description of what the interface is going to look like early in the design phase. Prototypes are also letting the user try these, whether the designer is using low-tech prototypes such as paper prototypes or via some prototyping software creating an interactive interface without its actual functionalities. If the case is the former the designer can use post-it notes as pop-ups and cover invisible parts with paper and removing it as the user makes them visible by interacting with it. Paper prototypes are a good tool for getting the user get a sense of the look and feel of the future system, without actually having to program a GUI. This makes it easier to do major changes to the structure of the GUI. As the project moves on these prototypes will not be sufficient. However as the actual GUI is produced, the user can give feedback on that. Hopefully any future changes will not be major.[[4]](#footnote-4)

* 1. Usability

Usability is a very important concept of human-computer interaction and is a measurement you can use in an evaluation of a system’s user friendliness. Usability means that usage of the system should require as little effort as necessary when performing the tasks needed. The system should provide the user with necessary information and be organized in a natural way in the GUI. [[5]](#footnote-5)

That the software has a short learning curve is also a feature of a system with high usability. Design principles are important tools for shortening the learning curve. Design principles are artifacts that are often found in similar systems that are easy to recognize for the users. This means simple artifacts such as “undo”-/”redo”-buttons, “back”-buttons for navigating, and generally just placing interface items where they would be expected to reside in a similar system.[[6]](#footnote-6)

Emphasis on usability tends to diminish in the software process for two main reasons. One is caused by the fact that the customer rarely specifies that they want a usable software system (they think it is implied). When the company delivers the software it is not as user friendly as the customer might have wanted. (Göransson)

The second issue that might arise is that usability experts are normally only actively participating in the early parts of the software development process and not as much in the implementation and testing phase. Since they are rarely software developers. What is needed to solve this is HCI experts that are familiar enough with programming to be able to participate actively in the implementation phase in particular, but also in the validation phase where system requirements are validated (including usability requirements).[[7]](#footnote-7)

* 1. Evaluating usability

Beror lite på hur slutliga utvärderingen blir, skriver därför detta i samband med utvärderingen.

1. Purpose and methodology
   1. Purpose

The motion vision lab needs a brand new user friendly system, FlyTracker, that can perform the tasks needed to study the behavior of a fly mounted on a track ball setup. To verify the goal of high usability, an evaluation of the finished software system needed to be done.

* 1. Choice of development process

As I, as a developer and designer, worked in the very same lab as the end users in the motion vision lab at Uppsala University, choosing an agile development process was natural. The initial phase of the project, also called phase 0, devoted to project description. This included finding and analyzing requirements of the software system, designing a suitable graphical user interface and planning the implementation phase.

The implementation phase was divided into four sprints of two weeks each. Each sprint was finalized by having a walkthrough of the progress so far, where FlyTracker was tested by checking of the list of requirements for that particular sprint. This made it possible to every second week get valuable user-feedback and to make sure that the development was on track. Furthermore, this also made it important to divide the work such that functions (or requirements) were completely implemented during its respective sprint. In the middle of every sprint there was also a shorter check-up meeting where we discussed potential problems or changes to the requirements before they were fully implemented.

Working according to the principles of user centered design was also a clear choice given that the end users were few and that I worked in the same lab. This meant that there, in addition to the scheduled meetings, were a lot of informal discussions about the project and questions could be asked about even the smallest details as they were implemented. This is something that the agile manifesto is clearly emphasizing (http://agilemanifesto.org/).

* 1. Implementation

The employees in the motion vision lab at Uppsala University are familiar with Matlab and also have some experience with Python. Therefore there was a requirement that the software system was implemented in Matlab. Using Python for certain modules of the system was acceptable, as it was mandatory for a few specific functions.

* 1. Evaluation

The evaluation was performed by letting the main user of FlyTracker try it out for a few weeks. An interview was then conducted where she got to answer questions about (see appendix xx) the system and the results of this interview were analyzed. No major problems with usability were found. Skriver klart när jag vet exakt hur det går till

1. Theoretical framework
   1. Usability in the software process

Performing a user-centered system design as a part of the overall software process means that you will need to implement a user-centered requirement framework in the selected software process. Zimmerman and Grötzbach suggest one framework where they introduce three types of non-functional requirements, usability requirements, work flow requirements and user interface requirements.

Usability requirements are..

Work flow requirements are a description of how the software system is supposed to support the user when trying to perform certain tasks. They are a description of what actions the user will need to take and how the system will achieve these. The actions needed to be performed can be described in analysis artifacts such as use cases or scenarios. These requirements can when the system is finished be used to evaluate the usability of the system and the requirement itself can be validated and see that the actual work flow corresponds to the requirement.

* 1. User interface requirements define how the interface should look like and translate into design artifacts such as sketches, navigation models, information architecture and eventually paper prototypes.[[8]](#footnote-8)
  2. Why agile user-centered methods?

Agile methods are the name for the group of software development processes where the project is divided into shorter iterations or sprints. Each iteration may last a few weeks where in the beginning goals for that iteration are defined and in the end the result is presented for the managers. During the sprints most of the phases of the software process are gone through, requirements are analyzed, designed for and implemented. This way of working has several advantages, first of all, in a rapidly changing environment that the software system is created new requirements may appear and old ones might be in the need of change. Following a plan based approach; months of work might be lost because of the huge amount of initial planning.

The key is in other words rapid and dynamic development which is also in line with what suitable for processes where the end users are involved, user-centered development processes.

* 1. Phase
  2. Release planning phase and sprints

In the release planning phase the sprint planning is done, first of user stories are created on so called story cards. These describes requirements for the systems from the user's perspective. The description consists of what the user want to do and for the development team this description is all that matters and any other restrictions or demands on the task needs to be specified in their own user story. In other words stories will be short and this serves another purpose in that they can be implemented quickly which is a great advantage in the agile process. Developers can implement a new story within hours or days and if there are problems which renders the story obsolete not much time has been wasted.

After all story cards are created the sprint planning can start, usually a sprint will go on for a few weeks and this includes both implementing and testing. Each sprint will be allocated a certain amount of stories where the number depends of the complexity of the stories. The planning should makes sense in that any stories that depends on other stories need to be scheduled after their dependencies. Furthermore, stories that are more important should be implemented early on and so should stories that are dependent on complex technology. In the case of issues with that particular technology there is than still time to fix these compared to if those stories would have been pushed back to end of the development phase.[[9]](#footnote-9)

* 1. Methods for evaulating usability

Evaluation techniques are often grouped into two categories, namely expert evaluation and participant-based evaluation. The latter means that end users or a group representing them will be a part of the evaluation, mostly by actually using the system and having to answer questions or surveys. Expert evaluation means that the system under design is evaluated by usability experts. This method however should never be used by the designers themselves as they could have significant bias towards the system as they know it very well and could potentially both find too few problems or obscure problems that aren’t realistically going to occur during regular use. [[10]](#footnote-10)

One example of participant-based evaluation is cooperative evaluation. This means that a user will try performing predefined tasks (these tasks should of course be part of realistic future use) together with the expert performing the evaluation. All this could be video- or audio recorded to get the most out if it but it can also be sufficient that the expert is taking notes on how the program is performing. During the process the participant will be encouraged to talk out loud and the expert will also be asking a series of questions. For a detailed description of guidelines for a cooperative evaluation see table 1.[[11]](#footnote-11)

1. Table - Guidelines for cooperative evaluation. [[12]](#footnote-12)

|  |  |
| --- | --- |
| Step | Notes |
| Using scenarios prepared earlier, write a draft list of tasks. | Tasks must be realistic, do-able with the software and explore the system thoroughly. |
| Try out the tasks and estimate how long they will take a participant to complete | Allow 50 percent longer than the total task time for each test session |
| Prepare a task sheet for the participants | Be specific and explain the tasks so that anyone can understand |
| Get ready for the test session. | Have the prototype ready in a suitable environment with a list of prompt questions, notebook and pens ready. A video or audio recorder would be very useful here. |
| Tell the participants that it is the system that is under test, not them; explain and introduce the tasks | Participants should work individually – you will not be able to monitor more than one participant at once. Start recording if equipment is available. |
| Participants start the tasks. Have them give you running commentary on what they are doing, why they are doing it and difficulties or uncertainties they encounter. | Take notes of where participants find problems or do something unexpected, and their comments. Do this even if you are recording the session. You may need to help if participants are stuck or have them move to the next task. |
| Encourage participants to keep talking. | Some useful prompt questions are provided below. |
| When the participants have finished, interview them briefly about the usability of the prototype and the session itself. | Some useful questions are provided below. If you have a large number of participants, a simple questionnaire may be helpful. |
| Write up your notes as soon as possible and incorporate into a usability report. |  |
|  |  |

1. Analysis and design
   1. Contextual inquiry

As I have been working closely with the end-users in the very lab they are going to be working with the project I felt no need for a formal contextual interview but rather I got the chance to observe how work was performed all the time. I could ask questions about routines, issues that might arise and generally observing work as it happened. This was especially relevant for the design of the interactive calibration wizard. The advanced hardware setup proved to be difficult to calibrate.

* 1. Scenarios

The user has finally set up the experimental environment and having software for visual stimuli installed on one computer and the data acquisition software on another machine. He now feels ready to finally start the experiments he needs for his research. Though he is comfortable with the stimuli software he has never used the DAQ-software and starts it up to look around.

* 1. Sequence models

Intent: Perform an experiment without trigger

Trigger: User starts program and calibration is done

* Load old calibration file
* Change settings to no trigger and plotting to the preferred plotting
* Press Run
* Press Stop when needed

Intent: Calibrate software

Trigger

* First program run after changing setup
* Measure ball radius and decide what is defined as forward in relation to the two sensors
* Identify each sensor and the desktop mouse
* Calibrate translation by moving the ball in a way that simulates translation in the fly’s coordinate plane. Also measure this distance
* Do this several times and measure average error to find and decide whether or not its acceptable
* Redo this for yaw
  1. Paper prototype

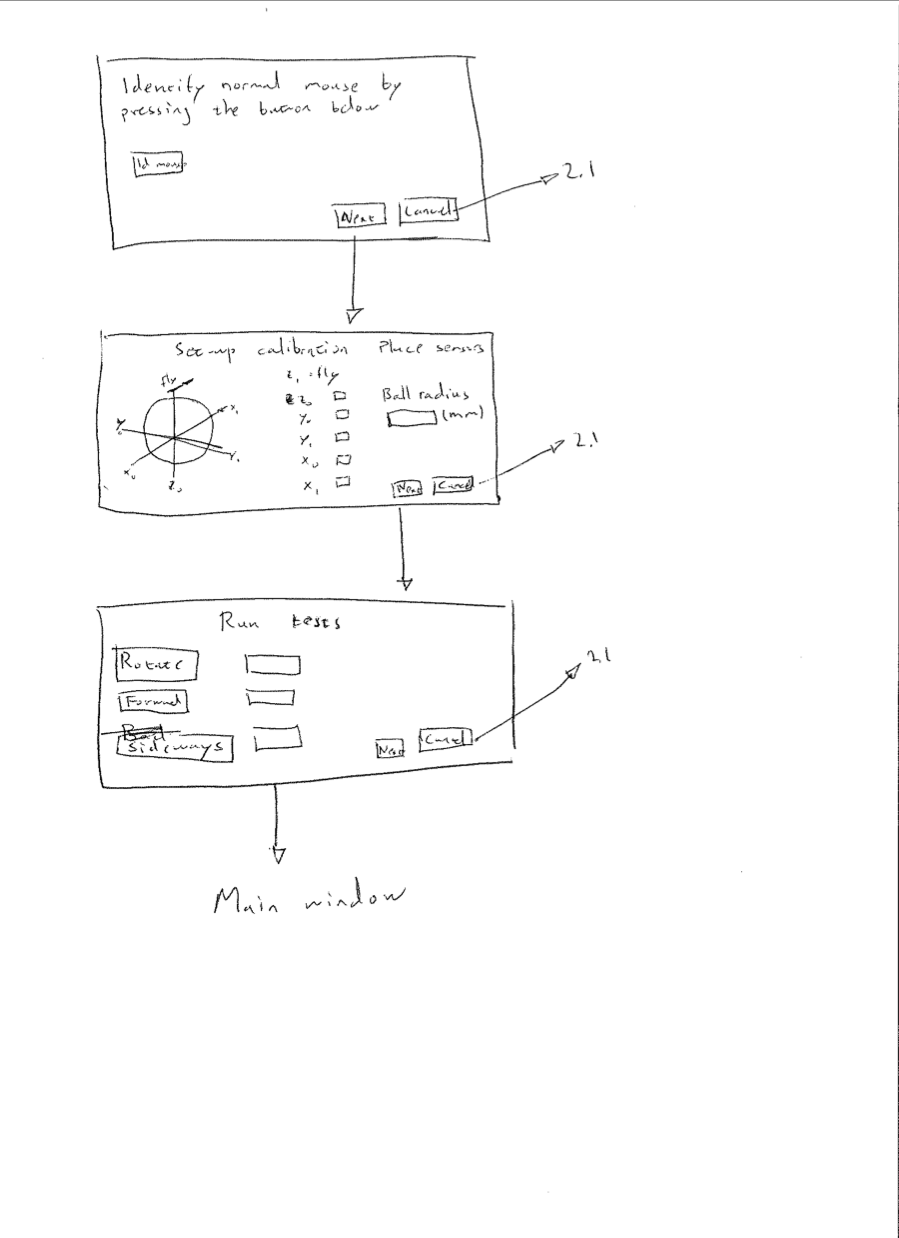
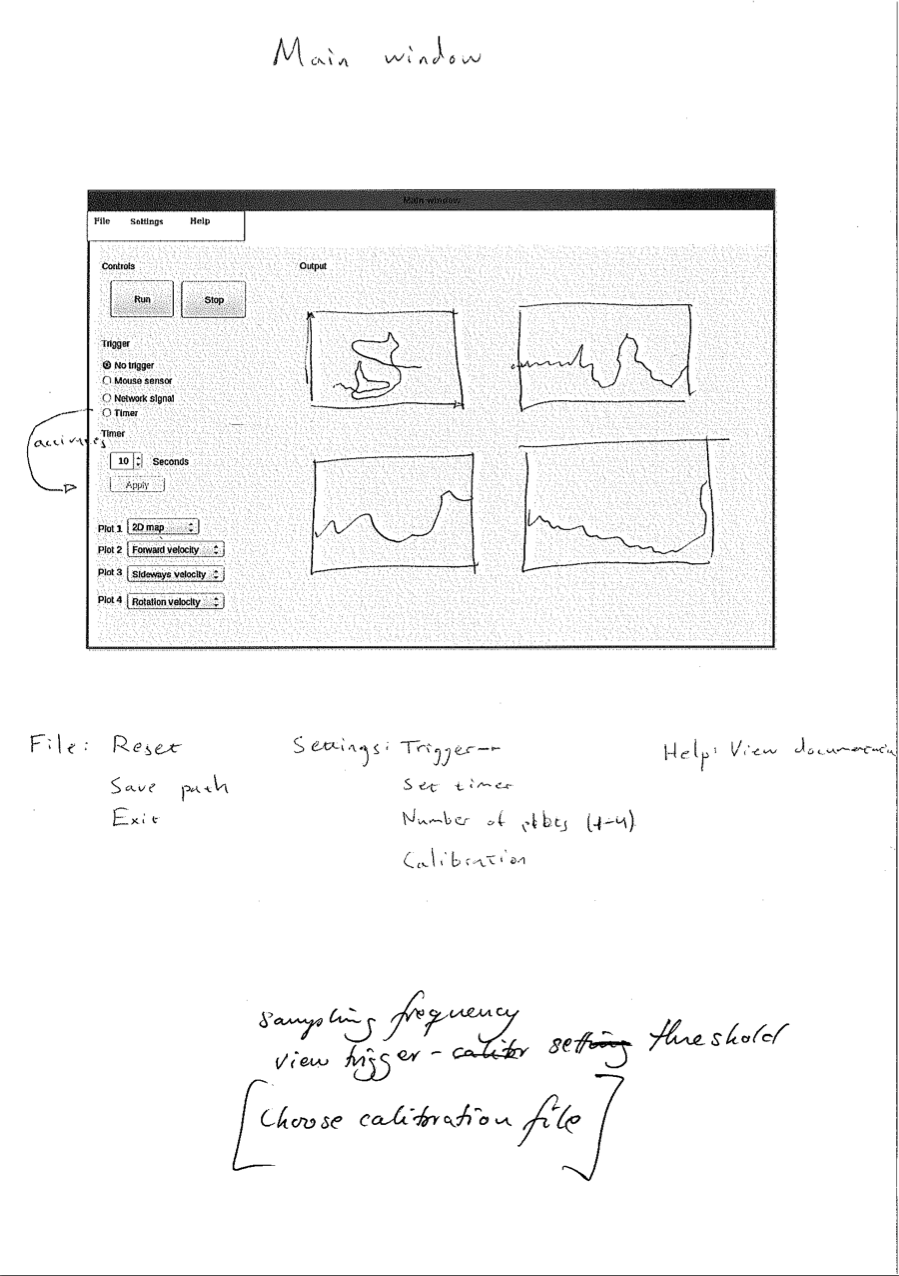
The paper prototype was made in two iterations before the actual draft of a GUI was first implemented. Using a combination of the open source tool Pencil and actual drawings a prototype was produced with no interaction for the user but clearly specified what actions are possible and what they lead to. This was then displayed for the end users and they had a chance to give feedback.

The prototype for the calibration step is displayed in figure xx:



Figure 1 - The calibration wizard (1/2)

How to implement the interactive calibration part was the biggest issue when designing the GUI.

1. Implementation

FlyTracker is implemented in Matlab and Python and the former provides the GUI and most of the calculations while the latter provides the core of the functionality. FlyTracker is also platform dependant and can only be run in Ubuntu Linux although changes to the Python module could make it more portable. In fact it would be fairly easy to replace the Python package right now as it is weakly coupled with the Matlab package.

* 1. The Matlab module

The Matlab module consists of two subpackages, model and view.

**6.1.1 Model**

The model holds the files that process the data, first by loading the temporary data from the txt-file that is created in the Python module.

Readdata.m is except for one of the GUI-files the most important part. As the name suggests, it is in this function that the temporary data file is read and parsed into the correct format. The readdata file also plot the current data continuously, updating the plot every hundred data points.   
  
What is worth mentioning is that there is no continuous saving however, the temporary data file holds all necessary information recreate the data though. So should FlyTracker exit unexpectedly or be closed before the recording has been stopped all data from that experiment can be recovered on FlyTracker start-up.

**6.1.2 View**

All of the files holding functionality for the GUI and dealing with user interaction are placed in the view. The main window here is the main window of FlyTracker, the result can be seen in fig x and be compared by the initial prototype.

An important part of FlyTracker is the calibration,

**6.1.3 Data**

The data is saved into mat-files. Each file is initialized by pressing “Run” in FlyTracker and saved when the user presses “Stop”. Everything that is recorded in between is saved into the same data file. If you are using network trigger, recordings can also be paused and each pause then creates a new data block within the same mat-file.

Data is saved into .mat-files with the filename of the current time. 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 thrust delta position, second row is for sideslip delta position, third is for yaw delta position 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, i.e. not an array but rather a date and a time.

* 1. The Python module

There are three files in the Python package, DAQ.py, idMouse.py and utilities.py.

**6.2.1 DAQ.py**

This is the main file where most of the essential functionality is placed. It consists of the parent mouse class called AbstractMouse (though not technically an abstract class). IdMouse and SensorMouse both inherit from AbstractMouse which in turn inherits from Thread. SensorMouse provides functionality for reading raw mouse deltas and storing them in the global variable coordinates. IdMouse is used to identify which mice are the sensors and which mouse is the desktop mouse in the calibration step.

The MouseHandler class is also running in its own thread and is as the name suggest responsible for handling the mice. It initializes the mice and starts and pauses them accordingly.

As the data acquisition is run in Python but the user interaction is dealt with in Matlab there has to be a way to communicate between these two. The MouseHandler is never stopped unless a stop command is sent from the Matlab application (exception is if you run FlyTracker with a timer as a trigger). The command is sent via a named pipe (which requires Linux) and is stopping the listening thread. All mouse threads are set to daemon which means that when all non-daemon threads have exited the Python process will terminate. As the terminating threads will exit abruptly there is potential for minimal data loss (one or two data points).

**6.2.2 idMouse.py**

IdMouse is used purely for identifying the mice .

**6.2.3 utilities.py**

Utilities.py provides some useful classes for saving and loading to file, logging exception and performing some necessary data processing.

* 1. Trigger

The triggering system is a major part of the functionality of FlyTracker. A trigger is basically an on-/off-switch for the recording. For example if the user uses a network trigger the recording won’t start as the user press “Run” but instead it will open up a network socket that listens for commands from a trigger client. When a client sends a “start” command the recording commences and runs until the client sends the “pause” or “quit” command or alternatively the user press stop.

The first triggering option is to use no trigger at all in which case you can manually control when FlyTracker is recording by pressing “Run” and “Stop”.

There is also an option to use a timer as a trigger which means that the user can set a timer for how long FlyTracker is supposed to record data.

* 1. Future expansion

A possible development of FlyTracker would make the entire experimental system to a closed loop system. With that means that as visual stimulus is presented to the fly, the behavior is recorded in FlyTracker and as a function of the behavior new visual stimulus is calculated. A closed loop experiment would set high demands of latency of FlyTracker and as the system is developed now with both a Matlab and Pyhon module it would be difficult to satisfy these demands. An alternative would be to possible migrating all data processing to the Python module and only display the result in Matlab. This would cut a lot of time as transfer of data between processes is currently the most time consuming part of FlyTracker. It would still be necessary to transfer the data to the Matlab module but as this would only be for displaying the output the time critical element would disappear.

7 Evaluation

* 1. Why cooperative evaluation?
  2. Process

The candidates for the evaluation were chosen from the institution of neuroscience at Uppsala University. Although they do not work with motion vision they are PhD-students in the same field as the potential end-users. Four people were chosen for a qualitative study of the usability of FlyTracker.

Each evaluation started with an introduction of what kind of research is done at the motion vision lab so that they get a general understanding on what the purpose is and what tasks they actually are going to perform. They were also encouraged to ask questions in case there was anything that was not clear.

There were two main tasks for the experimental subjects to perform. First of the system was to be calibrated following the steps of the calibration process but not setting up the actual calibration set up as that would be too time consuming and would not fulfill any purpose. The second step would be with the pseudo-calibrated system performs an experiment using the lab's visual stimuli software and moving the ball manually to simulate the fly’s movement.

* 1. Result

8 Results

9 Analysis and discussion

10 Conclusion

Bibliography

Articles:

Borst, A and Euler, T “*Seeing Things in Motion: Models, Circuits, and Mechanisms”*, Neuron 71, September 2011

Moore, Richad, Taylor, Gavin, Paulk, Angelique, Pearson, Thomas, van Swinderen, Bruno and Srinivasan Mandyam, ”*FicTrac: A visual method for tracking spherical motion and generating fictive animal paths”, Journal of Neuroscience Methods, 14*

Benyon, David “Designing interactive systems – A comprehensive guide to HCI and interaction design”, 2010

Beyer, Hugo “User-Centered Agile Methods”, 2010

Paulk, Mark, “Agile Methodologies and Process Discipline”, 2002

1. Beyer, Hugh, *User-Centered Agile Methods* [↑](#footnote-ref-1)
2. Beyer, Hugh, *User-Centered Agile Methods* [↑](#footnote-ref-2)
3. Beyer, Hugh, *User-Centered Agile Methods* [↑](#footnote-ref-3)
4. Benyon, s.184-187 [↑](#footnote-ref-4)
5. Benyon, David, *”Designing Interactive Systems”,* s. 84 [↑](#footnote-ref-5)
6. Benyon, D s.89 [↑](#footnote-ref-6)
7. Göransson, B, Gulliksen, J, Boivie, I, *”The Usability Design Process – Integrating User-Centered Systems Design in the Software Development Process”* [↑](#footnote-ref-7)
8. Zimmerman, D, Grötzbach, L *”A Requirement Engineering Approach to User Centered Design”* [↑](#footnote-ref-8)
9. Beyer [↑](#footnote-ref-9)
10. Benyon, s.228-232 [↑](#footnote-ref-10)
11. Benyon, s.232 [↑](#footnote-ref-11)
12. Benyon, s.232-233 [↑](#footnote-ref-12)