**“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. Beskrivning av varje heuristik

Nielsen’s usability heuristics can be used as guidelines when evaluating a system’s or a prototype’s level of usability. The heuristics are:

* Visibility of system status
* Match between system and real world
* User control and freedom
* Consistency and standards
* Error prevention
* Recognition rather than recall
* Flexibility and efficiency of use
* Aesthetic and minimalist design
* Help users recognize, diagnose and recover from errors
* Help and documentation

(<http://intra.iam.hva.nl/content/1112/verdieping1/research_for_design/intro-en-materiaal/RfD-Heuristic-Evaluation.pdf>)

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. Furthermore it was straightforward to call Python files from matlab.

* 1. Evaluation

As FlyTracker is aimed at a small group of end-users at the motion vision lab at Uppsala University the choice of evaluation method was straightforward. Although FlyTracker will be shared with other universities the main users will still be at the motion vision lab. Thus, after the implementation was done the main user tried FlyTracker for almost a month before an interview was conducted. This made it possible for her to find potential issues that needed to be fixed but not long enough so that she learnt how to circumvent them. Evaluating the software from the perspective of the actual users is more important than evaluating the software from the perspective of naïve users in this particular case.

The interview was semi-structured, although I knew what I wanted to find out it was still a good idea to leave room for exploring issues I had not thought about. The interview was not recorded, although this is normally a good idea it was not necessary as I am working in the same lab as the interviewee and therefore could follow up on unclear notes. (Lindgaard 154-160)

Nielsen’s ten usability heuristics were used as a guideline when writing the interview questions and were used for setting the goals and measurements of the evaluation.

1. Analysis and design

The software development process begins with an initial analysis and design phase. As the project was performed according to the trademarks of an agile user-centered systems design there is not an excessively long initial phase but none the less some up-front planning needs to be done. The first thing on the list was requirements engineering where requirements were identified, analyzed and prioritized. The second step was to analyze and learn to understand the user and the environment which was done through different kinds of work modeling techniques. The resulting work model can be described through different kinds of tools and artifact but as stated by the agile manifesto no documentation for the sake of documentation. The final goal with the user analysis was to produce a design, in this case it was manifested by the system requirements specification and the paper prototype. The former to describe the overall functionality of the system, it was important that it was written down and formalized because of the lack of domain knowledge on my part. The paper prototype was created to give the users an idea of how the GUI would look like to be able to give feedback before I started implementing the system.

* 1. Requirements engineering

The requirements phase was done in iterations in the initial phase but it continued to evolve also during the implementation phase. The requirements were documented thoroughly as both me and the stakeholders lacked domain knowledge of each other’ s field the requirements specification needed to be clear.

During the initial phase the end users and I had a few formal meetings where the main functionality that absolutely needed to be done was written down. These were, data recording and saving which is natural as FlyTracker is essentially a data acquisition program, the second was that there had to be some sort of automatic triggering system as data needed to be recorded at certain timeinterval. And the final one was that there had to be a interactive calibration function, hardcoding calibration values would disable the possibility for the user to ever change the hardware setup. The less important functional requirements as well as interface and non-functional requirements were both set during formal meetings and through informal communication. This way it was easy to optimize the requirements and along the way discover and correct any errors.

During the implementation phase the requirements were revised and reprioritized during the feedback sessions that were held during each sprint. The advantage of working at the same physical location as the stakeholders was once again evident as there was always a possibility of getting immediate feedback on even the smallest thing.

* 1. User analysis and the paper prototype

The paper prototype was made in two iterations before the first actual GUI, by using a combination of the open source tool Pencil and hand made drawings. The prototype provided no interaction with the user but clearly specified what actions were possible and what they lead to (see figure 1). This was then displayed for the end user and they had a chance to give feedback.

The prototype is showing each possible outcome for the different buttons in each window. The program starts out at a welcome screen. From there you can either navigate to the main window directly or start the calibration process, the user interaction is manifested in figure 1 and 2 by arrows leading from each button to the corresponding window.

As I have been working closely with the end-users in the 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 and this is not something the average phd student has a lot of experience with doing. Therefore it was necessary to analyzing how the work would be performed. Which steps needed to be taken to get a fully calibrated system and particularly in which order. Matching the system to the manual work model was important.

Therefore, before the calibration wizard was designed and implemented myself and the main user performed a manual trial calibration to calculate all the necessary values. This also made me aware of what values needed to be calculated to verify the calibration.   
  
The design of the wizard was clear early on, creating a installation wizard to force the user to take the steps in the correct order. But as explained above the order was decided only as the rest of the system was completely implemented.

The main window was design according to design principles, gathering all the controls to the left of the frame and the output to the right (see figure 3). As this type of display is common the selection of design is based on the fact that the user will quickly recognize the structure from other applications.



Figure 1 - The calibration wizard (1/2)

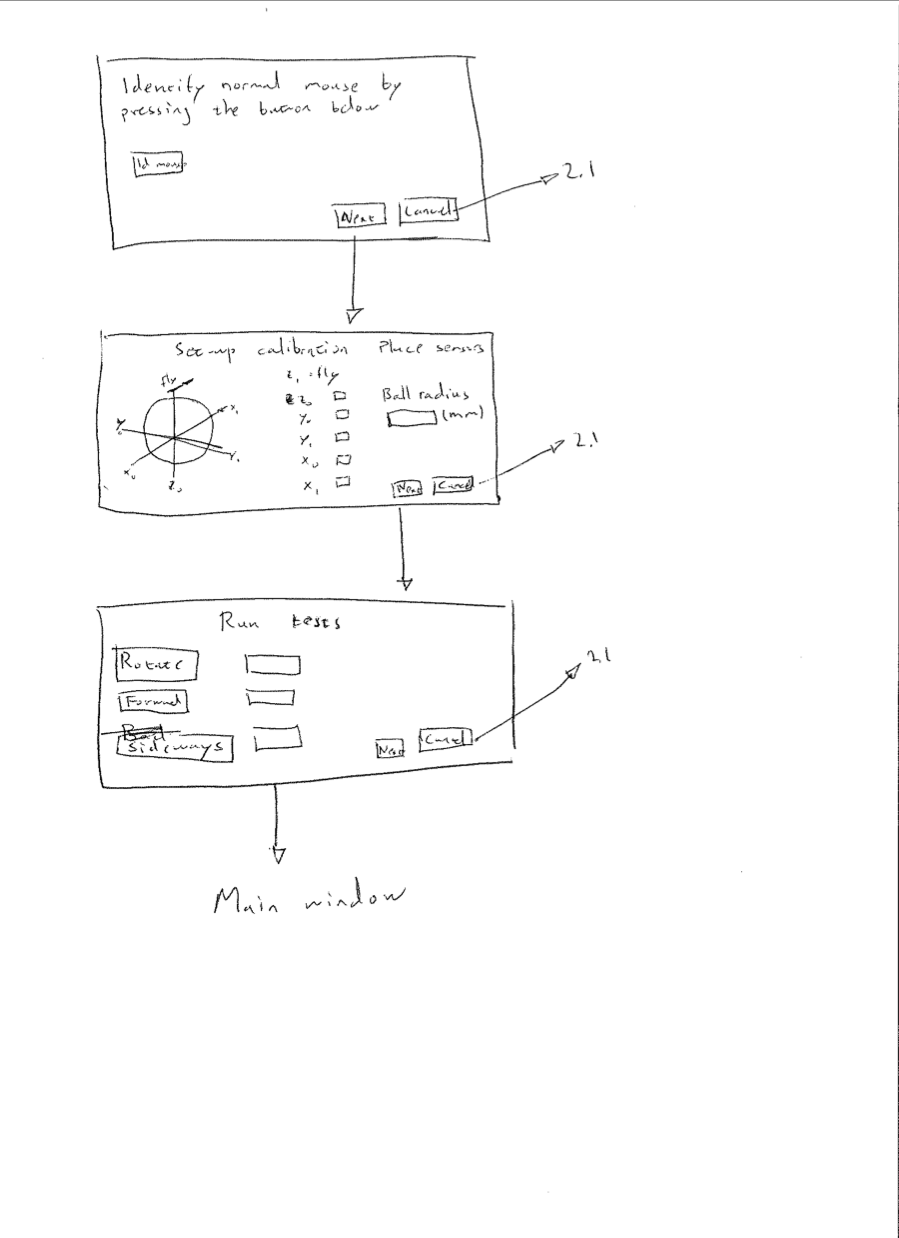


Figure 2 – Calibration wizard (2/2)

.



Figure 3 – The main window final prototype. The structure is clean with the controls to the left and output to the right which means the user will not have to look all over the window for certain functions or graph.

The main GUI is based on common design principles. All controls are placed on the left and the output on the right side. This makes it trivial for the user to locate necessary interaction items. The menu options provide controls that are less important thus unlikely to be used as often.

1. Implementation

FlyTracker is implemented in Matlab and Python. The Matlab module provides the GUI and most of the calculations, while the Python module provides the core of the functionality. FlyTracker is platform dependent and can only be run in Ubuntu Linux. However, changes to the Python module could increase portability. Indeed, it would be fairly easy to replace the Python package as it is weakly coupled with the Matlab module.

* 1. The Python module

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

**6.1.1 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 what mouse is the desktop mouse in the calibration step.

The *MouseHandler* class runs 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 the Python module, but the user interaction is done in the Matlab module, these need to communicate. The *MouseHandler* is never stopped unless a stop command is sent from the Matlab application (except if you run FlyTracker with a timer as a trigger because then the run time are predetermined). The command is sent via a named pipe (which requires Linux) and stops 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).

As data is read it is also saved into a temporary txt file, the data is saved as a json-parsed dictionary object. This object holds two x-values and two-y values, one for each mouse sensor. The final value is a timestamp of that data point. This temporary data file will be opened and read by the Matlab package, which process and saves it in the final output format.

**6.1.2 idMouse.py**

IdMouse is used purely for identifying the mice .

**6.1.3 utilities.py**

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

**6.1.4 Trigger**

The triggering system is a major part of the functionality of FlyTracker. A trigger is basically an on-/off-switch for when to record data. For example, if the user uses a network trigger the recording will not start when the user presses “Run”. Instead pressing run 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 presses “Stop”.

The first triggering option is to use no trigger at all. In this case the user can manually control when FlyTracker is recording by pressing “Run” and “Stop”.

There is also an option to use a timer as a trigger. This m eans that the user can set a time for how long FlyTracker records data, starting when “Run” is pressed.

* 1. The Matlab module

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

**6.2.1 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 (barring the GUI-files) the most important part of the Matlab module. As the name suggests, this function reads the temporary data file and parses them into the correct format. The readdata file also plots the current data continuously, updating the plot every hundred data points.

The raw data that are read are also transformed in the model package. The following two equations are used to calculate virtual translation movement and yaw movement from the sensors into the fly’s coordinate plane.

Calculating the translation vector of length two, containing x- and y-movement:

*w\_m = beta\_.\*[cos(omega),-sin(omega);sin(omega),cos(omega)]\*[y2;y1];*

Calculating yaw rotation where *r* is the radius of the ball:

w\_mz = alpha\_\*t/(r);

*t* is given by:

*t = (x1+x2)/2;*

*if sign(x1) == -1 && sign(x2) == -1*

*t = max(x1,x2);*

*elseif sign(x1)\*sign(x2) == -1*

*if sign(x1) == -1*

*if abs(x1)<x2*

*if abs(x1)<t*

*t = x1;*

*end*

*else*

*if x2<abs(t)*

*t = x2;*

*end*

*end*

*elseif sign(x2) == -1*

*if abs(x2)<x1*

*if abs(x2)<t*

*t = x2;*

*end*

*else*

*if x1<abs(t)*

*t = x1;*

*end*

*end*

*end*

*elseif sign(x1) == 1 && sign(x2) == 1*

*t = min(x1,x2);*

*end*

The last algorithm processes the horizontal component of each mouse sensor to reduce the yaw signal when the fly is not rotating. This algorithm uses the hypothesis that when the values of the horizontal component of each sensor are very different yaw or even having opposite signs the signal should be reduced. As when the fly is trying to rotate these values cannot have opposite signs.

Saving is also done in readdata and is done at the end of each experiment instead of continuously. However, the temporary data file holds all necessary information to recreate the data. This means that if FlyTracker should exit unexpectedly, or be closed before the recording has been stopped, all data from that experiment can be recovered on FlyTracker start-up. The user will then be prompted to either recover the data or delete the temporary data file.

**6.2.2 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 final layout can be seen in fig x and be compared by the initial prototype (figure 3).

An important part of FlyTracker is the calibration,

**6.2.3 Data**

The data are 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 the user uses the network trigger, recordings can also be paused. Each pause then creates a new data block within the same mat-file.

Data are 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 data 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. Future expansion

A possible development of FlyTracker would make the entire experimental set-up to a closed loop system. This means that as the visual stimulus is presented to the fly, the behavior is recorded in FlyTracker, and as a function of the behavior new visual stimulus are calculated. A closed loop experiment would set high demands of low latency on 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 migrate 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.

Migration to Python would also mean better support for increasing the modularity and decreasing coupling. There is definitely work needed to be done in this area to make extensions, that are not even thought of right now, easier to implement.

Another issue that could be changed is the lack of control over the mouse cursor while the fly is moving. Adding a separate mouse cursor would allow the desktop mouse to always have control of it’s own mouse cursor.

1. Results
   1. FlyTracker

FlyTracker is run by opening up Matlab and entering “flytracker” in the command line. This will launch the welcome screen seen in figure 4. The first thing the user needs to chose between is whether or not FlyTracker needs to be calibrated. If a calibration file exists and it corresponds to the current setup the user selects “Load config file” which will load the only existing calibration file (as there is no point in having several).

If there is no calibration file or the existing file is obsolete the user can start a new calibration process by pressing “Start configuration process”.   
  
If the user just want to get to know the program and does not intend to conduct any experiments she can also pick “Not now” which will lead you to the main window but without being able to run a recording. This way the user is protected from doing recordings with incorrect settings.



Figure 4 – Welcome screen. This is the window that is displayed at the start of FlyTracker, providing the user with the options seen above.

Figure 5 is a snapshot of the calibration wizard and arguably the most important step. Calibration of thrust and sideslip is done by having the user rotate the ball to simulate the respective movement and using that data calculate the calibration factor. It is therefore up to the user to make sure the distance the ball is rotated is correctly measured.   
  
A number of trials are necessary to reduce the effect of outliers, an average error is calculated for every run and displayed to the user. At least three runs are mandatory to be able to proceed to the next step of the calibration wizard.



Figure 5 – Calibration of translational movement (thrust and sideslip). In the step of the calibration wizard the user should calibrate thrust and sideslip.

Figure 6 is a screen shot of the main window during a recording (which you can see by the text of the run-button which says “Running..”).

As one can see in the figure only minor changes were made from the earlier paper prototype. The control buttons were moved to the bottom rather than the top as that made it more in line with the natural flow of the task of starting a new recording. First the user selects a “trigger”, then sets the timer if that system is used and finally the recording can be started. The user might lack control over the mouse pointer during experiments as the fly is trying to walk around, therefore both the “Run” and “Stop” buttons can be controlled via the keyboard (switching between them using tab).

The menu holds controls for setting save path, change plotting and network settings for the network trigger.



Figure 6 – Main window. This is the final version of the GUI, as in the prototype the controls are placed to the left and the output to the right. The unit of the first three plots can be changed by the user while the bottom right one always displays a 2D-map. The three different plots that the user can select between are delta position, cumulative position over time and velocity.

In the help menu the user can also find a user manual for guidance to the different parts of the system. The trigger system could cause confusion for the naïve user and the help manual therefore provides a detailed explanation of the functionality.

* 1. Evaluation

The result of the interview presented somehow

1. Analysis and discussion

Diskussion om metod, resultat och vad som bör ändras iom intervjun

1. Conclusion
   1. Issues

During the project there has been quite a few challenges to overcome, some more problematic than others. Deciding on platform for the development was the first issue as while Ubuntu provided a simpler solution for reading mouse data the NIDAQ cards did not work in that environment. NIDAQ cards was our initial solution for triggering. So when deciding to use Ubuntu the trigger became a new issue, quite a few methods were considered but using a network trigger seemed to be the best of the options. And in reality it is actually superior to any other solution in terms of temporal precision and the only drawback is it’s portability. When using a network trigger the visual stimulus software needs to be edited so that it sends the correct signal at the correct moment.

The most difficult challenge to overcome was however the calibration stage. Not only how to actually do the calibration physically but also how to implement the interactive calibration wizard. There was a lot of time spent on solving issues like getting the ball to rotate around a stable axis and being able to measure the distance with high precision. While this was solvable and the error margin became smaller than five percent a new issue followed where FlyTracker actually recorded a yaw rotation signal when the ball was rotated in such a way it would during pure translation.

* 1. Future research

For future developments of FlyTracker it could be useful to do a more thorough evaluation of the software system by letting naïve users try it out. As FlyTracker is currently being shared with other research groups at other universities there is potential for naïve users. A cooperative evaluation could be performed, where a number of naïve users (with similar background) could try it out and give feedback on usability. This way FlyTracker could optimized even further.

Bibliography

Written sources:

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

Lingaard, G “Usability Testing And System Evaluation – A guide for designing useful computer systems”

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

Web -sources:

http://intra.iam.hva.nl/content/1112/verdieping1/research\_for\_design/intro-en-materiaal/RfD-Heuristic-Evaluation.pdf

agile manifest (find link)

**Appendix A:**

**Interview questions**

**General questions**

* How much have you used FlyTracker?
* Whats the best thing about the system?
* What is the worst thing?
* If you could change one thing what would that be?
* Do you feel you got a good grasp of all the functionality
* Do you feel you know where to look for specific functions in FlyTracker?

**The interface**

* Is the interface letting you know what is going on?
* Is the look and feel of the system familiar (is names of functions, buttons and texts understandable, placement of buttons and structures)?

**Functionality**

* Do you feel that you have to be careful not to cause FlyTracker to go into a errenous state?
* Do you feel there are limitations to the functionality and you have to work around it?
* Do you feel the documentation is sufficient?

**User manual**

* Have you used the user manual?

If yes:

* Did it answer your questions?
* Was it easily navigateable?

**Appendix B:**

**System requirement specification**

**1.Non-functional requirements**

* 1. **Usability requirements**

**1.1.1 There is a GUI**

Priority: high

**1.1.2 GUI shows relevant experimental output and let the user select which ones to view**

Priority: Medium

**1.1.3 GUI is clear and follows usability heuristics**

Priority: High

**1.1.4 What graphs to display can be selected by the user**

Priority: Low

**1.1.5 There is a specific interface for the set-up calibration system**

Priority: Low

**1.1.6 Help and documentation is available for the system**

Priority: Medium

**1.1.7 Code is well documented**

Priority: Medium

**1.1.8 System is portable**

Priority: High

* 1. **Performance requirements**

**1.2.1 Delay between mouse input and when the system records data input is low or can be calibrated for, such that it doesn't affect experimental data**

Priority: High

**1.2.2 System performs well over longer experiments**

Priority: Low

**1.2.3 System displays output in real time**

Priority: Medium

* 1. **Other**

**1.3.1 Software is easy to maintain**

Priority: Medium

**1.3.2 Data is saved in standard format**

Priority: High

**1.3.3 Timing of trigger and sensor inputs is correlated**

Priority: Low (?)

**1.3.4 All data files are timestamped at the start of recording**

Priority: High

**1.3.5 System**

Priority: High

**2. Functional requirements**

* 1. **Set-up calibration**

Precondition: Hardware setup including sensor placements relative to center must be done and known.

Trigger: Upon start up, program will alert the user in a clear fashion that calibration is a must for running the software successfully. Options between loading an existing calibration file or creating a new one is presented to the user. If no calibration file exist, the user must calibrate so its not possible to access the main functionality until this has been done.

Flow: After selecting, “create new calibration file” the calibration screen appear, here input fields for ball radius (or diameter) exists. The user can also input the sensor placement relative to each other and to the center of the ball. Finally, the user can run experiments where the ball is moved and the distance moved will be showed in form of sideways, forward and rotation.

Alternate flow: If the old calibration file is loaded, before this is executed a warning pop up appears with information emphasizing the need of correct calibration.

Result: After calibrating the system is fully functional and correctly calibrated.

Priority: High

* 1. **Mouse identification**

Precondition: Data calibration is done and both sensors are connected to the computer.

Trigger: The user selects “calibrate mouse function” in the GUI. If the user haven't identified the mice and tries to start the system the user will be prompted to do the calibration before being able to run the experiment.

Flow: A new window opens where the user can select which two mice to use to read data from for the experiment. When selecting two there will be a output showing which one is selected by activating the sensor corresponding to the given mouse selection on screen.

Alternate flow: Load old calibration from file and info pop-up appears emphasizing the importance of correct settings

Result: Sensor data will now be read from the correct mice and execution of experiment can commence.

Priority: High

* 1. **Data is saved to file**

Precondition: Directory path for where to save data is set.

Trigger: A recording is started.

Flow: Data is saved iteratively to file.

Alternate flow: Any I/O-error or errors related to corrupt data leads to an error-message for the user providing details of what went wrong.

Result: Data is saved into file and user is informed of that fact or an error message describing the problem will be provided for the user

Priority: high

* 1. **Data can be recorded using trigger**

Precondition: Set-up calibration and mouse identification is done

Trigger: “Use trigger” is selected and “start recording” is pressed.

Flow: System will wait for trigger before starting recording.

Alternate flow: -

Result: A recording is saved to file

Priority: high

* 1. **Data can be recorded without trigger**

Precondition: Data and mouse calibration is done.

Trigger: In settings, record without trigger is activated, then requirement is triggered by user pressing “start recording”.

Flow: User press start, recording is ongoing until stopped or timer runs out.

Alternate flow: -

Result: Recording is shown in plots selected by the user.

Priority: Medium

* 1. **User can set trigger threshold**

Sampsamp

* 1. **User can view object movement on a 2D map**

Precondition: A successful recording must have been made

Trigger: User has selected to view data in this particular format

Flow: As the recording starts the 2D-trajectories are drawn as long as the system keeps on recording. Stops but is still visible when the recording is terminated (for whatever reason). At any point the user can clear the map and it will start redrawing from scratch.

Alternate flow: Data is corrupted and makes no sense to present in this format. In this case an error-message will pop up informing the user what is going on.

Result: A 2D map is displayed.

Priority: Low

* 1. **User can view object movement speed, forward and sideways**

Precondition: A successful recording must have been made.

Trigger: The user has selected to view data in this particular format

Flow: As the recording starts the map is drawn as long as it keeps on recording. Stops but is still visible when the recording is terminated (for whatever reason). At any point the user can clear the map and it will start redrawing it from scratch

Alternate flow: Data is corrupted and makes no sense to present in this format. In this case an error-message will pop up informing the user whats going on

Result: Graphs for each speed dimension is shown

Priority: Low

* 1. **User can stop recording using keyboard as input device**

Precondition: A recording must be going on.

Trigger: The user presses key for stopping recording (standard space, can be set in system though)

Flow: When the user sends the stop command the recording stops and the mouse cursor is once again under the control of the user.

Alternate flow: -

Result: Recording stops

Priority: Medium

* 1. **A timer can be used to dictate recording interval**

Precondition: Set-up calibration and mouse identification must be done.

Trigger: User has selected to view data in this particular format.

Flow: As the recording starts the map is drawn as long as it keeps on recording. Stops but is still visible when the recording is terminated (for whatever reason). At any point the user can clear the map and it will start redrawing it from scratch.

Alternate flow: Data is corrupted and makes no sense to present in this format. In this case an error-message will pop up informing the user whats going on.

Result: Graphs for each speed dimension is shown.

Priority: Low

* 1. **There is a python client for network triggering**

Precondition: Network triggering is selected as trigger option

Trigger: -

Flow: -

Alternate flow: -

Result: -

Priority: High

* 1. **U****ser can set port and hostname for trigger server**

Precondition: -

Trigger: -

Flow: -

Alternate flow: -

Result: -

Priority: Low

* 1. **User can start recording**

Precondition: -

Trigger: -

Flow: -

Alternate flow: -

Result: -

Priority: High

1. **Requirements sorted by priority**

**High:**

* There is a GUI which will have necessary user input
* GUI is clear and follows usability heuristics
* Delay between mouse input....
* Data is saved in standard format
* All data files are timestamped
* Set-up calibration
* Mouse identification
* Data can be saved to file by the user
* User can start recording

**Medium**

* GUI shows relevant experimental output....
* Help and documentation is available for the system
* System displays output in real time
* Code is well documented
* Software is easy to maintain
* User can stop recording using keyboard as input device

**Low:**

* There is a data buffer for mouse inputs
* What graphs to display can be selected by the user
* System performs well over longer experiments
* User can set timer on recording via user interface
* User can view object movement speed, forward and sideways
* User can view object movement on a 2D map

1. **User stories**
2. **Sprint planning**

**Sprint 1:**

* Calibration GUI
* Main GUI
* Mouse calibration
* User can start recording
* There is a python client for network triggering
* Implement trigger solution

**Sprint 2:**

* All data is timestamped at certain interval
* Data can be saved
* User can view object movement speed (thrust, sideways, yaw)

**Sprint 3:**

* Setup calibration
* User can set timer on recording

**Sprint 4:**

* User can stop recording using keyboard as input device
* What plots to display can be selected by user in a drop down menu
* User can set port and view hostname for trigger server via the GUI
* User can view object movement on 2D-map
* User manual
* User can choose between data blocks to show
* System is portable
* User can set save-path
* GUI settings are saved at shutdown

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)