UPTEC F10 056
Examensarbete 30 hp
November 2010

Visual Stimulus Development

FlyFly - A user friendly interface for MatLab and the Psychophysics toolbox

Jonas Henriksson

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Visual Stimulus Development - Flyfly: A user friendly interface for MatLab and the Psychophysics toolbox

Jonas Henriksson

Flies use visual cues for a variety of tasks, such as maneuvering through the environment and finding potential mates. Hoverflies, in particular, have very developed eyes and use them to be able to hover mid air and perform fast, elegant movements. The Motion Vision Group, located at the Department of Neuroscience at BMC, Uppsala, studies the motion vision system of the hoverfly brain, using electrophysiology. Experiments are performed by displaying visual stimuli on a screen in front of an immobilized fly, while recording the response from a single neuron with a thin electrode.

Until now, the Motion Vision group has been using the open source program VisionEgg to generate the stimuli. VisionEgg is able to display stimuli at high frame rate and has a large set of useful features such as perspective distortion. It also has a lot of drawbacks that makes it desirable to acquire new software. The main drawbacks include it being hard to learn, use and modify, as well as being unable to generate the stimuli needed for some key experiments.

This master´s thesis describes the development of software more suited to the lab´s needs. This software should be able to generate some of the stimuli that were impossible to do at the moment, as well as being easy to expand and add upon. The frame rate of the displayed stimuli has to be both high and stable in order to perform high precision experiments.

The resulting program is called FlyFly and has been developed iteratively in close cooperation with its end users, ensuring a user friendly end product capable of meeting the lab´s needs. FlyFly is implemented using MatLab and the Psychophysics toolbox with the graphical user interface (GUI) designed with the Guide editor. The GUI is decoupled from the functions drawing the stimuli, making it easy to improve or remove parts altogether. FlyFly is intuitive to use and allows anyone to quickly get started. It allows easy manipulation of series of trials, and supports drawing of multiple objects simultaneously. With the current machine set-up, it displays stimuli at 160 frames per second with few or no dropped frames.

FlyFly is currently being used in the lab and will be so for the foreseeable future.

Handledare: Karin Nordström
Ämnesgranskare: Justin Pearson
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F10 056

Contents

1 Sammanfattning

Hur vi ser är fundamentalt för hur de flesta av oss uppfattar vår tillvaro. Med hjälp av synen kan vi undvika hinder när vi rör oss, uppskatta avstånd och känna igen diverse objekt beroende på färg och form, helt utan att ta i dem!

En sorts synintryck vi tar in och behandlar är olika typer av rörelser. Detta brukar kallas för rörelseseende och används för att uppskatta hur vi rör oss i förhållande till vår omgivning och hur objekt i vår omgivning rör sig i förhållande till oss. Rörelseseendet gör att vi kan fokusera på rörliga objekt, även mot otydliga bakgrunder eller om vi själva rör oss.

Även flugor använder rörelseseende till en rad olika saker, t.ex. för att flyga rakt och för att hitta partners. En flughjärna har ungefär 100 000 nervceller. Jämförelsevis har en människa ungefär 100 miljarder nervceller, alltså en miljon gånger fler. Flugor har också mycket sämre ögon än människor och kan nätt och jämt uppfatta varandra på mer än ett par decimeters avstånd. Trots dessa begränsade förutsättningar kan flugor använda rörelseseendet till att utföra en rad avancerade manövrar, t.ex. undvika rovdjur och jaga varandra i mycket hög fart.

I *The motion vision group* vid neurovetenskapliga instutitionen studeras blomflugans rörelseseende med hjälp av elektrofysiologi. Målet är att förstå hur blomflugan trots enbart ett fåtal nervceller och dåliga ögon ändå kan flyga robust . Dessa forskningsresultat kan både hjälpa oss att förstå hur den mänskliga synen fungerar samt ge idéer till hur man bygger t.ex. robotar som kan röra sig autonomt i okända miljöer.

Elektrofysiologi är studerandet av olika cellers elektriska egenskaper. För att göra mätningar på en enskild nervcell används en tunn elektrod. Elektroden mäter den elektriska spänningen i en punkt vilket betyder att om man placerar den i en nervcell kan man se hur just den cellen reagerar på olika synintryck.

För att placera elektroden i en nervcell skär man ett litet hål i flugans bakhuvud och för in elektroden med mikroskopiskt små rörelser, se figur nedan. Den avlästa spänningen är väldigt olika beroende på om man är i en nervcell eller utanför, vilket gör det väldigt tydligt när man hamnat i en cell. Att få elektroden att hamna rätt är dock väldigt svårt eftersom man i princip letar i blindo.

Experimentuppställning: En fluga placerad framför en skärm som visar animationer. En elektrod mäter spänningen i en nervcell i flugans hjärna.

När elektroden med lite tur hamnat i en nervcell placeras flugan framför en bildskärm som spelar upp särskilda animationer, visuella stimuli. För att undersöka hur neuronen fungerar väljer man olika

typer av stimuli. Vissa nervceller reagerar bara på väldigt små objekt medan andra bara reagerar på skarpa kanter eller rörelser över hela näthinnan. Forskningen går i stora drag ut på att reda ut vilka olika typer av nervceller som finns och vad de reagerar på, och för detta behöver man kunna visa en rad olika stimuli.

För att skapa dessa stimuli används ett speciellt datorprogram. Animationerna måste flyta stabilt och med hög precision för att kunna användas i vetenskapliga experiment. En animation gjord i PowerPoint skulle t.ex. inte kunna användas. Fram tills nu har ett program som heter VisionEgg använts för att göra dessa bilder, men av ett par olika anledningar önskas ett bättre program.

VisionEgg är svårt att lära sig och det krävs en hel del tid och energi för nya användare att komma igång, t.o.m. med enklare animationer. Vissa sorters stimuli går inte att göra överhuvudtaget, vilket såklart begränsar möjligheterna att göra olika experiment. Programmet är dessutom inte byggt för att lätt kunna ändras i, vilket gör det svårt att lägga till de typer av stimuli som saknas.

Målet med det här examensarbetet var att skriva ett nytt program för att generera stimuli med, utan de begränsningar som finns för tillfället. Programmet ska framförallt vara mycket lätt att använda, men det ska också vara möjligt att enkelt bygga ut programmet vid ett senare tillfälle.

Programmet som jag har skrivit heter FlyFly och det används just nu i pågående experiment. FlyFly är lätt att använda även helt utan utbildning och det går mycket fort att göra enkla experiment. Programmet har flera funktioner för att koppla ihop flera stimuli i en sekvens och variera parametrarna mellan varje försök, t.ex. med vilken kontrast bilder visas eller med vilken hastighet ett objekt rör sig. Det finns funktioner för att göra dessa inställningar automatiskt men det går också att ställa in allting helt manuellt. Programmet gör det också enkelt att snabbt växla mellan olika experiment vilket inte alls är möjligt med VisionEgg.

FlyFly kan inte göra alllt som VisionEgg kan, t.ex. kan det inte spela upp filmer eller kalibrera stimulin med avseende på flugans position. Det är däremot lätt att ändra i programmet, även för en användare som inte har så stor erfaranhet av programmering.

Sammanfattningsvis gör FlyFly det enkelt att sätta ihop och spela upp stimuli. Det är också lätt att lägga till nya stimuli som inte finns ännu. FlyFly används dagligen i *The motion vision group* sedan början av juni 2010. Det kommer också fortsätta att utvecklas och användas i framtiden.

2 Project Description

The aim of this master thesis was to develop the basic structure of a new program able to generate visual stimuli for use in motion vision research. The program should be able to construct and display stimuli with high and stable frame rate. Furthermore, the program should be easy to learn and use for scientists as well as students.

Abbreviations

STMD	- small target motion detector
FPS	- frames per second
GUI	- graphical user interface
OS	- operating system
PTB-3	- psychophysics toolbox version 3

3 Background

3.1 Flies and visual behavior

Across the globe, about 125 000 different species of flies have been described (Yeates and Wiegmann 1999). To put that into context, there are about 1 250 000 described animal species in total, meaning that one out of ten of these species is a fly! Why are there so many flies? How come that such a tiny creature with such a short lifespan has spread all over the world and evolved into all these thousands of different species?

In some ways the fly is a developed species. They have amazing aerodynamic capabilities with a turning velocity of 3000 degrees/second and a maximum speed at close to 25 km/h. They have huge eyes and can see in all directions that are not directly blocked by their own body (Land and Collett 1974; Wagner 1986).

Even with the relative huge eyes, flies still have spatially poor eyesight. But even with this limited resolution, flies are able to use visual cues for a large number of different behaviors, such as flight path correction, landing, object detection and target pursuit. Equivalent visual processing by digital means is very hard, or even impossible, with modern computers - even with the aid of computational power far beyond that of the tiny insect brain, and a camera with much higher resolution then the insect eye.

Simply put, flies can perform amazing maneuvers even though they are equipped with crude sensors. What if a human robot could do the same?

3.2 Fly biology

There are many different classes of neurons, with different tasks. The neurons dealing with motion vision can broadly be categorized into two groups, wide-field and target neurons. An example of a target neuron is the small target motion detector (STMD) class, which plays an important role in detecting small targets. An example of a wide-field neuron class is the horizontal system (HS), which detects wide-field motion along the horizontal axis.

Motion vision neurons can receive information both from the retina and from other neurons, and may in turn send information to a number of different neurons. Single neurons receive input from a selected part of the retina, which we refer to as the neurons´ receptive field.

The fly compound eyes have a resolution much lower than that of the human eye. The human eye has an angular resolution of about 0.017°, which corresponds to being able to resolve the width of a hair strand. Fly eyes typically have an angular resolution of 1-2°, which is comparable to the width of your thumb at arm's length (Land and Eckert, 1985). For a rough approximation of the difference between human and fly vision, see Figure 1.

Flies are able to resolve high temporal frequencies, with a corner frequency of about 125 Hz (Tatler, 2000). The human one is about 50 Hz, meaning that animations made for human vision will appear very jerky through the eyes of a fly.

Figure 1: The lower image is an approximation of how a fly would perceive the landscape in the upper image.

3.3 Fly studies

Studies on insect motion vision have been conducted since the 1950´s by researchers in diverse fields such as zoology, engineering and physics. The actual studies can be broadly categorized into a few different areas such as behavioral biology, physiology, modeling and biomechanics. It is quite uncommon for researchers to be specialized in more than one field, though the fields do overlap each other on many occasions. Progress in one field is likely to affect other fields. For example, from behavioral studies we know that male flies are much better than females at pursuing targets. Morphological studies show a number of gender specific differences (sexual dimorphism) in the visual system, such as the neuron classes known as male and female lobula giants (MLG and FLG). These two facts together indicate that the MLG has an important role in target pursuit, which might inspire further studies.

3.3.1 Electrophysiology

One way to investigate the motion vision system is by electrophysiology, the study of the electrical properties of cells. This is done by making a small cut in the back of the fly's head, and inserting a thin electrode into the part of the fly brain where motion vision is processed, thus making it possible to measure and record the electrical activity of a neuron. Depending on the equipment, the electrode might be placed inside a neuron (intracellular) or outside (extracellular). Intracellular readings are more accurate and are only affected by one cell, while extracellular readings are easier to make but record from a group of nearby cells.

A possible set-up for an experiment is to place the fly in front of some display device, such as a monitor, with the data acquisition equipment mounted, see Figure 2. By displaying an animation, a stimulus, in front of the fly, it is possible to study how its motion vision neurons react to different kinds of visual input.

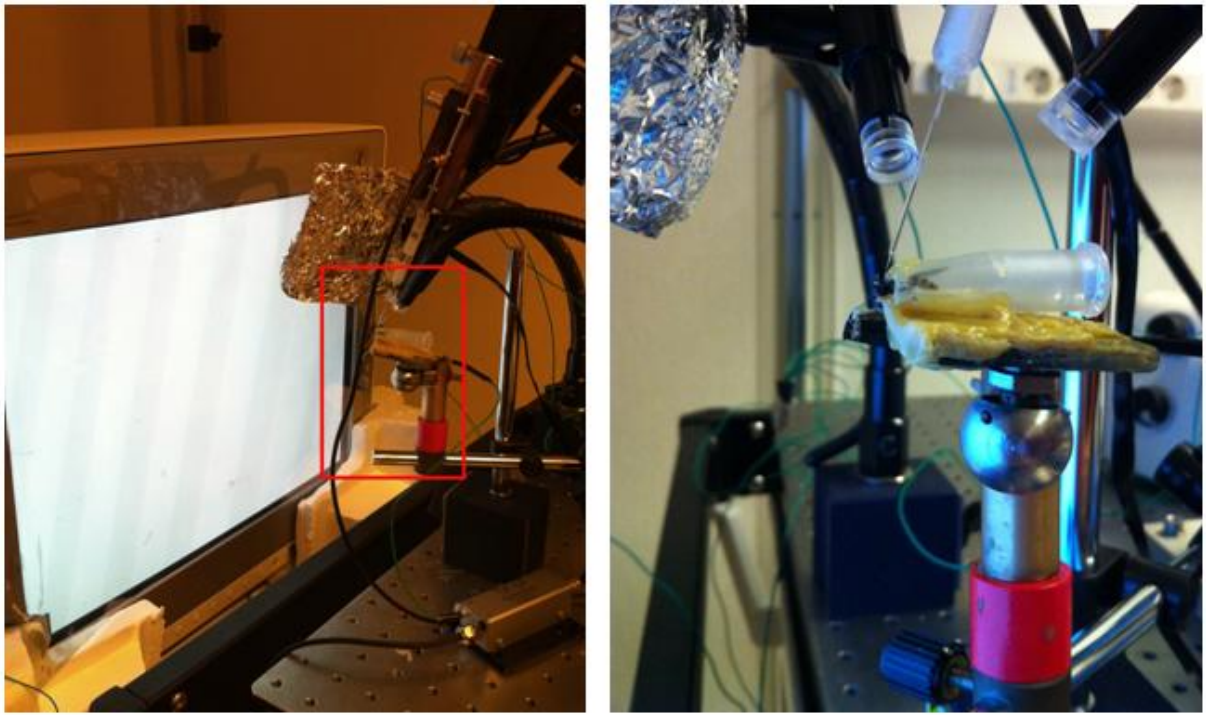

Figure 2: Lab set up. Left image shows a fly positioned in front of the monitor. The right image shows a magnification (marked in red in the left image) of the immobilized fly, with the electrode in position.

A block view of a set-up of this kind can be seen in Figure 3. In this figure, 'Other input' refers to none-visual input, such as sound and smell, while 'Response' is the behavioral output of the fly. They are grayed out in the image because only visual input directly affects the neurons in the visual system, and the behavioral response is minimal due to the fly being immobilized.

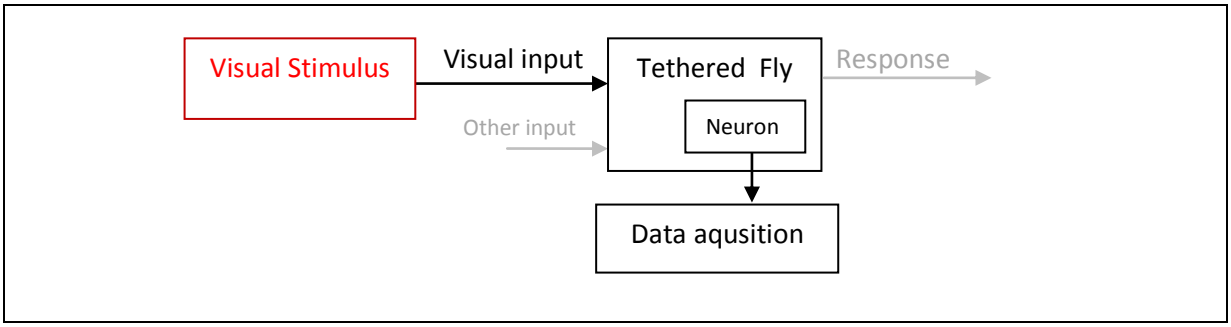

Figure 3: Block view of the lab set-up from Figure 2. This project focuses on the red part.

3.3.2 Significance

Understanding the motion vision system of the fly can help us understand other visual systems, such as the human one. Doing electrophysiology currently requires the back of the head to be cut open, which makes it impractical to do the experiments on humans.

The results obtained can also be used to design robust algorithms for use in artificial objects. At present there are ongoing projects to develop both flying and walking autonomous robots, used for example in reconnaissance, surveillance and rescue. Other appliances inspired by insect technology include sensors for panoramic vision, attitude stabilization, course control and orientation (see Srinivasan, Wootton, Whitten, 2004).

3.4 Visual Stimuli

In electrophysiology, the most important component of the experimental set-up, apart from the equipment needed to collect the signals from the investigated neurons, is the visual stimulus.

3.4.1 Types

One can distinguish between artificial and naturalistic stimuli. Artificial stimuli are stimuli without direct natural counterpart, for example a moving bar or a sinusoidal grating. These types of stimuli can be useful when studying basic properties of a neuron, for example if it responds to small or large targets or where it has its receptive field.

Naturalistic stimuli are an approximation of stimuli that could be experienced in nature. It is of course desirable to make the stimuli as close to nature as possible. True natural stimuli are not possible at the moment because of technical constraints, such as the fly being immobilized and the display device not covering the entire visual field.

3.4.2 History of display devices

Earlier studies in motion vision used mechanical display devices to display their stimuli. One such example is the mechanical spinning drum experiment, were the test subject sits in the middle of a spinning drum covered with sinusoidal stripes or other patterns, see Figure 4.

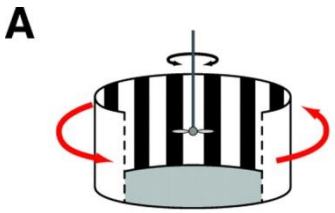

Figure 4: Spinning drum experiment. The fly is tethered in the middle with a striped drum moving around it (Fry, Rohrseitz, Straw, Dickinson, 2009).

Mechanical stimuli have some advantages over digital ones: There is no need to worry about flickering or temporal resolution, and there is no risk of ruining an experiment because of a computer crash. There are of course also many disadvantages, such as a limited number of stimuli possible to construct, slow switching between stimuli and difficulties in producing smooth and exact movements. Because of these disadvantages mechanical stimuli are not very common today - as an example, none of the papers published the last year, in the field of fly motion vision, used mechanical stimuli.

During the 1980s and the 1990s, a common way to generate stimuli was by using the Tectronix 608 display, a cathode ray tube intended to be used as an oscilloscope, combined with specialized function generator, such as the Innisfree Picasso device. This set-up was found to be easy to use and capable of easily generating a large number of different stimuli. There are of course some limitations with this set-up. The display device is quite small, with a diameter of about 12 cm, which limits the maximum coverage of the fly´s visual field. The function generator is also expensive and limits the possible stimuli to various kinds of gratings (see Straw, 2008).

Using a computer it is possible to generate any arbitrary stimuli, though with the introduction of a set of new limitations, such as limited frame rate and brightness. The first computer generated stimuli

were made in the early 90s, using pre-rendered animations in order to achieve a satisfactory frame rate (see Straw, 2008).

The evolution of video cards in recent years has been massive and since the 2000s it is possible to render stimuli in real-time, which would have been impossible just a few years earlier. The video game industry continues to push the boundaries, and modern hardware of today will probably be considered ready for retirement in less than 10 years.

The new and powerful graphic cards are not only able to render stimuli in real-time, they are also easily available and cheap compared to specialized equipment.

3.4.3 Existing software

Many vision labs have engineers working full time developing their own specialized software. This, combined with the fact that the field of insect motion vision is not that big, means that no commercially available tools to render stimuli are available. However, there are a small number of open-source software able to generate the needed visual stimuli available. The most common and well known is VisionEgg by Andrew Straw (Straw, 2008), which is currently used in the Motion Vision Group. While VisionEgg has been used for a long time and has a large number of useful features, it has some fundamental flaws that make it suboptimal in some cases.

First of all, installing VisionEgg requires downloading and installing about six different packages in the correct order. In addition, you need to compile the source code in python, a challenging task if you are inexperienced with the language.

Second, learning VisionEgg is a time consuming project. Due to its nature as an open source program originally developed for internal use, help files and documentation are sparse and lacking.

Third, being open source it is possible to modify VisionEgg to your own needs and experiments. However, the program is not written with these kinds of extensions in mind and it is therefore not very easy for an inexperienced programmer to understand and modify the code. Also, VisionEgg is strictly object oriented and written in Python, a concept and language unfamiliar to most biology students, who will be the ones using the program.

Another software available, is PsychoPy by Jonathan Peirce (Peirce, 2006). While it tries to solve some of the problems of VisionEgg, such as the strict object orientation, it is also written in python, which makes it hard to make small changes and add new modules.

3.5 The Motion Vision Group, Uppsala

The Motion Vision Group in the Unit of Physiology, Department of Neuroscience, is a small and newly founded research group. The Motion Vision Group mainly deals with electrophysiology, as described in 3.3.1, but also houses morphological and behavioral studies. At the moment there are three main projects going, of varying length and with different focus.

The one most closely related to this thesis is the project of PhD student Frank Lee, *Small Target Detection in Male Hoverflies*, see 4.2.1. Lee is investigating the different subclasses of STMDs and the connections between them. This is basically done by mapping the receptive field of the neuron and measuring how fast different neurons respond to the sudden appearance of a small target.

In the future, many projects are likely to be performed by students staying in the group for a limited time. Because of these short time spans, it is desirable that the software used for generating the visual stimuli is easy and intuitive to learn and use in order to let the students get going with the experiments as fast as possible.

The students doing these projects have a background in biology, with little to no experience in programming. Any experience will probably be with MatLab, which is a standard tool for use in data analysis.

3.6 Problem formulation

Because of the issues with existing software, it would be beneficial to develop a new set of software tools adapted to the needs of the lab. The goal of this thesis work is to lay the foundation of such a toolset. Because of the project´s time constraints, it is not possible to design a set-up that is able to display any arbitrary stimuli, while still being easy to learn and use. Instead the aim is to make a framework which is able to generate a few key experiments, which are either very time consuming at the moment or not possible to generate at all, while also being easy to modify and add on at a later stage.

4 Problem analysis

4.1 Overview

There are three main goals of the actual stimulus development. In order of falling importance they are *stability*, *usability* and *modifiability*.

The *stability* criteria can be broken down into a number of hard specifications that needs to be fulfilled in order for the program to be usable at all in the scientific experiments conducted in the lab. *Usability* refers to the fact that the program should not only be able to generate the required stimuli, but also do it in a user friendly environment. *Modifiability* means that it should be possible to modify the program after this project is finished.

An overview of the specifications formulated at the start of the project is found in Table 1. Each of the specifications is explained and analyzed in more detail between section 4.2 and 4.9.

Table 1: Software specifications

Hard Specification	**Meaning**
Stimuli	The program should be able to generate a number of different stimuli.
Frame rate	The program should display all stimuli at the maximum possible frame rate of the monitor (160 Hz), without dropping any frames.
Precision	The timing has to be precise and reliable.
Trigger	The program should display a trigger when the experiment starts in order to synchronize the experiment with the data logging.
Sequencer	The program should allow setting up a series of experiments, in which the input parameters vary.
Saving parameters	The parameters used in an experiment should be saved automatically in an easily accessible format.
Soft Specifications	**Meaning**
Usability	The user should not be forced to read a long manual in order to set up simple experiments.
Modifiability	The user should be able to make small changes to the code without causing major damage. A programmer should be able to add all new experiments without modifying the framework.

4.2 Stimuli

The program should be able to generate and display all stimuli necessary for the *Small Target Detection in Male Hoverflies* project. This project, explained in detail in 4.2.1, will be the main evaluation of the program code.

In addition to this, it should also be possible to generate some general stimuli that can be used in a multitude of unspecified future projects.

4.2.1 Small target detection in male hoverflies

One goal of this project is to measure the delay times of different STMD classes. When a neuron is first found, a series of different experiments are done in order to classify the neuron and then, if it is the correct class, measure the delay time. The following procedure is used:

First, a wide-field sinusoidal grating is displayed. If the neuron reacts to this, it is a wide-field neuron, not a small target neuron, and the electrode can be moved to another neuron. If it does not respond a new stimulus is displayed, this time with a small mouse controlled black dot. If the target responds to this it could be either a STMD or some other type of feature neuron, such as a bar cell.

To see what kind of STMD it is, if it is, its receptive field must be mapped. The receptive field of the neuron is found by moving a small target straight across the screen, horizontally and vertically, at varying positions.

With the receptive field mapped, it is possible to identify the neuron as an STMD, by performing a size tuning. STMDs respond optimally to targets of about 3°. Therefore, a test with an increasingly high bar, needs to be done. The bars are moved through the center of the neuron´s receptive field. If the neuronal response continues to increase after 3° then it is not an STMD - if it peaks at 3°, it is.

With the STMD identified it is possible to run delay experiments. In these experiments, a small target is drawn straight into the middle of the receptive field, and then moved off screen. The delay time is the time it takes until the neuron responds to this target. The delay times can then be analyzed and, hopefully, give some new information on the different STMD classes.

These four different experiments can easily be broken down to two different stimuli: One capable of drawing small square targets moving across the screen and one capable of displaying a sinusoidal grating at an arbitrary angle. For each of them the user should be able to set a number of different parameters, such as target height and width and the wavelength of the grating.

4.2.2 Additional stimuli

In addition to the features needed for the *Small Target Detection in Male Hoverflies* project, a few other features have been specified in order to make the program able to generate a range of stimuli for future projects.

First of all, it should be possible to set a background image to the target stimuli. The background should be able to roll, i.e. edges disappearing on one side should reappear on the other, with a specified speed and direction. This is useful in many different experiments, for example target detection in clutter.

Apart from the moving background of the target stimulus a different stimulus with just the background is wanted. In this stimulus, the user should be able to set some additional parameters, such as background rotation and contrast.

Finally, both the target stimulus and the grating stimulus should be able to draw several objects simultaneously. For the target stimulus this refers to the actual moving target, while in the case of the grating stimulus it refers to the patch with the grating. Each target and patch should be independent, and have its own set of parameters possible to adjust from the user interface.

4.2.3 Pre and post stimulus

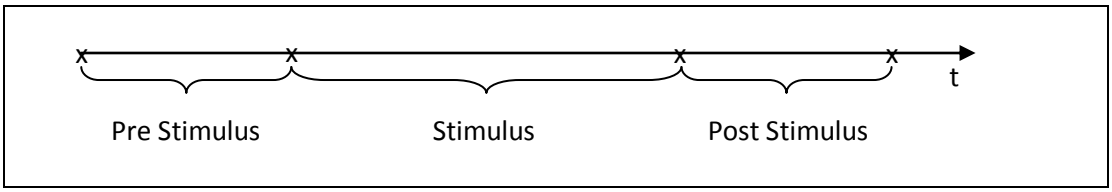

Figure 5: Pre and post stimulus is a period of time before and after the stimulus, which allows the neuron to cool down and simplifies the correlation between input and response.

The basic layout, as seen in Figure 5, will be the same for all experiments. When the stimulus starts, there will be a short time in the beginning when nothing except the trigger is drawn to the screen. The reason for this is twofold: It gives room between the start of the recording and the "interesting" part of the stimulus, which facilitate the later analyzes of the data compared to base-line. It also gives the neuron some time to "cool down" from any previous adaptations to visual stimuli. For the same reason, we also have a post stimulus after the stimuli finished playing.

4.3 Frame Rate

Because flies have such a high temporal resolution, it is necessary to display the stimuli at a much higher rate than would be enough for a human test subject. Estimates put the corner frequency of flies' temporal resolution at about 125 Hz (Tatler, 2000), meaning that stimuli displayed with this frequency will appear smooth moving. Lower frequencies will make motion appear jerky, which in turn may affect the experiments.

The necessary frame rate is a highly debated issue in the field. Some labs do experiments with stimuli from 120 Hz LCD monitor, while other labs only use 200 Hz CRT monitors or even faster LED arenas. When doing neuronal recordings, these frequencies show up as large spikes when plotting the power spectrum of the recorded signal. Biological systems are rarely linear, which means it is not possible to simply subtract the power spectrum of the monitor response. Because of this reason, it is preferable to use a display device with an update frequency over the corner frequency of the insect.

4.3.1 Set-up

The current lab setup uses a 160 Hz CRT monitor. Finding a monitor with a high update frequency is not easy, since CRT monitors are no longer in production and LCD monitors typically run at a maximum of 120 Hz. There are also other common problems with LCD monitors, such as problems with backlight bleeding and limited viewing angles.

While it is important to keep a high and stable frame rate, it is also important to use a general set-up. For example, using a proper real time operating system would be very advantageous for the frame rate, but a huge disadvantage for the lab in terms of cross compatibility and education of lab staff. It would also make it much harder to use the program in another lab or change the hardware of the set-up.

Before looking at the source code of the program, there are a number of things that can be done in order to improve the general performance of the operating system. The most important thing is to make sure the operating system used is as clean as possible with a minimum of programs running. If possible all automatic updates, virus scans and similar should be turned off. It is also beneficial to disconnect the computer from the network and to unplug external devices. Depending on the OS,

there might be a number of performance enhancing settings to do, such as turning off unnecessary graphical bling-bling.

4.4 Precision

There are some inherent precision problems with displaying pixel based stimuli, most notable for slow and fast moving objects.

Slow movements of an object with hard edges are limited by the minimum distance of one pixel. Due to the limited spatial resolution of flies this problem can be largely ignored when using a monitor type display. It is worth noting that many labs use LED arenas to generate their stimuli. In this case, the minimum movement will be a single LED light, which is about 2-4° in diameter.

Fast movements are problematic because of the limited number of frames per second a monitor can display. An animation is just a sequence of images with small movements between them. When played rapidly, these small changes will make objects appear to move smoothly. If the change between each image is too big then the movement will appear jerky.

The effects of fast moving objects can be reduced by the use of motion blur. Motion blur can be achieved by rendering an animation with a higher frequency, and then averaging multiple images together - this is often referred to as temporal anti-aliasing. Photographs of moving objects, taking with a normal camera, will naturally contain motion blur due to the exposure time, as seen in Figure 6.

Figure 6: Natural motion blur occurs in photographs due to the exposure time. Photograph taken from another moving train.

4.5 Trigger

In order to synchronize the data logging with the stimuli, a part of the screen is used as a trigger. The trigger is simply an area that changes color from black to white whenever the stimulus starts running. A photodiode placed in front of this area, reacts to the change in brightness and triggers the data collector. This may seem to be a crude solution at a first glance, but is actually the most fool proof way of implementing the logging. A graphical trigger will always appear on the same frame as the

stimuli starts, even if some frames are dropped. If another type of trigger is used, for example an electric output signal from a DAQ card, there is always the risk of jitter between the call to the DAQ card and the actual drawing of the stimulus to the screen.

With a sample rate of 10 kHz, the maximum delay between frame onset and the data logging will be less than 0.1ms, which is significantly smaller than the neuronal response times.

4.6 Sequencer

When doing experiments, it is common to vary one or more parameters in a series of different trials. For example, when investigating how small targets a neuron reacts to, each experiment could use a smaller size until no reaction is provoked. Some experiments might use up to dozens of different trials, varying one or more parameters. In order to achieve a smooth flowing work process, it is important to be able to set up these series of trials in a fast and intuitive way.

Other software typically use two different ways to handle the setting of the sequencer, either a table based view or some form of string input, see Figure 7 and Figure 8. The former gives the advantage of letting the user directly see which values will be used in the different trials, and allows easy change of values at arbitrary points. The latter on the hand might be preferable if handling large sets of trials, which might end up looking cluttered in a table view.

	Trial 1	Trial 2	Trial 3
Height	5	10	15
Width	5	5	5
Speed	20	20	20

Figure 7: Sequencer in table form

```
{ Height:   5, 15, 5, 'linear' }
{ Width:    5   }
{ Speed:   20  }
```

Figure 8: Sequencer in string form

4.7 Saving parameters

After each experiment the internal parameters need to be saved for later analysis. It is important that this is done automatically and systematically: when doing the data analysis there should be no doubt as to which set of saved parameters corresponds to which set of gathered data.

4.8 Usability

The user friendliness of the program is mostly affected by the graphical user interface (GUI).

The GUI can be designed with two different intentions in mind, being intuitive and being complex. An intuitive system is desirable for obvious reasons: many people should be able to use the program without extensive training. A complex GUI on the other hand, can possibly handle lots of different types of experiments with different options, and is easier to work with in case extensions are added to the program. Good interfaces can be both intuitive and complex, though complex features tend to make the interface crowded and less easy to understand.

One important aspect of the GUI is how easy and fast it is to switch between different experiments and get the stimuli running. Mining for neurons is a slow job and it might take hours or even days just to get a good signal from a single neuron. When a neuron is found it is not really possible to tell how long it will last - if unlucky the signal might be gone after just a few minutes. Because of this reason, it is helpful if the GUI of the program makes it easy to quickly switch between different set-ups, adjust settings and then launch the intended stimuli, in order to waste as little time as possible.

4.9 Modularity

Because of the time limit on this project it is not possible to make a complete set of modules able to display any arbitrary stimuli. However, it is possible, and also intended, to make it easy to modify and add to the framework of the program

Using a clear and logical program structure makes it easy to understand how the program works and how all the different parts interact with each other. This can be improved further with clear and well commented code, as well as a proper documentation.

Functions should have a well defined role and not make anything unexpected. If the function has access to any global variables, it has to be clear. Ideally, global variables should be kept at a minimum or not be used at all.

5 Method and theory

5.1 Development process

This project has been developed inspired by the agile method *Scrum*. Scrum uses a number of different roles and activities as seen in Table 2.

Table 2: Key features of the agile method Scrum.

Role	**Description**
Product Owner	The costumer, adds features to the product backlog
Scrum Master	Keeps the team focused on their task
Team	Develops the product, typically 5-9 people
Activity	**Description**
Product Backlog	A list of new features requested by the Product owner
Sprint Backlog	A list of new features to be implemented in the coming sprint
Sprint	A time interval, typically 1-4 weeks, in which a set of new features is implemented
Daily Scrum	A short meeting in which the team review what they have done since yesterday, what they will do today and if there are any obstacles hindering the team
Sprint Planning	A longer meeting held at the beginning of each sprint in which the team plans the work of the coming sprint
Sprint Review	A meeting where the features implemented in the latest sprint is presented to the product owner
Sprint Retrospective	A meeting after each sprint reflecting on what went well and what can be improved to the next sprint.

Obviously, a lot of these features are meant to optimize the performance of a team, and this project has been developed by a single person. This means that the roles of Scrum Master and Team are the same and the Daily Scrum focuses on the goals of the day, and not the coordination of the team. Also, the longer meetings held before and after each sprint have been omitted in favor of a less strict approach. The features used in this project can be seen in Table 3.

Table 3: Key features used in this project

Role	**Person**
Product Owner	Karin Nordström
Scrum Master	Jonas Henriksson
Main users	Frank Lee, Karin Nordström
Activity	**Description**
Product Backlog	List of wanted features
Sprint Backlog	High priority features to implement
Sprint	2-3 weeks of work, no hard deadlines.
Daily Scrum	A short review of what to this day. Combined with continuous feedback from the users on what is working and what needs improvement.
Sprint Review	A meeting where the features, implemented in the latest sprint, are presented and additions for the next sprint discussed.

5.1 MatLab and Guide

Because of the reasons discussed in the background section, it was decided to write the program in MatLab. MatLab by itself has poor functions for drawing graphics, which would not be suitable at all for this project. Fortunately there are several MatLab toolboxes available that come with improved graphics handling. One of them, the Psychophysics toolbox, is even designed with the intent to be used in visual research, which seems to be the perfect base for this project.

The user interface was designed in Guide, a graphical editor supplied with MatLab, see Figure 9. Guide makes it easy to quickly add new features to the GUI, as well as changing existing ones. The biggest drawback is the lack of native support for tabs and a slight lack of control due to automatically generated code segments.

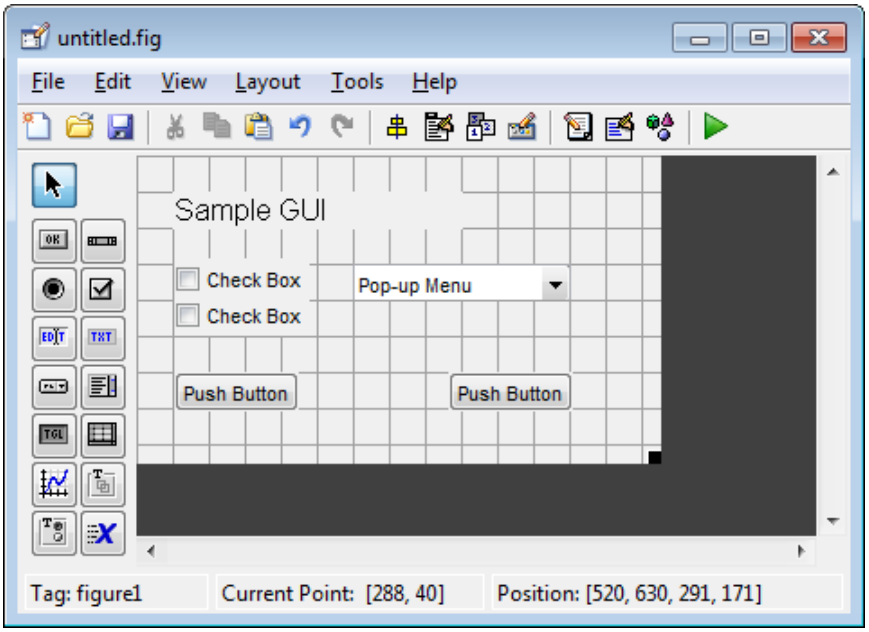

Figure 9: The Guide editor

5.2 Psychophysics toolbox

The Psychophysics toolbox (PTB-3) is a MatLab toolbox designed to let its user create different kinds of stimuli to be used in visual research. The core routines are built as subroutines from the *VideoToolbox* (Pelli, 1997). Key features of PTB-3 are access to display frame buffer, synchronization with the vertical retrace, and millisecond timing (Brainard, 1997). The implementations of these routines are handled by a set of MatLab Extension files (MEX). The *MEX* files are written in C and can be called from MatLab as normal functions, though with much enhanced performance.

There are several advantages of utilizing PTB-3. It has been in use for several years and is well tried out and has a large community. The core routines allow smooth communication with the underlying hardware, allowing more time to be spent on actual experiments.

Note that even though MatLab with PTB-3 is a great way to quickly set up experiments for an experienced programmer, it does not ship with any user interface or pre made stimuli suited to the lab´s needs. In short, PTB-3 by itself is by no means able to solve the problems of VisionEgg.

5.2.1 Layering and OpenGL

The concept of layering is one of the most important in computer science. Instead of writing programs in low level machine language, which is very different from our language used in everyday life, layering allows the use of high level languages which closer resembles the way humans communicate.

For example, when displaying an image on the monitor, first the CPU sends information to the graphic card about the image, such as where to draw it and where in memory the texture is located. The GPU then computes the actual pixels and sends them to the monitor.

The actual communication between the graphic card and the CPU is handled by OpenGL. OpenGL hides all the nasty hardware details, such as the data transmission across the computer bus, which makes it much easier to use the graphic card. The layers most relevant to this project are shown in Figure 10.

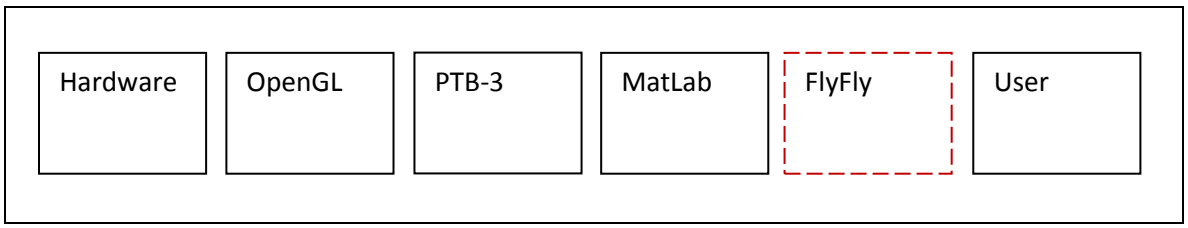

Figure 10: Layers between user and hardware. Aim of this project dashed in red

5.2.2 Double buffer and Vsync

OpenGL utilizes a double buffer system. This means that all rendering is performed to a back frame buffer while the content of the front buffer is drawn to the screen. When the rendering is complete, the two buffers are swapped with a high speed command, commonly called a *flip*. This method means that no partially completed scenes will be drawn to the screen.

If the flip is not synchronized with the vertical retrace of the monitor, the new scene will probably be flipped at the same time as the old one is being drawn. If the scene features moving objects, this will lead to something called a tearing artifact, see Figure 11. Most graphics card includes a feature called *VSync mode* which ensures that the flip takes place between two retraces instead of during one.

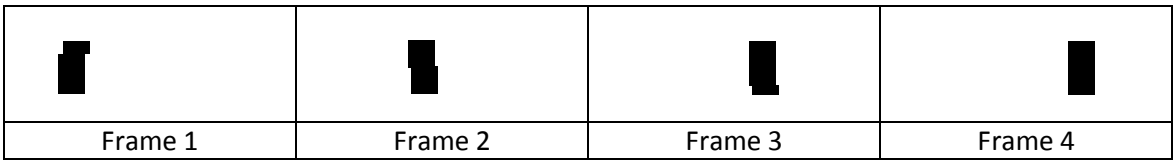

Figure 11: Tearing artifact on a moving bar

5.3 Evaluation of set-up

A number of different tests and experiments were conducted in order to maximize the frame rate of the system and evaluate various features of MatLab and Psychophysics toolbox. The tests were performed on two different machines, a Windows 7 set-up with a cheap built-in graphics card, and the better OSX set-up used in the actual experiments. Their respective specifications can be seen in Table 4.

Table 4: Available machine set-ups

	Machine 1	Machine 2
Computer	HP Compaq 8100 Elite SFF PC	Mac Pro 4.1
OS	Windows 7, 64 bit	OSX 10.5.8, 64 bit
MatLab	R2009b 32 bit	R2009b 32 bit
RAM	16,0 GB	3,0 GB 1066 MHz DDR3
Graphic Card	NVIDIA QuadroNVS 290, 256 Mb	ATI Radeon HD4870, 512 Mb
Processor	Intel Core i7 860 @2.8 GHz	Intel Core 2 Quad Q6700 @2.66 GHz

6 Results

6.1 FlyFly overview

The result of this project is a program named FlyFly, designed from the start with the aim of making it easy to modify and add new modules. A block view of the program structure can be seen in Figure 12.

Each experiment is written as a function in its own *.m* file which can be called with the experiment parameters and settings as input. Each experiment is also connected to a graphical user interface. Many experiments are similar in their layout and can therefore use the same GUI with just the current parameters updated. Individual settings are accessible in a separate settings GUI which is individual for each different experiment.

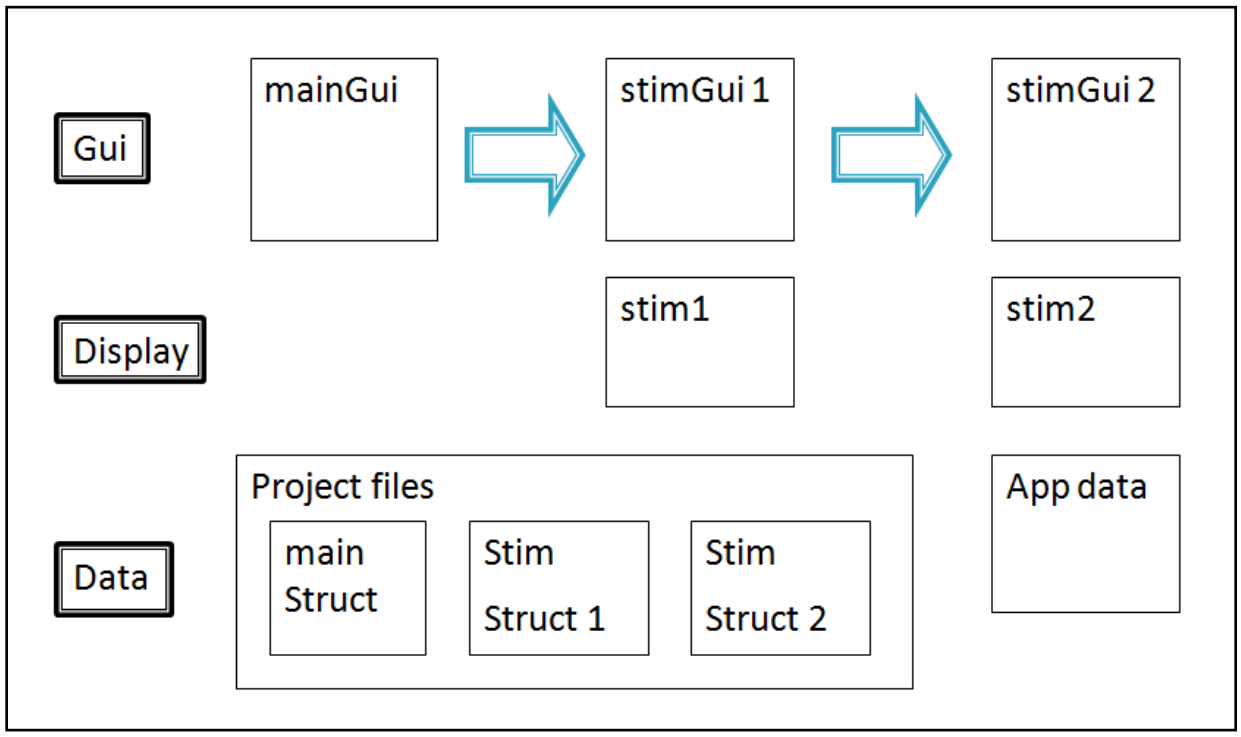

Figure 12: FlyFly overview. The program can be separated into three distinct parts: The Gui seen by the user, the display functions drawing the animations, and the data set holding the experiment parameters and program settings.

6.2 Frame Rate

Figure 13 shows an approximation of what happens during each frame. First of all, any computations needed are made and the drawing to the back buffer performed. After that we have a small random wait time, jitter, where the OS distributes CPU time to other running processes. In reality we do not know where in the frame this jitter occurs, though the size of it can be estimated from old statistics.

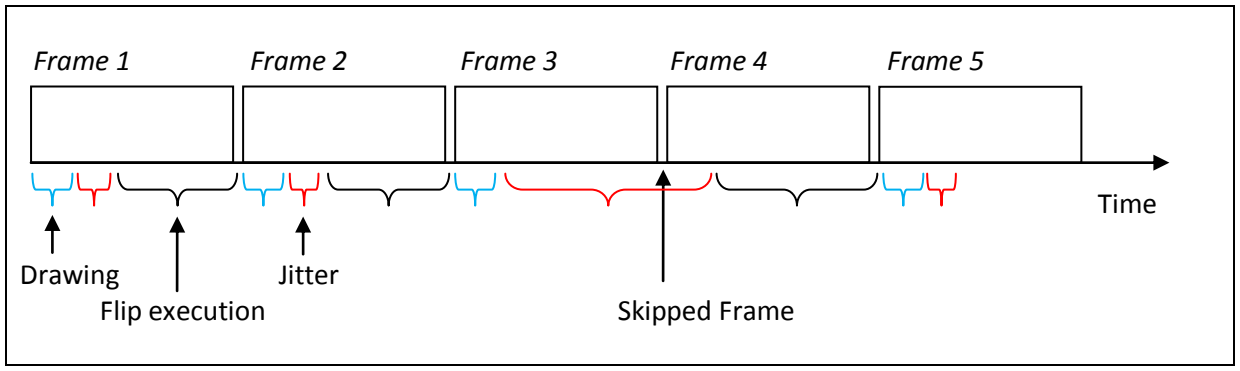

Figure 13: A skipped frame occurs when the program is unable to update the back buffer before the flip call. This may be due to advanced drawing operations and/or external jitter.

6.2.1 Skipped frames

In order to accurately measure and evaluate the frame rate, it is important to know exactly if and how many frames have been skipped in an experiment. Each call to the flip function returns a parameter, 'missed', which, according to the documentation, is set to a positive value if a frame is estimated to have been skipped.

To find out how this value is calculated, some simple tests were run on Machine 1 using a stripped down version of the animation loop. A pause of 20 ms was inserted in frame 30, 60 and 90, which was enough to skip one frame at each occasion. The execution time of each frame was measured and plotted against the 'missed' value, see Figure 14. From this graph it is clear that the 'missed' value, M, is

$$M = F_{exec} - ifi * 1.05 \quad (1)$$

where F_{exec} is the Frame execution time and *ifi* the frame length. The value of 1.05 was found measured from the plot.

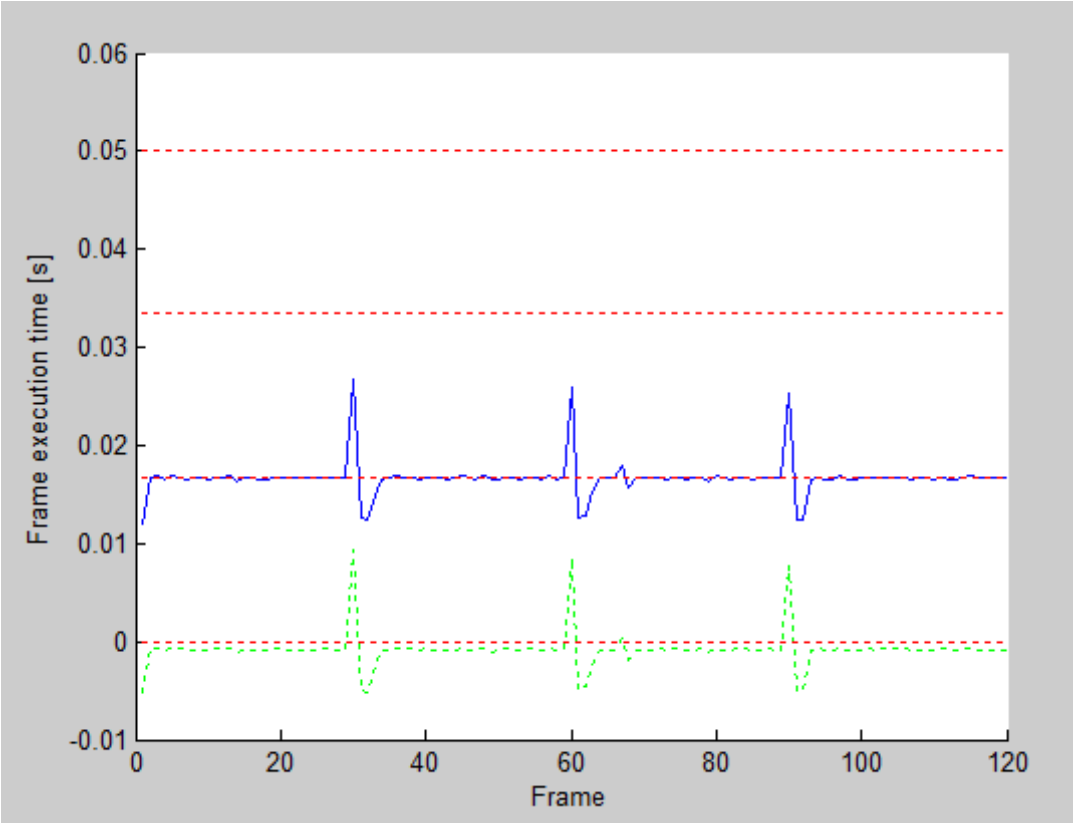

Figure 14: Missed Frames. Frame execution time (blue), value 'missed' (dashed green), frame lengths (dashed red)

With this knowledge it is possible to calculate if several frames in a row have been missed. Assume N is the number of skipped frames, then

$$N = \left\lfloor \frac{F_{exec}}{ifi*1.05} \right\rfloor - 1 \quad (2)$$

$$\leftrightarrow$$

$$N = \left\lfloor \frac{M+ifi*1.05}{ifi*1.05} - 1 \right\rfloor = \left\lfloor \frac{M}{ifi*1.05} \right\rfloor \quad (3)$$

6.2.2 Frame computations

The update frequency of the monitor used, in the case of this project 160 Hz, gives an upper limit to how much computation time each frame is allowed. At 160 Hz each frame is 6.25 ms long. Depending on the jitter from the OS and some added margin it is possible to calculate how much computation time each frame is allowed. This time was measured empirically for the final version of the code, as seen in 6.4.2.

6.2.3 The animation loop

The drawing of the stimulus to the monitor is handled by a set of different display functions, each written for a different stimulus type. The functions start by getting the stimulus parameters from the user interface, and reformat them to be easier to handle. After that some pre-computations are made, such as creating textures and setting the frame rate.

The critical section in the stimulus display function is the animation loop, i.e. the loop in which the frames of the stimulus are drawn to the monitor. The setup of this loop greatly affects the performance and great care should be put into making sure that it runs smoothly.

There are a number of ways to set up the animation loop with different advantages and drawbacks. The first and perhaps most straightforward way is to set up the loop enclosed by the rendering of the pre and post stimuli, see Figure 15. This makes the code easy to follow but introduces several gaps between the different flips, which may cause some problems due to MatLab initializations.

```
DrawBlankScreen();
Screen('Flip');
WaitSecs(preTime);

for frame = 1:totalFrames
    DrawStimuli();
    DrawTrigger('On');
    Screen('Flip');
end

DrawBlankScreen();
Screen('Flip');
WaitSecs(preTime);

DrawTrigger('Off');
Screen('Flip');
```

Figure 15: Enclosed animation loop. Pre and post stimulus is handled separate from the animation.

Another way to set up the loop is to include the pre and post stim in the same loop as the drawing of the stimuli, see Figure 16.

```
for frame = 1:totalFrames

    if preEndFrame < frame < postStartFrame
        DrawStimuli();
    else
        DrawBlankScreen();
    end

    DrawTrigger('On');
    Screen('Flip');
end

DrawTrigger('Off');
Screen('Flip');
```

Figure 16: Frame based animation loop. The animation runs for a specified number of frames.

During testing it was found that this set-up generates a timing error which grows with time, as seen in Figure 17. The figure shows the rendering of a white square which is supposed to turn black after 17 s. The luminance is measured with a photodiode, which generates the voltage output. Note that the CRT screen is not continuously white but turns black between each displayed frame.

The timing error is caused by the fact that the total number of frames needed for a stimulus is hard to calculate. The frame length is measured when a new Screen is initialized, but the value is not precise enough. If the precision is off by just one microsecond, the total error will still be over one frame in less than one minute. In Figure 17, the timing is off by 5 frames after 17 seconds, meaning the error in measured frame length is about 10 µs.

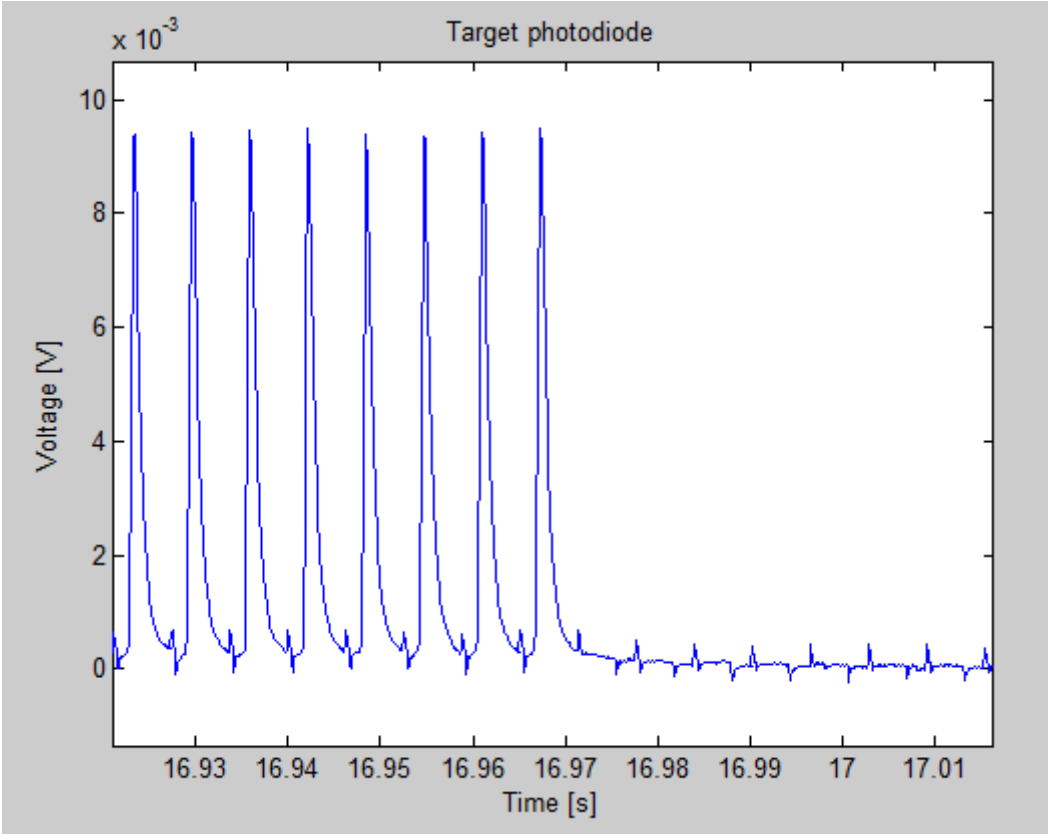

Figure 17: Precision with set number of frames. A photodiode is placed on the screen to measure the timing precision. Each spike to the left corresponds to the drawing of a white frame. The monitor is supposed to be white for 17 seconds. The discrepancy is due to a small difference between the measured and the actual frame rate.

Instead of using a set number of frames it is better to use the internal clock to handle the length of the loop, as seen in Figure 18. With this set-up it is not necessary to obtain the length of a frame at all. Also, any frame drops will not mess up the timing of the experiment. A potential drawback is some loss of control - different trials might differ in one frame due to rounding errors. There is also the chance of still having an increasing timing error in long experiments. This is partly due to the discrete nature of the display device. For example, if a trial is specified to last for one second the closest possible time to display might be 1.003 seconds.

```
while currentTime <= stimRunTime

    if preEndTime < currentTime < postStartTime
        DrawStimuli();
    else
        DrawBlankScreen();
    end

    DrawTrigger('On');
    Screen('Flip');
end

DrawTrigger('Off');
Screen('Flip');
```

Figure 18: Time based animation loop. The animation runs for a specified time.

This set-up is the one being used in the latest build and it has proven to be very robust, as seen in section 6.4.4.

6.2.4 Timed 'Flip'

It is possible to flip the double buffer with a timestamp specifying at what time the flip should take place, which should reduce the risk of dropping frames. A stripped down version of the animation loop, drawing a simple black bar, was run for 150 trials on Machine 1, one time with and one time without using a timestamp. The results from this experiment can be seen in Figure 19. It is clear that timing the flips significantly increases the performance of the frame rate.

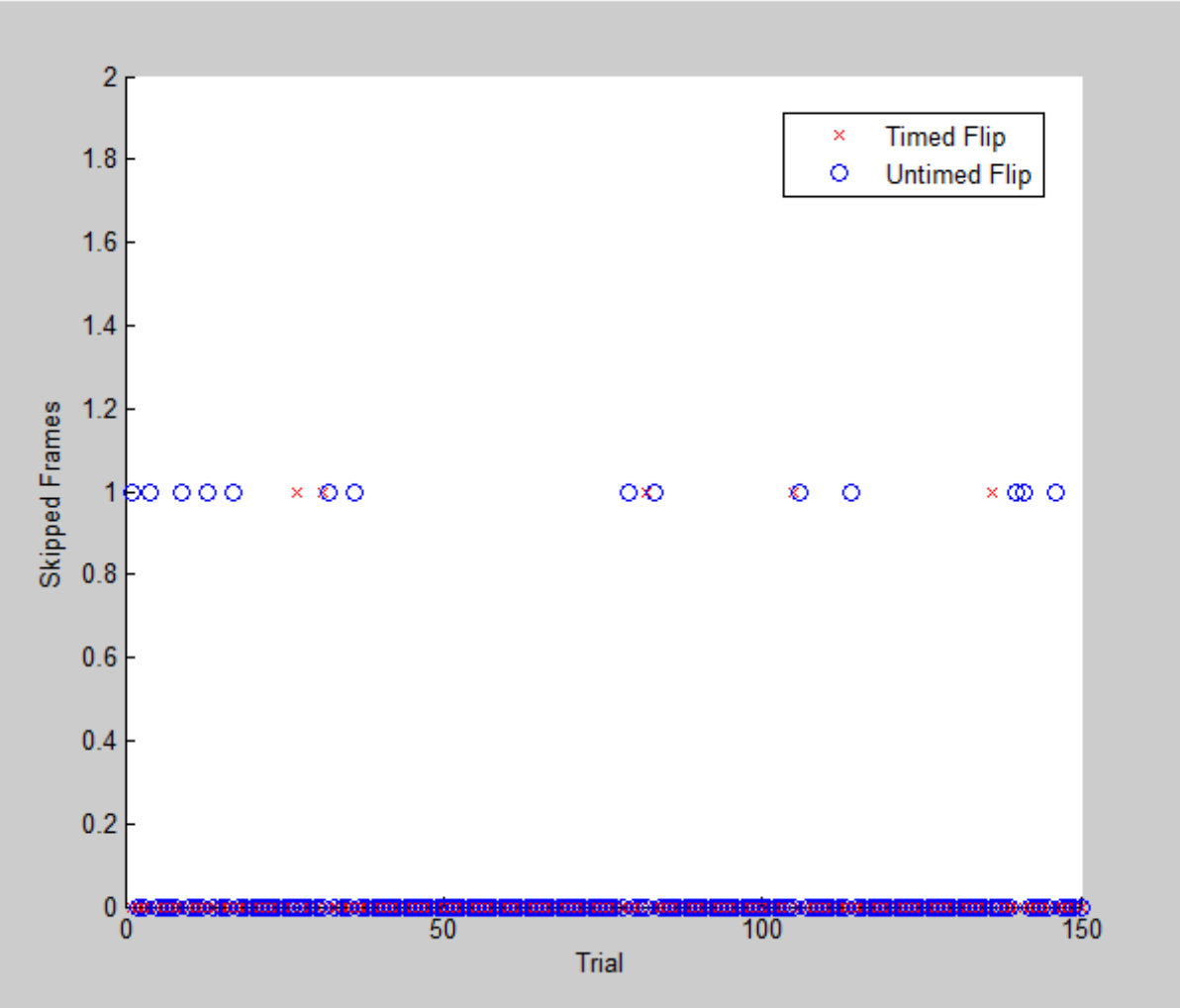

Figure 19: Skipped frames with (red) and without (blue) timestamp.

6.2.5 Different Priority modes

Using PTB-3 it is possible to set the priority of part of a code from within MatLab. To see the difference between the different priority modes, a simple test script was written. In this script a loop is running for a set number of iterations while constantly polling and saving the system time. In the case of a perfect linear execution, the system time will increase linearly. In practice the execution will be interrupted by the OS at somewhat periodic intervals. If we plot the discrete derivative of the polled times these interrupts will show up as spikes. Figure 20 shows a loop with 10 000 iterations, running in normal priority on Machine 1. It is easy to see that the interrupts are fairly periodic with a few larger spikes. Figure 21 shows the same experiment on the same machine, but using the maximum priority mode. The interrupts here are sparser, meaning that the process gets a bigger slice of the CPU time, i.e. it is interrupted less often. Also, the baseline is lower, which indicates that the actual loop is running at a higher speed even when not interrupted, which can be explained with the turbo boost technique, allowing for momentarily higher clock speed, used by the i7 processor.

To make sure that the experiment was not influenced by the current workload of the system, it was repeated a large number of times in random order and the results averaged (data not shown).

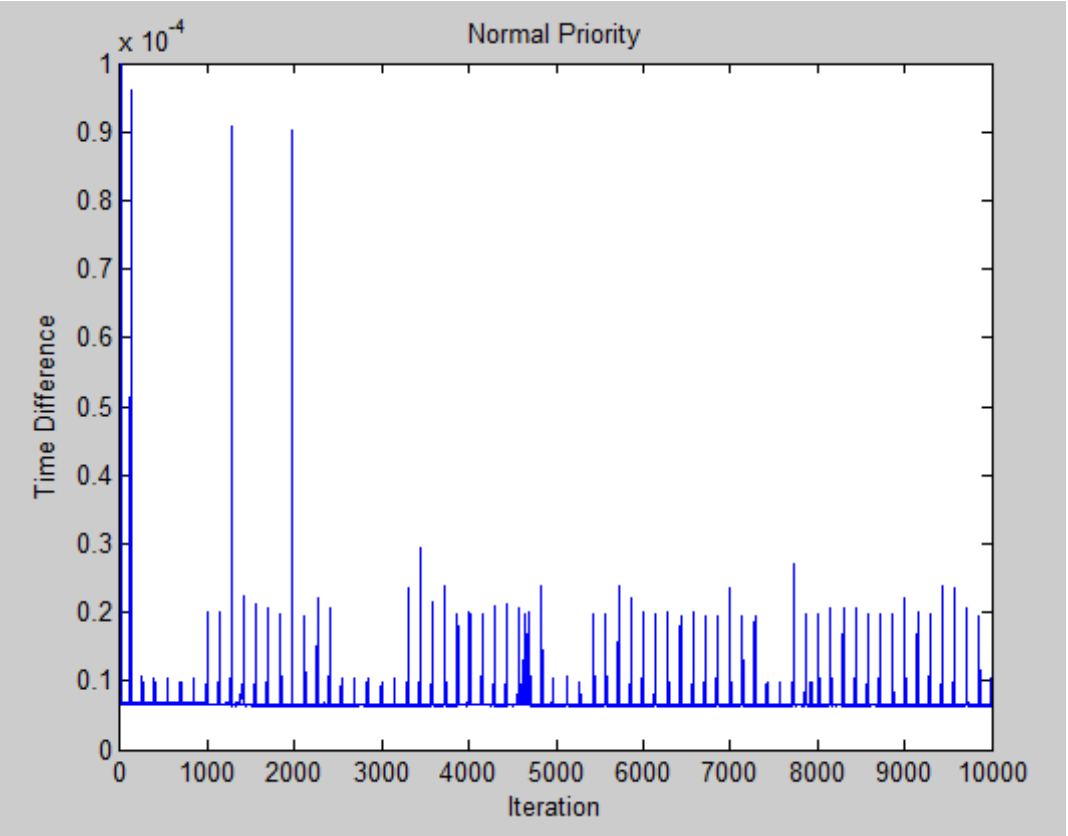

Figure 20: Time difference between iterations in normal priority mode. The spikes correspond to external disturbances from the operating system.

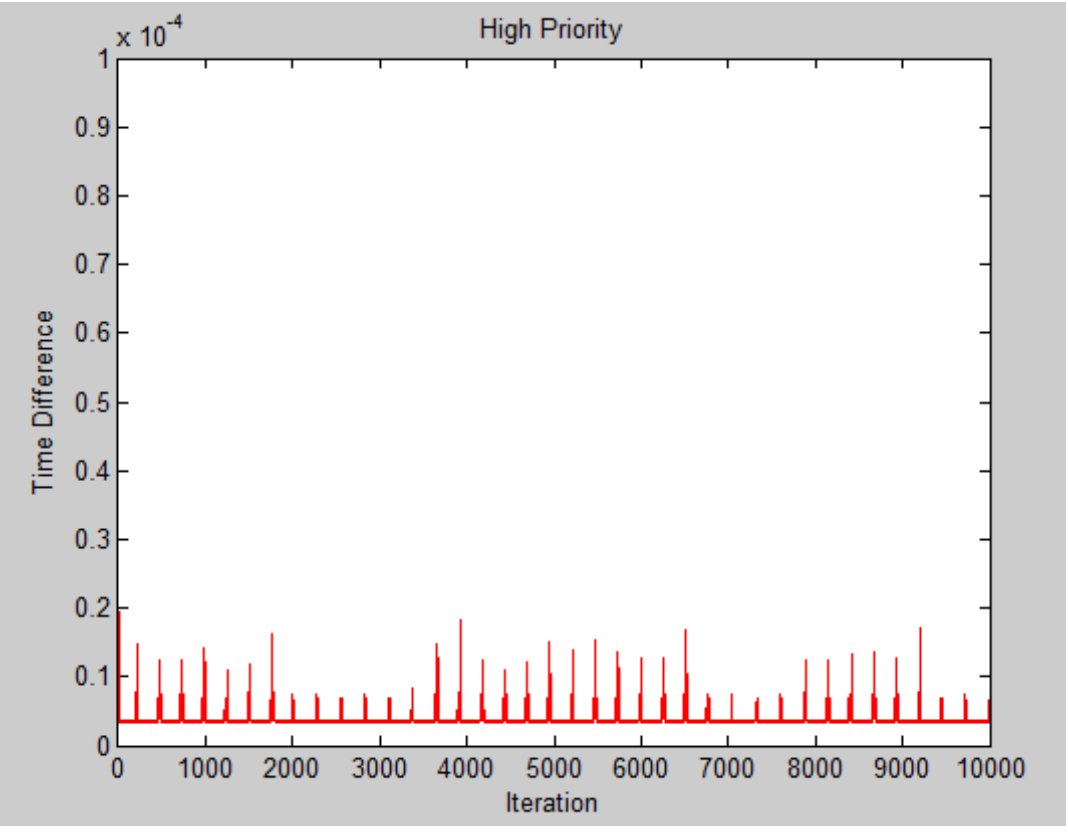

Figure 21: Time difference between iterations in high priority mode. The spikes correspond to external disturbances from the operating system.

6.3 GUI

The user interface has been updated continuously during the entire project as new features have been added. An early version can be seen in Figure 22. This version is able to run simple experiments, but lacks a number of features, such as support for manipulating large data sets and displaying multiple targets.

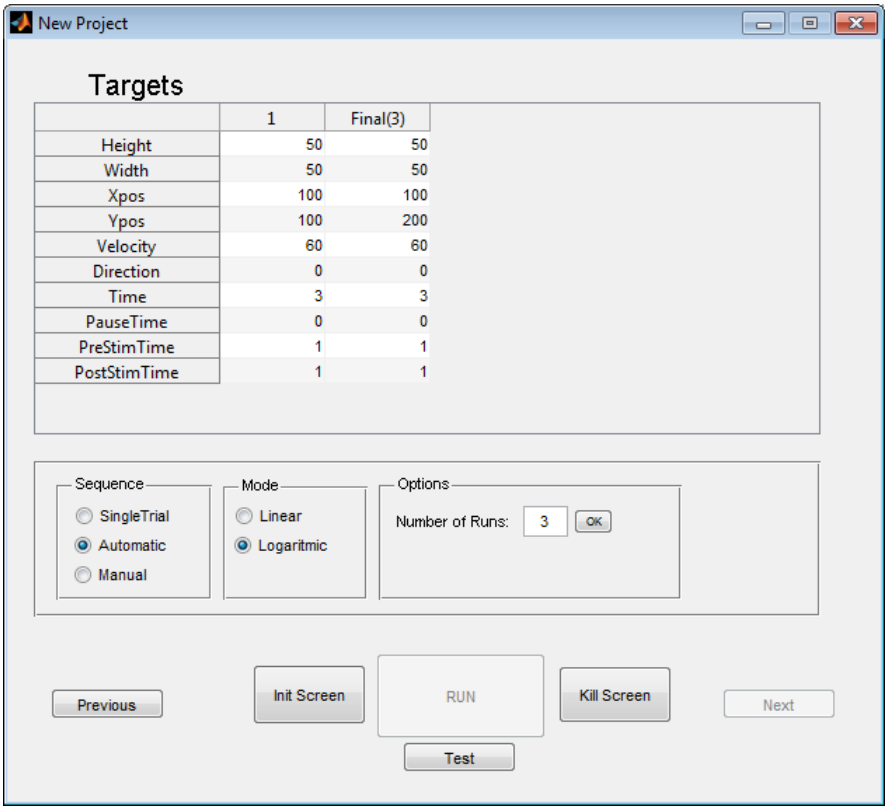

Figure 22: FlyFly beta 1.1. The first version used in live experiments.

The latest version at the moment of writing can be seen in Figure 23. There are a number of differences between the different versions, most notably the change from calculating all parameters automatically to setting them manually with semi-automatic options.

6.3.1 Sequencer

FlyFly uses a table based sequencer which allows any parameter in any trial to be set manually. It is also possible to use several objects in a stimulus, in which case each object can be manipulated individually. This has proven to be a very robust set-up, which allows easy manipulation of the different parameters. It is also easy to add new features to this set-up for further enhancement of the usability of the interface.

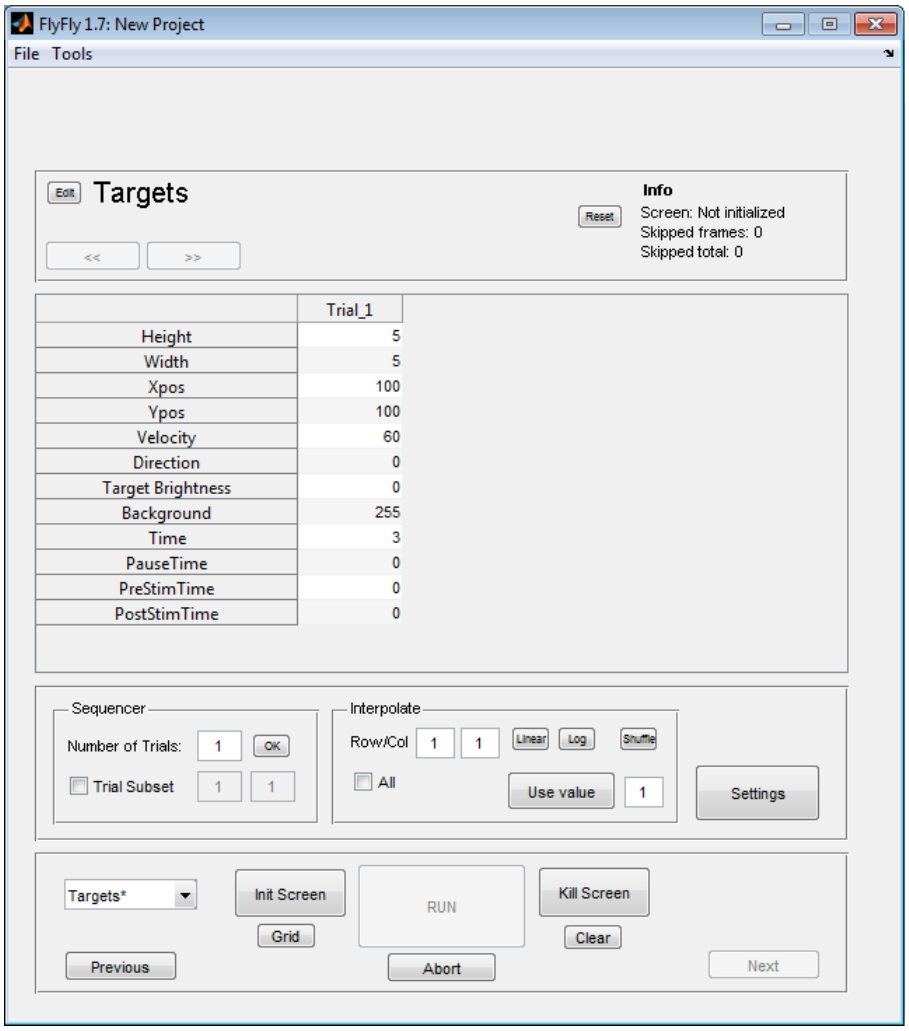

Figure 23: FlyFly beta 1.7.2. The latest FlyFly version at the time of writing.

6.4 Final Code

The most recent version of the code has been evaluated and tested to see how well it works in actual experiments. Much evaluation has been handled by feedback from the users. In these cases, fixes and solutions have been implemented quickly without formal evaluation.

6.4.1 Statistics

At the moment FlyFly has been used in experiments for some months without any major crashes. Because of the constant updates some issues have been encountered when trying to use old saved files with newer versions of the program.

Statistics from all experiments between 11th of June and 13th of august can be seen in Figure 24. Most experiments consist of between 10 and 20 trials lasting for about 2 seconds each. Some statistics to note can be seen in Table 5.

Table 5: Statistics

Experiments in total	920
Experiments without frame drops	732 (79.57%)
Experiments with fps of 159 or better	897 (97.50%)
Mean fps	159.5928
Mean fps (excluding values under 150)	159.9332

Even though most experiments run without or with few dropped frames there are a few exceptions with really horrendous results. Looking at the parameters used in these experiments shows nothing unusual. This means the bad results are likely due to external disturbances, such as the user manipulating the GUI at the same time as the experiment is run.

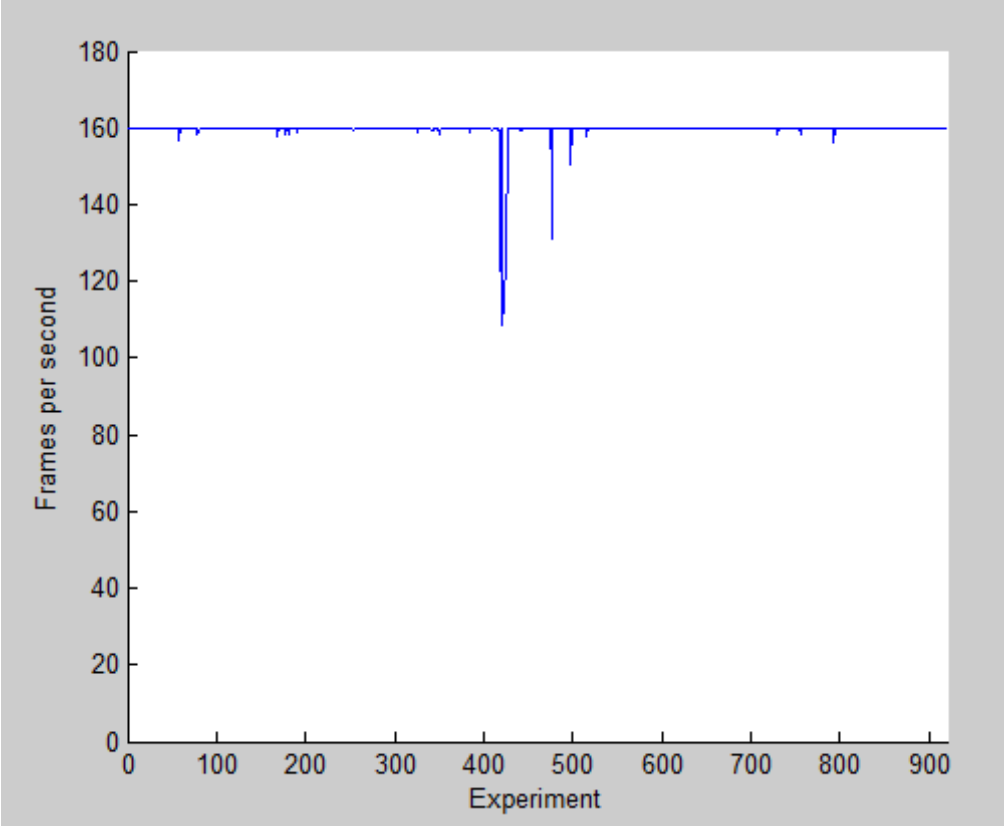

Figure 24: Frame rate in real experiments. Experiments plotted in alphabetical order.

6.4.2 Computational margins

To evaluate the computational margins of the final version, a few typical stimuli used in actual experiments were chosen. A series of trials with identical parameters was set up for each stimulus. Before the drawing of each frame a pause was added - this pause was increased in size for each trial in order to see the effects on performance.

Figure 25 shows the effect on a target stimulus used to map receptive fields, a small target moving across the screen for two seconds. With a pause time of 4.95 ms or more, the frame rate starts to decrease rapidly.

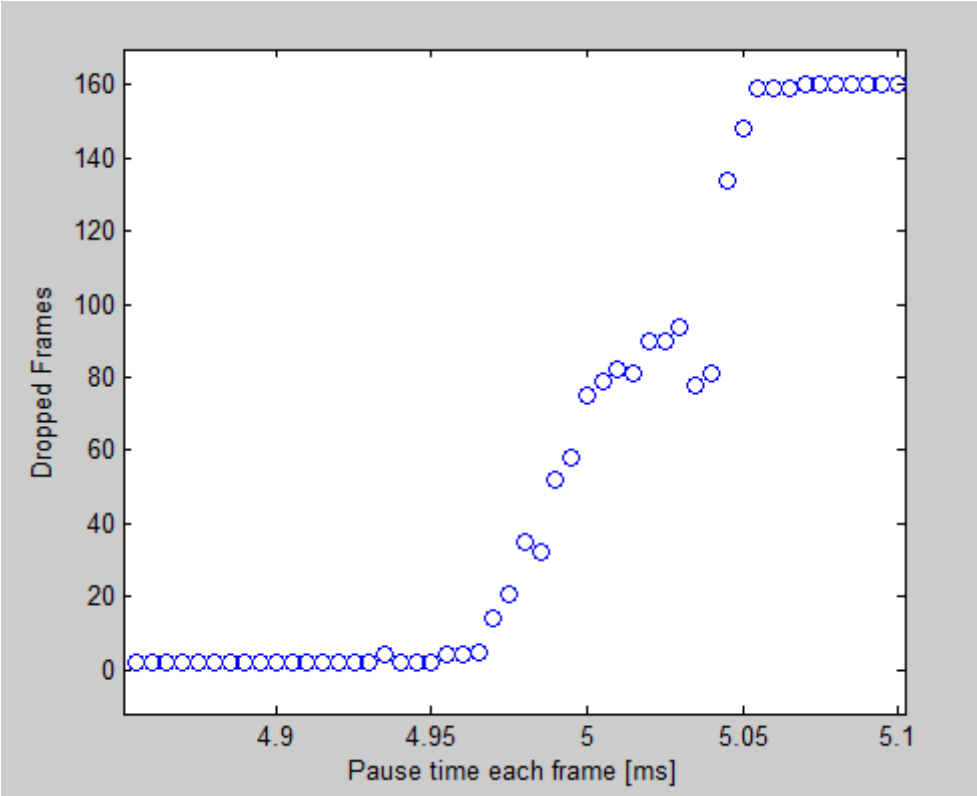

Figure 25: Computational margins in the target stimulus.

6.4.3 Unexpected values

Some care has been taken to make sure that disallowed values, such as characters or fractions instead of integrals, do not make the code crash. However, no extensive stress testing has been done in this area and a careless user might experience some issues.

6.4.4 Precision

To evaluate the precision in time, a second photodiode was placed on the monitor and connected to the data acquisition equipment. After a pre-stim of 1 s, a large black target was drawn straight in front of the second photodiode. It can be seen that in this case, it is not possible to obtain a much better precision. Depending on the desired rending time the precision may be off by up to 3 ms as a result of the actual frame length.

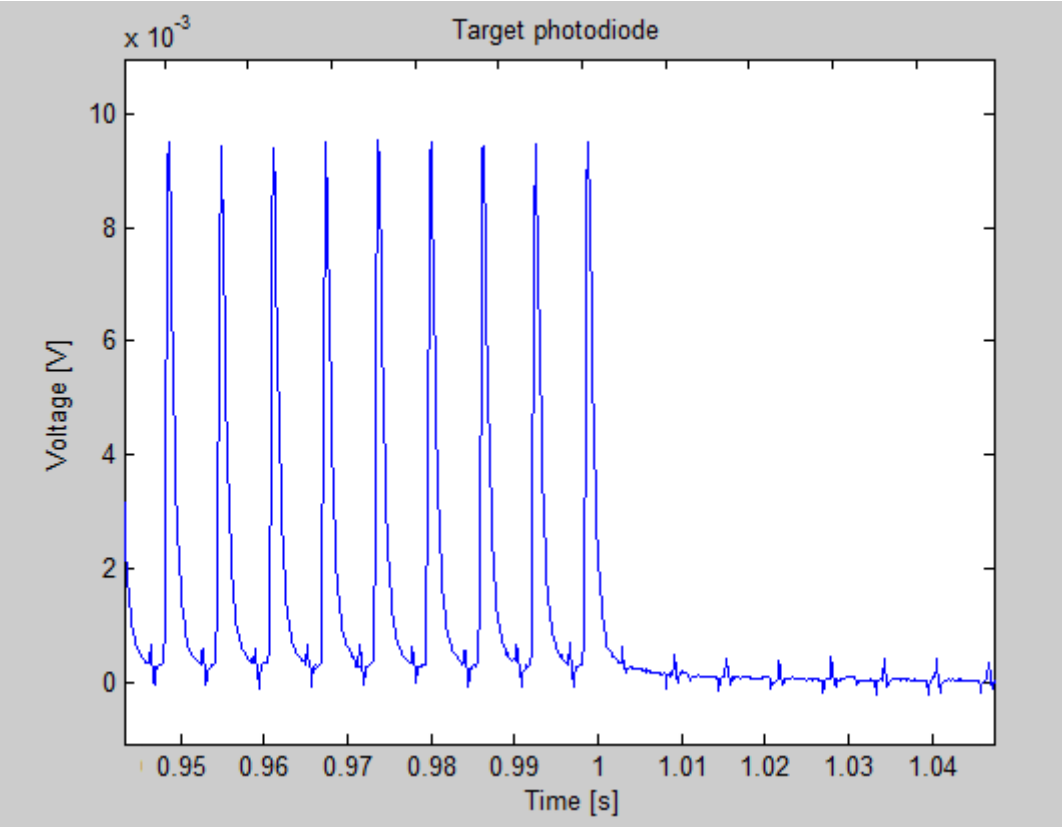

Figure 26: Frame precision. A photodiode is placed on the screen to measure the timing precision. Each spike to the left corresponds to the drawing of a white frame. A black target appears after 1 s.

6.4.5 Usability

The usability of FlyFly has been briefly evaluated with a poll directed to its current users. The poll was in free-form, asking the users to name the main advantages and drawbacks of FlyFly and explain its use in future projects.

"FlyFly is user friendly and intuitive to use. It is easy to understand the code and to design experiments. I will absolutely use FlyFly in future experiments, for example in motion adaption with flicker adaption and multiple target experiments. These are not possible to do with the current version of VisionEgg."

"It is easy to change between different experiment set-ups, and easy to learn. It is also easy to change to different settings and modulate for individual test trials. I will use FlyFly to design experiments for target detection in neurons, i.e. to display a small target moving across the screen while checking the response in fly neurons."

Through the entire development of FlyFly there has been lots of feedback from the users, both on general features and smaller issues. This means many small changes have been made and incorporated almost daily, resulting in a very user-friendly end product.

7 Discussion

7.1 Changes from timeline and shift of focus

While the original time schedule of the project remained intact, the actual focus of the project shifted a bit from what was originally planned. It was anticipated that most of the work would be spent on optimizing the stimuli and making the performance as close to real time as possible. Instead, a large amount of time has been spent improving the interface of the program to make it possible to actually work with the product.

All the original planned features have been implemented on time, though the backlog has continued to grow for the entire project. This is not really a problem, since the original plan was to make a basic framework that is easy to add upon. For a brief overview of possible improvements, see section 7.4.

7.2 The design process

As discussed earlier, working with agile methods has many advantages, such as assuring that there will always be a working product in the end and allowing the customer to constantly see the progress of the project. However, adding features one at a time is not always the best way to reach the end product. If you have a clear picture from the beginning of exactly what you wish to achieve, it is possible to tailor the solution. For example, some set of features might just be different special cases of a general feature. This general feature is probably easier to spot if the full set of sub features are known from the beginning and not implemented one at a time.

A concrete example of this is the set-up of a stimulus combining features from several simpler stimuli, such as a target moving across an image background. From the beginning of the project it was not clear that this is a general and very useful feature, and therefore it was implemented as a special case. A better way to do this may have been to generate a huge list of all thinkable stimuli that could be wanted, both now and in the future. These stimuli could then be grouped and sorted according to common features. Instead of having three separate stimuli such as *Target*, *Mouse Target* and *Rolling Image* these could all have been implemented as general *Image* stimulus, allowing a chosen image to be drawn to an arbitrary position.

Of course it is possible to have a clear idea of the end goal, while still working to get there one feature at a time. A problem is that the goal is often unclear to the customer when the project is started, and even if it is not, the goal might still change during the development. Unexpected problems emerging might also force some features to be scrapped. At the same time some problematic features might be easier than expected to implement which might give room for new possibilities.

7.3 Problems

Overall the project has been running smoothly, though there are a few distinct issues that have caused some problems, or if you prefer, challenges.

7.3.1 Development platform

The biggest problem has been developing the program on a different computer with different hardware and OS than the computer used to run the actual stimuli. MatLab is supposed to be

platform independent and most functions work fine on both the Windows and the OSX versions of MatLab. This is not true for all features though. The biggest problem was the *interruptible* property of callback functions which worked perfectly on the Windows machines but refused to work at all on the Macs. After some research it appears that the MatLab does not implement a "true" interrupt, but allows the process to be interrupted when it performs one of a number of operations, such as *pause* and *waitfor*. It is unclear why the behavior is different between the two operating systems.

There is at least one benefit of working on two different machines. The windows machine used for most of the development has an inbuilt graphics card with significantly lower performance than the stimulus machine. Of course this is far worse if using the machine in an actual experiment, but it actually makes optimizing the code easier, since it is much easier to notice increased performance without any special measurements taken. As an example, the animation using untimed flip plays quite smoothly on the Mac, though with some dropped frames, but very jerkily at the Windows machine.

Initializing the Screen object is much smoother and faster on the Mac. It is hard to say if the reason is improved hardware or different OS. PTB was originally developed for the Mac so it is not that farfetched to assume that this is the reason of the improved performance of *Screen*. It would still be interesting to compare the performance between two machines with identical hardware set-up, running different OS.

7.3.2 Interference with other projects

Another big problem relates to the actual usage of the machines. Other experiments conducted use other software which in some cases requires different screen settings. This has on some occasions led to bugs being discovered without any actual changes made to the code or to experiment settings. Discovering these bugs and mistakes is of course a good thing, though it is frustrating to have problems appearing for no apparent reason.

7.4 Possible Improvements

While most of the project goals have been fulfilled there are still a number of possible improvements and additions that would be nice to implement.

7.4.1 Improvements to current code

It might be beneficial to implement a network server mode, in which the user interface and the stimuli run on separate computers. This allows the stimuli computer to focus all system resources solely on drawing stimuli to a single screen, instead of drawing both the stimuli and the user interface at the same time. This was in fact planned to be included in this first version, but was dropped because the performance of the dual screen set-up was deemed to be good enough. It would, however, be nice to compare the different implementations. A network server might be needed for more advanced stimuli, or to be able to run the current ones on a slower machine.

On a stimuli level, a natural next step would be to implement perspective corrections. This means that instead of drawing a straight line on the screen it is drawn curved, in order to appear straight from the fly´s point of view. OpenGL has a lot of these transforms implemented for use in 3D-graphics which might facilitate the MatLab implementation.

It would also be nice to implement the option of using motion blur to enhance the target stimuli.

7.4.2 Structural changes

The current set-up lets the GUI set a series of parameters and then sends them to the stimulus function, which manages the loop and the drawing. A slightly different set-up is to separate the loop from the actual drawing, which would make it possible to call several different stimuli in the same experiment. This way several stimuli can be arbitrarily combined. This would make it easy to add new functionality, though building the actual experiment might not be as intuitive.

It would also be useful to somehow let the user modify which parameters in an experiment it is possible to change. This could be implemented quite easily by making it possible to change all parameters used in an experiment, and then allowing the user to hide those not relevant to his set-up.

Making some kind of general settings GUI would make it much easier to implement new stimuli, since no time is wasted re-writing code.

7.4.3 New stimuli

Needless to save, it is possible to implement a close to infinite number of new stimuli. Adding new variants to existing stimuli, such as a making a target with variable speed and more complex movement paths, should be easy even for someone without programming background.

A big and useful addition, that is very different from the stimuli already implemented, would be the possibility to play movies and image sequences of arbitrary frame rates.

Future projects might include implementing 3D graphics, which would be useful in experiments dealing with translational and rotational movements.

7.4.4 Merge with data acquisition

If the data acquisition part was implemented in MatLab, that would open up a lot of new possibilities. For example it would be easy to set semi-automatic plotting of the results directly after each experiment, something that would be somewhat complicated to implement at the moment.

7.5 Conclusion

The goal of this project was to create a *stable*, *useable* and *modifiable* framework. The current version is able to construct a multitude of different stimuli, and display them with few or no dropped frames. The user interface is very intuitive, but still allows powerful manipulation of input parameters. Finally, it is easy to add new modules to the framework or make changes to the existing ones.

FlyFly is currently being used in the lab and has been so since early June. It is very easy to install, learn and get started with, which hopefully might result in other labs across the globe using it in experiments.

8 References

Agile Alliance, 2001, http://www.agilealliance.org/the-alliance/the-agile-manifesto/ [retrieved 16 august]

Brainard, D., 1997, The Psychophysics Toolbox, *Spatial Vision*, Vol. 10, No. 4, 433-436

Franceschini, N., Viollet, S., Ruffier, F., 2004, Insect based autopilots for micro-air vehicles, *Insect Sensors and Robotics*, Brisbane

Huo, M., Verner, J., Zhu, L., Babar, M. , 2004, Software Quality and Agile Methods, *Annual International Computer Software and Applications Conference*

Land M.F., Collett T.S., 1974, Chasing behavior of house flies (Fannia canicularis). *Journal of Comparative Physiology,* A 89:331–357

Land, M.F., Eckert, H., 1985, Maps of the acute zones of fly eyes, *Journal of Comparative Physiology,* A 156:525-538

Fry, S., Rohrseitz, N., Straw, A., Dickinson, M., 2009, Visual control of flight speed in Drosophila melanogaster, *Journal of Experimental Biology*, 212:1120-1130

Peirce, J., 2006, PsychoPy—Psychophysics software in Python, *Journal of Neuroscience Methods*, 162:8–13

Pelli, D., 1997, The VideoToolbox software for visual psychophysics: transforming numbers into movies. *Spatial Vision* 10, 437-442.

Srinivasan, M., Wootton, R., Whitten, M., 2004, Foreword , *Insect Sensors and Robotics*, Brisbane

Straw, A., 2008, Vision Egg: an open-source library for realtime visual stimulus generation, *Frontiers in Neuroinformatics,* volume 2

Sumeet Moghe, 2009, http://www.learninggeneralist.com/2009/06/agile-elearning-design-manual-think.html [retrieved 13 august 2010]

Tatler, B., O'Carroll, D.C., Laughlin, S.B., 2000, Temperature and the temporal resolving power of fly photoreceptors, *Journal of Comparative Physiology,* A 186, 399–407.

Yeates, D.K., Wiegmann, B.M., 1999, Congruence and controversy: toward a higher level phylogeny of diptera. *Annual Review of Entomology* 44:397–428

Wagner H (1986) Flight performance and visual control of flight of the free flying house fly (Musca domestica). II. Pursuit of targets. *Philosophical Transactions of the Royal Society London* B 312:553–579

9 Acknowledgements

Working with the Motion Vision group has been extremely instructive and a great deal of fun. I have learned a lot about neuroscience, I have improved my communication and writing skills, I am heaps better with MatLab, Guide and Psychophysics toolbox, and I learned a lot about how to plan and organize a big independent project.

My project FlyFly has been received with much enthusiasm and encouragement. It has been used for several weeks with great success and will hopefully contribute to the making of several exciting papers. I hope to be able to continue working on the project after this thesis is finished, and add some of the features discussed earlier.

I´d like to thank my supervisor Karin Nordström for valuable input, feedback and suggestions on how to improve my project.

I´d also like to thank my assessor Justin Pearson for taking the time with this project.

Finally, big thanks to the people in the Motion Vision Group for helping me with all the nasty neurobiological stuff as well as making my stay in the group thoroughly enjoyable. Thank you Sharn, Frank and Angela!

Jonas Henriksson 2010

10 Appendices

10.1 Original project description

Visual stimulus development

Many animals, including humans, are largely guided by vision: we use self-generated optic flow to navigate through the surround and to perform common tasks such as staying on a straight path when driving a car, riding a bicycle, or going for a morning run. We can also detect objects that move relative to the remaining visual surround, such as a ball or a bird, even if we are moving ourselves and thereby generating wide-field optic flow across our retina. The visual system handles enormous data sets in real time and the information flow in the vertebrate optic nerve has been compared to the transmission rate through an Ethernet cable.

This project suits a student with a background in computer science. It involves developing, streamlining and optimizing software used for displaying visual stimuli used in motion vision research. The student will work in close collaboration with neurophysiology students to ensure that the developed software is 1) functional for the hypotheses addressed, and 2) user friendly for scientists from other backgrounds than programming. The student will gain extensive training using MatLab for generating visual stimuli, in particular using the Psychophysics toolbox.

The student will develop software for displaying novel visual stimuli. While we currently used a python based program (VisionEgg, www.visionegg.org), the software limits possible stimuli to those currently present in different GUIs. Therefore, the project student will develop a MatLab based system for displaying visual stimuli, which should be relatively easy to modify and adapt by biology students with minimal programming training.

The student will be collaborating closely with the end users of the developed software – namely students in physiology. This will provide an additional benefit of giving the student an insight into the problems asked in a neurophysiological lab. The biology students will continuously be testing the developed software during experiments to ensure that the software is developed in a user-friendly manner.

The lab has weekly lab meetings, and journal clubs, and the student will be expected to take an active part in these. The lab is still quite small, ensuring that the student will have the opportunity to present his work, and an appropriate accompanying paper every 4 weeks (on average). The literature should reflect questions asked in his project. At the end of the project, the student is expected to deliver documentation of his software. He will also present his project with a seminar at the Department of Neuroscience, and submit a written thesis.

10.2 Project Timeline

Week	Subject	Deadline
v 11	Introduction and basic ptb3 and GUI functions	
v 12		
v 13	Development of visual stimuli framework	
v 14		
v 15		
v 16	(vacation)	
v 17		Start betatesting of framework
v 18		
v 19	Extensions to the framework	
v 20		
v 21	(project halftime)	Basic framework operational
v 22		
v 23	Start on report	
v 24	(vacation)	
v 25		
v 26	(vacation)	
v 27	Evaluation of framework	First draft of report
v 28		
v 29	Documentation	Extended framework operational
v 30		Documentation of framework
v 31	Work on report and presentation	
v 32		
v 33		First hand-in of report (Friday)
v 34	Corrections and additions to report	
v 35		